

mavtables

0.2.1

Generated by Doxygen 1.8.14

Contents

1	mavtables	1
2	Configuration	3
3	Makefile Targets	13
4	MAVTables User Manual	15
5	GNU General Public License	21
6	Contributing	27
7	mavtables	29
8	Module Index	33
8.1	Modules	33
9	Namespace Index	35
9.1	Namespace List	35
10	Hierarchical Index	37
10.1	Class Hierarchy	37
11	Class Index	39
11.1	Class List	39
12	File Index	41
12.1	File List	41

13 Module Documentation	45
13.1 Configuration functions.	45
13.1.1 Detailed Description	45
13.1.2 Function Documentation	45
13.1.2.1 operator<<()	45
13.1.2.2 print_node()	46
13.2 Macros	48
13.2.1 Detailed Description	48
13.2.2 Macro Definition Documentation	48
13.2.2.1 PACKED	48
13.3 MAVLink Library and Helpers	49
13.3.1 Detailed Description	49
13.3.2 Function Documentation	49
13.3.2.1 id()	49
13.3.2.2 name()	50
13.4 Utility Functions	52
13.4.1 Detailed Description	52
13.4.2 Function Documentation	52
13.4.2.1 append() [1/2]	52
13.4.2.2 append() [2/2]	53
13.4.2.3 operator<<()	53
13.4.2.4 str()	54
13.4.2.5 to_bytes()	54
13.4.2.6 to_lower()	55

14 Namespace Documentation	57
14.1 config Namespace Reference	57
14.1.1 Function Documentation	57
14.1.1.1 parse()	57
14.2 logger Namespace Reference	58
14.2.1 Function Documentation	58
14.2.1.1 heartbeat()	59
14.2.1.2 main()	59
14.2.1.3 parse_args()	60
14.2.1.4 signal_handler()	60
14.2.1.5 start_heartbeats()	60
14.3 mavlink Namespace Reference	61
14.4 packet_scripter Namespace Reference	61
14.4.1 Function Documentation	61
14.4.1.1 main()	61
14.4.1.2 parse_args()	62
14.4.1.3 parse_file()	62
14.4.1.4 parse_line()	63
14.4.1.5 send_packet()	63
14.4.1.6 start_connection()	64
14.5 packet_v1 Namespace Reference	64
14.5.1 Function Documentation	64
14.5.1.1 header()	65
14.5.1.2 header_complete()	65
14.5.1.3 packet_complete()	65
14.5.2 Variable Documentation	66
14.5.2.1 CHECKSUM_LENGTH	66
14.5.2.2 HEADER_LENGTH	66

14.5.2.3 START_BYTE	66
14.5.2.4 VERSION	66
14.6 packet_v2 Namespace Reference	67
14.6.1 Function Documentation	67
14.6.1.1 header()	67
14.6.1.2 header_complete()	68
14.6.1.3 is_signed()	68
14.6.1.4 packet_complete()	68
14.6.2 Variable Documentation	68
14.6.2.1 CHECKSUM_LENGTH	69
14.6.2.2 HEADER_LENGTH	69
14.6.2.3 SIGNATURE_LENGTH	69
14.6.2.4 START_BYTE	69
14.6.2.5 VERSION	69
15 Class Documentation	71
15.1 Accept Class Reference	71
15.1.1 Detailed Description	73
15.1.2 Constructor & Destructor Documentation	73
15.1.2.1 Accept() [1/2]	73
15.1.2.2 Accept() [2/2]	73
15.1.3 Member Function Documentation	74
15.1.3.1 action()	74
15.1.3.2 clone()	75
15.1.3.3 operator"!="()	75
15.1.3.4 operator==()	76
15.1.3.5 print_()	77
15.1.4 Member Data Documentation	77

15.1.4.1	condition_	77
15.2	Action Class Reference	78
15.2.1	Detailed Description	79
15.2.2	Member Enumeration Documentation	79
15.2.2.1	Option	79
15.2.3	Constructor & Destructor Documentation	79
15.2.3.1	Action() [1/2]	80
15.2.3.2	Action() [2/2]	81
15.2.4	Member Function Documentation	81
15.2.4.1	action()	82
15.2.4.2	make_accept()	82
15.2.4.3	make_continue()	83
15.2.4.4	make_default()	84
15.2.4.5	make_reject()	85
15.2.4.6	operator=() [1/2]	86
15.2.4.7	operator=() [2/2]	87
15.2.4.8	priority() [1/2]	87
15.2.4.9	priority() [2/2]	88
15.2.5	Friends And Related Function Documentation	89
15.2.5.1	operator"!="()	89
15.2.5.2	operator<<()	90
15.2.5.3	operator==()	91
15.3	AddressPool< TC > Class Template Reference	92
15.3.1	Detailed Description	93
15.3.2	Constructor & Destructor Documentation	93
15.3.2.1	AddressPool()	93
15.3.2.2	~AddressPool()	94
15.3.3	Member Function Documentation	94

15.3.3.1 add()	94
15.3.3.2 addresses()	95
15.3.3.3 contains()	95
15.4 App Class Reference	96
15.4.1 Detailed Description	96
15.4.2 Constructor & Destructor Documentation	96
15.4.2.1 App()	96
15.4.3 Member Function Documentation	97
15.4.3.1 run()	97
15.5 Call Class Reference	97
15.5.1 Detailed Description	100
15.5.2 Constructor & Destructor Documentation	100
15.5.2.1 Call() [1/2]	100
15.5.2.2 Call() [2/2]	101
15.5.3 Member Function Documentation	101
15.5.3.1 action()	102
15.5.3.2 clone()	103
15.5.3.3 operator"!="()	103
15.5.3.4 operator==()	104
15.5.3.5 print_()	104
15.5.4 Member Data Documentation	105
15.5.4.1 condition_	105
15.6 Chain Class Reference	105
15.6.1 Detailed Description	106
15.6.2 Constructor & Destructor Documentation	106
15.6.2.1 Chain() [1/3]	106
15.6.2.2 Chain() [2/3]	107
15.6.2.3 Chain() [3/3]	107

15.6.2.4 ~Chain()	108
15.6.3 Member Function Documentation	108
15.6.3.1 action()	108
15.6.3.2 append()	109
15.6.3.3 name()	110
15.6.3.4 operator=() [1/2]	110
15.6.3.5 operator=() [2/2]	111
15.6.4 Friends And Related Function Documentation	111
15.6.4.1 operator!=	111
15.6.4.2 operator<<	112
15.6.4.3 operator==	112
15.7 ConfigParser Class Reference	113
15.7.1 Detailed Description	115
15.7.2 Constructor & Destructor Documentation	115
15.7.2.1 ConfigParser() [1/3]	115
15.7.2.2 ConfigParser() [2/3]	116
15.7.2.3 ConfigParser() [3/3]	116
15.7.3 Member Function Documentation	116
15.7.3.1 make_app()	117
15.7.3.2 operator=() [1/2]	117
15.7.3.3 operator=() [2/2]	117
15.7.4 Friends And Related Function Documentation	117
15.7.4.1 init_chains()	117
15.7.4.2 operator<<	118
15.7.4.3 parse_action()	119
15.7.4.4 parse_chain()	120
15.7.4.5 parse_condition()	121
15.7.4.6 parse_filter()	122

15.7.4.7	parse_interfaces()	123
15.7.4.8	parse_serial()	124
15.7.4.9	parse_udp()	125
15.7.5	Member Data Documentation	126
15.7.5.1	in_	126
15.7.5.2	root_	126
15.8	Connection Class Reference	126
15.8.1	Detailed Description	127
15.8.2	Constructor & Destructor Documentation	127
15.8.2.1	Connection()	128
15.8.2.2	~Connection()	128
15.8.3	Member Function Documentation	128
15.8.3.1	add_address()	129
15.8.3.2	next_packet()	129
15.8.3.3	send()	130
15.8.4	Friends And Related Function Documentation	131
15.8.4.1	operator<<	131
15.9	ConnectionFactory< C, AP, PQ > Class Template Reference	132
15.9.1	Detailed Description	133
15.9.2	Constructor & Destructor Documentation	133
15.9.2.1	ConnectionFactory()	133
15.9.2.2	~ConnectionFactory()	134
15.9.3	Member Function Documentation	134
15.9.3.1	get()	134
15.9.3.2	wait_for_packet()	134
15.10	ConnectionPool Class Reference	135
15.10.1	Detailed Description	136
15.10.2	Constructor & Destructor Documentation	136

15.10.2.1 ~ConnectionPool()	136
15.10.3 Member Function Documentation	136
15.10.3.1 add()	136
15.10.3.2 remove()	137
15.10.3.3 send()	137
15.11 DNSLookupError Class Reference	139
15.11.1 Detailed Description	140
15.11.2 Constructor & Destructor Documentation	140
15.11.2.1 DNSLookupError()	140
15.11.3 Member Function Documentation	140
15.11.3.1 what()	140
15.12 Filesystem Class Reference	141
15.12.1 Detailed Description	141
15.12.2 Member Typedef Documentation	141
15.12.2.1 path	142
15.12.3 Constructor & Destructor Documentation	142
15.12.3.1 ~Filesystem()	142
15.12.4 Member Function Documentation	142
15.12.4.1 exists()	142
15.13 Filter Class Reference	143
15.13.1 Detailed Description	144
15.13.2 Constructor & Destructor Documentation	144
15.13.2.1 Filter() [1/3]	144
15.13.2.2 Filter() [2/3]	144
15.13.2.3 Filter() [3/3]	145
15.13.2.4 ~Filter()	145
15.13.3 Member Function Documentation	145
15.13.3.1 operator=() [1/2]	145

15.13.3.2 operator=() [2/2]	145
15.13.3.3 will_accept()	146
15.13.4 Friends And Related Function Documentation	146
15.13.4.1 operator!=	147
15.13.4.2 operator==	147
15.14 GoTo Class Reference	148
15.14.1 Detailed Description	150
15.14.2 Constructor & Destructor Documentation	150
15.14.2.1 GoTo() [1/2]	150
15.14.2.2 GoTo() [2/2]	151
15.14.3 Member Function Documentation	151
15.14.3.1 action()	152
15.14.3.2 clone()	153
15.14.3.3 operator!=()	153
15.14.3.4 operator==()	154
15.14.3.5 print_()	154
15.14.4 Member Data Documentation	155
15.14.4.1 condition_	155
15.15 If Class Reference	155
15.15.1 Detailed Description	156
15.15.2 Constructor & Destructor Documentation	156
15.15.2.1 If() [1/3]	156
15.15.2.2 If() [2/3]	157
15.15.2.3 If() [3/3]	157
15.15.3 Member Function Documentation	158
15.15.3.1 check()	158
15.15.3.2 from() [1/2]	159
15.15.3.3 from() [2/2]	160

15.15.3.4 operator=() [1/2]	160
15.15.3.5 operator=() [2/2]	160
15.15.3.6 to() [1/2]	161
15.15.3.7 to() [2/2]	161
15.15.3.8 type() [1/2]	162
15.15.3.9 type() [2/2]	163
15.15.4 Friends And Related Function Documentation	164
15.15.4.1 operator!=	164
15.15.4.2 operator<<	165
15.15.4.3 operator==	165
15.16 Interface Class Reference	166
15.16.1 Detailed Description	167
15.16.2 Constructor & Destructor Documentation	167
15.16.2.1 ~Interface()	167
15.16.3 Member Function Documentation	167
15.16.3.1 print_()	167
15.16.3.2 receive_packet()	168
15.16.3.3 send_packet()	168
15.16.4 Friends And Related Function Documentation	169
15.16.4.1 operator<<	169
15.17 InterfaceThreader Class Reference	170
15.17.1 Detailed Description	170
15.17.2 Member Enumeration Documentation	171
15.17.2.1 Threads	171
15.17.3 Constructor & Destructor Documentation	171
15.17.3.1 InterfaceThreader() [1/3]	171
15.17.3.2 InterfaceThreader() [2/3]	172
15.17.3.3 InterfaceThreader() [3/3]	172

15.17.3.4 ~InterfaceThreader()	172
15.17.4 Member Function Documentation	173
15.17.4.1 operator=() [1/2]	173
15.17.4.2 operator=() [2/2]	173
15.17.4.3 shutdown()	173
15.17.4.4 start()	174
15.18 InvalidPacketIDError Class Reference	174
15.18.1 Detailed Description	175
15.18.2 Constructor & Destructor Documentation	175
15.18.2.1 InvalidPacketIDError()	175
15.18.3 Member Function Documentation	176
15.18.3.1 what()	176
15.19 IPAddress Class Reference	176
15.19.1 Detailed Description	177
15.19.2 Constructor & Destructor Documentation	177
15.19.2.1 IPAddress() [1/5]	177
15.19.2.2 IPAddress() [2/5]	178
15.19.2.3 IPAddress() [3/5]	178
15.19.2.4 IPAddress() [4/5]	179
15.19.2.5 IPAddress() [5/5]	180
15.19.3 Member Function Documentation	181
15.19.3.1 address()	181
15.19.3.2 operator=() [1/2]	182
15.19.3.3 operator=() [2/2]	182
15.19.3.4 port()	183
15.19.4 Friends And Related Function Documentation	183
15.19.4.1 dnslookup()	184
15.19.4.2 operator"!="()	184

15.19.4.3 operator<()	185
15.19.4.4 operator<<	186
15.19.4.5 operator<=()	187
15.19.4.6 operator==()	187
15.19.4.7 operator>()	188
15.19.4.8 operator>=()	189
15.20Logger Class Reference	190
15.20.1 Detailed Description	190
15.20.2 Member Function Documentation	191
15.20.2.1 level() [1/2]	191
15.20.2.2 level() [2/2]	191
15.20.2.3 log() [1/2]	192
15.20.2.4 log() [2/2]	193
15.21MAVAddress Class Reference	194
15.21.1 Detailed Description	195
15.21.2 Constructor & Destructor Documentation	195
15.21.2.1 MAVAddress() [1/5]	195
15.21.2.2 MAVAddress() [2/5]	196
15.21.2.3 MAVAddress() [3/5]	196
15.21.2.4 MAVAddress() [4/5]	197
15.21.2.5 MAVAddress() [5/5]	198
15.21.3 Member Function Documentation	199
15.21.3.1 address()	199
15.21.3.2 component()	200
15.21.3.3 operator=() [1/2]	201
15.21.3.4 operator=() [2/2]	201
15.21.3.5 system()	202
15.21.4 Friends And Related Function Documentation	202

15.21.4.1 operator"!=()	202
15.21.4.2 operator<()	203
15.21.4.3 operator<<()	204
15.21.4.4 operator<=()	205
15.21.4.5 operator==()	206
15.21.4.6 operator>()	207
15.21.4.7 operator>=()	207
15.22 MAVSubnet Class Reference	208
15.22.1 Detailed Description	209
15.22.2 Constructor & Destructor Documentation	210
15.22.2.1 MAVSubnet() [1/5]	210
15.22.2.2 MAVSubnet() [2/5]	210
15.22.2.3 MAVSubnet() [3/5]	210
15.22.2.4 MAVSubnet() [4/5]	211
15.22.2.5 MAVSubnet() [5/5]	211
15.22.3 Member Function Documentation	213
15.22.3.1 contains()	213
15.22.3.2 operator=() [1/2]	214
15.22.3.3 operator=() [2/2]	214
15.22.4 Friends And Related Function Documentation	215
15.22.4.1 operator"!="	215
15.22.4.2 operator<<	215
15.22.4.3 operator==	216
15.23 MockCErr Class Reference	217
15.23.1 Detailed Description	217
15.23.2 Constructor & Destructor Documentation	217
15.23.2.1 MockCErr()	218
15.23.2.2 ~MockCErr()	218

15.23.3 Member Function Documentation	218
15.23.3.1 buffer()	218
15.23.3.2 reset()	218
15.24 MockCOut Class Reference	219
15.24.1 Detailed Description	219
15.24.2 Constructor & Destructor Documentation	219
15.24.2.1 MockCOut()	219
15.24.2.2 ~MockCOut()	220
15.24.3 Member Function Documentation	220
15.24.3.1 buffer()	220
15.24.3.2 reset()	220
15.25 Options Class Reference	221
15.25.1 Detailed Description	221
15.25.2 Constructor & Destructor Documentation	222
15.25.2.1 Options()	222
15.25.3 Member Function Documentation	223
15.25.3.1 ast()	223
15.25.3.2 config_file()	223
15.25.3.3 loglevel()	224
15.25.3.4 operator bool()	224
15.25.3.5 run()	225
15.25.4 Friends And Related Function Documentation	225
15.25.4.1 find_config()	225
15.26 Packet Class Reference	227
15.26.1 Detailed Description	229
15.26.2 Member Enumeration Documentation	229
15.26.2.1 Version	229
15.26.3 Constructor & Destructor Documentation	229

15.26.3.1 <code>Packet()</code> [1/3]	229
15.26.3.2 <code>Packet()</code> [2/3]	230
15.26.3.3 <code>Packet()</code> [3/3]	230
15.26.3.4 <code>~Packet()</code>	230
15.26.4 Member Function Documentation	230
15.26.4.1 <code>connection()</code> [1/2]	231
15.26.4.2 <code>connection()</code> [2/2]	232
15.26.4.3 <code>data()</code>	232
15.26.4.4 <code>dest()</code>	234
15.26.4.5 <code>id()</code>	234
15.26.4.6 <code>name()</code>	235
15.26.4.7 <code>operator=()</code> [1/2]	235
15.26.4.8 <code>operator=()</code> [2/2]	235
15.26.4.9 <code>source()</code>	236
15.26.4.10 <code>version()</code>	236
15.26.5 Friends And Related Function Documentation	237
15.26.5.1 <code>operator"!="()</code>	237
15.26.5.2 <code>operator<<()</code>	238
15.26.5.3 <code>operator==()</code>	239
15.27 <code>packet_v1::Packet</code> Class Reference	240
15.27.1 Detailed Description	243
15.27.2 Member Enumeration Documentation	243
15.27.2.1 Version	243
15.27.3 Constructor & Destructor Documentation	244
15.27.3.1 <code>Packet()</code> [1/3]	244
15.27.3.2 <code>Packet()</code> [2/3]	244
15.27.3.3 <code>Packet()</code> [3/3]	244
15.27.4 Member Function Documentation	245

15.27.4.1 connection() [1/2]	245
15.27.4.2 connection() [2/2]	246
15.27.4.3 data()	247
15.27.4.4 dest()	249
15.27.4.5 id()	249
15.27.4.6 name()	250
15.27.4.7 operator=() [1/2]	251
15.27.4.8 operator=() [2/2]	251
15.27.4.9 source()	251
15.27.4.10 version()	252
15.27.5 Friends And Related Function Documentation	252
15.27.5.1 header()	252
15.27.5.2 header_complete()	254
15.27.5.3 operator"!="()	254
15.27.5.4 operator<<()	255
15.27.5.5 operator==()	256
15.27.5.6 packet_complete()	257
15.28 packet_v2::Packet Class Reference	258
15.28.1 Detailed Description	261
15.28.2 Member Enumeration Documentation	261
15.28.2.1 Version	261
15.28.3 Constructor & Destructor Documentation	262
15.28.3.1 Packet() [1/3]	262
15.28.3.2 Packet() [2/3]	262
15.28.3.3 Packet() [3/3]	262
15.28.4 Member Function Documentation	263
15.28.4.1 connection() [1/2]	263
15.28.4.2 connection() [2/2]	264

15.28.4.3 <code>data()</code>	265
15.28.4.4 <code>dest()</code>	267
15.28.4.5 <code>id()</code>	267
15.28.4.6 <code>name()</code>	268
15.28.4.7 <code>operator=() [1/2]</code>	269
15.28.4.8 <code>operator=() [2/2]</code>	269
15.28.4.9 <code>source()</code>	269
15.28.4.10 <code>version()</code>	270
15.28.5 Friends And Related Function Documentation	270
15.28.5.1 <code>header()</code>	270
15.28.5.2 <code>header_complete()</code>	272
15.28.5.3 <code>is_signed()</code>	272
15.28.5.4 <code>operator"!=()</code>	273
15.28.5.5 <code>operator<<()</code>	274
15.28.5.6 <code>operator==()</code>	275
15.28.5.7 <code>packet_complete()</code>	276
15.29 PacketParser Class Reference	277
15.29.1 Detailed Description	278
15.29.2 Constructor & Destructor Documentation	278
15.29.2.1 <code>PacketParser() [1/3]</code>	279
15.29.2.2 <code>PacketParser() [2/3]</code>	279
15.29.2.3 <code>PacketParser() [3/3]</code>	279
15.29.3 Member Function Documentation	279
15.29.3.1 <code>bytes_parsed()</code>	280
15.29.3.2 <code>clear()</code>	280
15.29.3.3 <code>operator=() [1/2]</code>	281
15.29.3.4 <code>operator=() [2/2]</code>	281
15.29.3.5 <code>parse_byte()</code>	281

15.30 PacketQueue Class Reference	282
15.30.1 Detailed Description	283
15.30.2 Constructor & Destructor Documentation	283
15.30.2.1 PacketQueue()	283
15.30.2.2 ~PacketQueue()	284
15.30.3 Member Function Documentation	284
15.30.3.1 close()	284
15.30.3.2 empty()	284
15.30.3.3 pop() [1/2]	285
15.30.3.4 pop() [2/2]	285
15.30.3.5 push()	286
15.31 PartialSendError Class Reference	286
15.31.1 Detailed Description	288
15.31.2 Constructor & Destructor Documentation	288
15.31.2.1 PartialSendError()	288
15.31.3 Member Function Documentation	288
15.31.3.1 what()	288
15.32 QueuedPacket Class Reference	289
15.32.1 Detailed Description	290
15.32.2 Constructor & Destructor Documentation	290
15.32.2.1 QueuedPacket() [1/3]	290
15.32.2.2 QueuedPacket() [2/3]	290
15.32.2.3 QueuedPacket() [3/3]	291
15.32.3 Member Function Documentation	291
15.32.3.1 operator=() [1/2]	291
15.32.3.2 operator=() [2/2]	291
15.32.3.3 packet()	292
15.32.4 Friends And Related Function Documentation	292

15.32.4.1 operator"!=	292
15.32.4.2 operator<	293
15.32.4.3 operator<<	293
15.32.4.4 operator<=	294
15.32.4.5 operator==	295
15.32.4.6 operator>	295
15.32.4.7 operator>=	296
15.33RecursionData Class Reference	296
15.33.1 Detailed Description	297
15.33.2 Constructor & Destructor Documentation	297
15.33.2.1 RecursionData() [1/3]	298
15.33.2.2 RecursionData() [2/3]	298
15.33.2.3 RecursionData() [3/3]	298
15.33.3 Member Function Documentation	298
15.33.3.1 operator=() [1/2]	298
15.33.3.2 operator=() [2/2]	298
15.33.4 Friends And Related Function Documentation	299
15.33.4.1 RecursionGuard	299
15.34RecursionError Class Reference	299
15.34.1 Detailed Description	300
15.34.2 Constructor & Destructor Documentation	300
15.34.2.1 RecursionError()	300
15.34.3 Member Function Documentation	301
15.34.3.1 what()	301
15.35RecursionGuard Class Reference	301
15.35.1 Detailed Description	302
15.35.2 Constructor & Destructor Documentation	302
15.35.2.1 RecursionGuard()	302

15.35.2.2 ~RecursionGuard()	303
15.36 Reject Class Reference	304
15.36.1 Detailed Description	306
15.36.2 Constructor & Destructor Documentation	306
15.36.2.1 Reject()	306
15.36.3 Member Function Documentation	306
15.36.3.1 action()	306
15.36.3.2 clone()	307
15.36.3.3 operator"!="()	308
15.36.3.4 operator==()	308
15.36.3.5 print_()	309
15.36.4 Member Data Documentation	309
15.36.4.1 condition_	309
15.37 Rule Class Reference	310
15.37.1 Detailed Description	312
15.37.2 Constructor & Destructor Documentation	312
15.37.2.1 Rule()	312
15.37.2.2 ~Rule()	312
15.37.3 Member Function Documentation	312
15.37.3.1 action()	313
15.37.3.2 clone()	313
15.37.3.3 operator"!="()	314
15.37.3.4 operator==()	315
15.37.3.5 print_()	315
15.37.4 Friends And Related Function Documentation	315
15.37.4.1 operator<<	316
15.37.5 Member Data Documentation	316
15.37.5.1 condition_	316

15.38 semaphore Class Reference	317
15.38.1 Detailed Description	317
15.38.2 Constructor & Destructor Documentation	318
15.38.2.1 <code>semaphore()</code> [1/3]	318
15.38.2.2 <code>semaphore()</code> [2/3]	318
15.38.2.3 <code>semaphore()</code> [3/3]	318
15.38.3 Member Function Documentation	318
15.38.3.1 <code>notify()</code>	318
15.38.3.2 <code>operator=()</code> [1/2]	319
15.38.3.3 <code>operator=()</code> [2/2]	319
15.38.3.4 <code>wait()</code>	319
15.38.3.5 <code>wait_for()</code>	319
15.38.3.6 <code>wait_until()</code>	320
15.39 SerialInterface Class Reference	320
15.39.1 Detailed Description	322
15.39.2 Constructor & Destructor Documentation	323
15.39.2.1 <code>SerialInterface()</code>	323
15.39.2.2 <code>~SerialInterface()</code>	323
15.39.3 Member Function Documentation	323
15.39.3.1 <code>print_()</code>	323
15.39.3.2 <code>receive_packet()</code>	324
15.39.3.3 <code>send_packet()</code>	325
15.40 SerialPort Class Reference	325
15.40.1 Detailed Description	327
15.40.2 Member Enumeration Documentation	327
15.40.2.1 <code>Feature</code>	327
15.40.2.2 <code>Parity</code>	328
15.40.3 Constructor & Destructor Documentation	328

15.40.3.1 ~SerialPort()	328
15.40.4 Member Function Documentation	328
15.40.4.1 print_()	328
15.40.4.2 read() [1/2]	329
15.40.4.3 read() [2/2]	330
15.40.4.4 write() [1/2]	331
15.40.4.5 write() [2/2]	332
15.40.5 Friends And Related Function Documentation	332
15.40.5.1 operator<<	332
15.41 UDPInterface Class Reference	334
15.41.1 Detailed Description	335
15.41.2 Constructor & Destructor Documentation	336
15.41.2.1 UDPInterface()	336
15.41.2.2 ~UDPInterface()	336
15.41.3 Member Function Documentation	336
15.41.3.1 print_()	336
15.41.3.2 receive_packet()	337
15.41.3.3 send_packet()	338
15.42 UDPSocket Class Reference	339
15.42.1 Detailed Description	340
15.42.2 Constructor & Destructor Documentation	340
15.42.2.1 ~UDPSocket()	340
15.42.3 Member Function Documentation	340
15.42.3.1 print_()	340
15.42.3.2 receive() [1/2]	341
15.42.3.3 receive() [2/2]	342
15.42.3.4 send() [1/2]	343
15.42.3.5 send() [2/2]	344

15.42.4 Friends And Related Function Documentation	344
15.42.4.1 operator<<	344
15.43 UnixSerialPort Class Reference	345
15.43.1 Detailed Description	347
15.43.2 Member Enumeration Documentation	347
15.43.2.1 Feature	347
15.43.2.2 Parity	347
15.43.3 Constructor & Destructor Documentation	348
15.43.3.1 UnixSerialPort()	348
15.43.3.2 ~UnixSerialPort()	348
15.43.4 Member Function Documentation	349
15.43.4.1 print_()	349
15.43.4.2 read() [1/2]	349
15.43.4.3 read() [2/2]	351
15.43.4.4 write() [1/2]	352
15.43.4.5 write() [2/2]	352
15.44 UnixSyscalls Class Reference	353
15.44.1 Detailed Description	354
15.44.2 Constructor & Destructor Documentation	355
15.44.2.1 ~UnixSyscalls()	355
15.44.3 Member Function Documentation	355
15.44.3.1 bind()	355
15.44.3.2 close()	356
15.44.3.3 ioctl()	356
15.44.3.4 open()	357
15.44.3.5 poll()	357
15.44.3.6 read()	357
15.44.3.7 recvfrom()	358

15.44.3.8 sendto()	358
15.44.3.9 socket()	359
15.44.3.10 tcgetattr()	359
15.44.3.11 tcsetattr()	359
15.44.3.12 write()	360
15.45 UnixUDPSocket Class Reference	360
15.45.1 Detailed Description	362
15.45.2 Constructor & Destructor Documentation	362
15.45.2.1 UnixUDPSocket()	362
15.45.2.2 ~UnixUDPSocket()	362
15.45.3 Member Function Documentation	363
15.45.3.1 print_()	363
15.45.3.2 receive() [1/2]	363
15.45.3.3 receive() [2/2]	364
15.45.3.4 send() [1/2]	365
15.45.3.5 send() [2/2]	366
15.46 mavlink::v1_header Struct Reference	367
15.46.1 Detailed Description	368
15.46.2 Member Data Documentation	368
15.46.2.1 compid	368
15.46.2.2 len	368
15.46.2.3 magic	368
15.46.2.4 msgid	368
15.46.2.5 seq	369
15.46.2.6 sysid	369
15.47 mavlink::v2_header Struct Reference	369
15.47.1 Detailed Description	370
15.47.2 Member Data Documentation	370
15.47.2.1 compat_flags	370
15.47.2.2 compid	370
15.47.2.3 incompat_flags	371
15.47.2.4 len	371
15.47.2.5 magic	371
15.47.2.6 msgid	371
15.47.2.7 seq	371
15.47.2.8 sysid	371

16 File Documentation	373
16.1 Accept.cpp File Reference	373
16.2 Accept.cpp	373
16.3 Accept.hpp File Reference	375
16.4 Accept.hpp	376
16.5 Action.cpp File Reference	377
16.6 Action.cpp	378
16.7 Action.hpp File Reference	380
16.7.1 Function Documentation	381
16.7.1.1 operator"!="	381
16.7.1.2 operator<<()	382
16.7.1.3 operator==()	382
16.8 Action.hpp	382
16.9 AddressPool.hpp File Reference	383
16.10AddressPool.hpp	384
16.11ansi_codes.sh File Reference	386
16.12ansi_codes.sh	386
16.13App.cpp File Reference	387
16.14App.cpp	388
16.15App.hpp File Reference	389
16.16App.hpp	390
16.17Call.cpp File Reference	390
16.18Call.cpp	391
16.19Call.hpp File Reference	393
16.20Call.hpp	394
16.21Chain.cpp File Reference	395
16.22Chain.cpp	396
16.23Chain.hpp File Reference	399

16.23.1 Function Documentation	400
16.23.1.1 operator"!="	400
16.23.1.2 operator<<()	400
16.23.1.3 operator==()	400
16.24Chain.hpp	401
16.25common.hpp File Reference	402
16.25.1 Function Documentation	402
16.25.1.1 mock_shared()	402
16.25.1.2 mock_unique()	403
16.26common.hpp	404
16.27common_Packet.hpp File Reference	405
16.28common_Packet.hpp	406
16.29common_Rule.hpp File Reference	411
16.30common_Rule.hpp	411
16.31config_grammar.cpp File Reference	412
16.32config_grammar.cpp	413
16.33config_grammar.hpp File Reference	416
16.34config_grammar.hpp	417
16.35ConfigParser.cpp File Reference	422
16.36ConfigParser.cpp	423
16.37ConfigParser.hpp File Reference	429
16.37.1 Function Documentation	430
16.37.1.1 init_chains()	430
16.37.1.2 operator<<()	431
16.37.1.3 parse_action()	431
16.37.1.4 parse_chain()	431
16.37.1.5 parse_condition()	431
16.37.1.6 parse_filter()	431

16.37.1.7 parse_interfaces()	431
16.37.1.8 parse_serial()	432
16.37.1.9 parse_udp()	432
16.38 ConfigParser.hpp	432
16.39 configuration.md File Reference	433
16.40 configuration.md	433
16.41 Connection.cpp File Reference	439
16.41.1 Function Documentation	440
16.41.1.1 operator<<()	440
16.42 Connection.cpp	440
16.43 Connection.hpp File Reference	445
16.43.1 Function Documentation	446
16.43.1.1 operator<<()	446
16.44 Connection.hpp	446
16.45 ConnectionFactory.hpp File Reference	447
16.46 ConnectionFactory.hpp	448
16.47 ConnectionPool.cpp File Reference	450
16.48 ConnectionPool.cpp	450
16.49 ConnectionPool.hpp File Reference	451
16.50 ConnectionPool.hpp	452
16.51 CONTRIBUTING.md File Reference	453
16.52 CONTRIBUTING.md	453
16.53 DNSLookupError.cpp File Reference	454
16.54 DNSLookupError.hpp	455
16.55 DNSLookupError.hpp File Reference	456
16.56 DNSLookupError.hpp	456
16.57 Filesystem.cpp File Reference	457
16.58 Filesystem.hpp	457

16.59Filesystem.hpp File Reference	458
16.60Filesystem.hpp	459
16.61Filter.cpp File Reference	460
16.62Filter.cpp	460
16.63Filter.hpp File Reference	461
16.63.1 Function Documentation	462
16.63.1.1 operator"!="()	463
16.63.1.2 operator==(())	463
16.64Filter.hpp	463
16.65GoTo.cpp File Reference	464
16.66GoTo.cpp	464
16.67GoTo.hpp File Reference	467
16.68GoTo.hpp	468
16.69If.cpp File Reference	469
16.69.1 Function Documentation	469
16.69.1.1 operator<<()	469
16.70If.cpp	470
16.71If.hpp File Reference	473
16.71.1 Function Documentation	474
16.71.1.1 operator"!="()	474
16.71.1.2 operator<<()	475
16.71.1.3 operator==(())	475
16.72If.hpp	476
16.73Interface.cpp File Reference	477
16.74Interface.cpp	477
16.75Interface.hpp File Reference	478
16.75.1 Function Documentation	479
16.75.1.1 operator<<()	479

16.76Interface.hpp	479
16.77InterfaceThreader.cpp File Reference	480
16.78InterfaceThreader.cpp	480
16.79InterfaceThreader.hpp File Reference	482
16.80InterfaceThreader.hpp	483
16.81InvalidPacketIDError.cpp File Reference	484
16.82InvalidPacketIDError.cpp	485
16.83InvalidPacketIDError.hpp File Reference	486
16.84InvalidPacketIDError.hpp	486
16.85IPAddress.cpp File Reference	487
16.85.1 Function Documentation	487
16.85.1.1 operator<<()	488
16.86IPAddress.cpp	489
16.87IPAddress.hpp File Reference	494
16.87.1 Function Documentation	496
16.87.1.1 dnslookup()	496
16.87.1.2 operator"!="()	496
16.87.1.3 operator<()	496
16.87.1.4 operator<<()	496
16.87.1.5 operator<=()	497
16.87.1.6 operator==()	497
16.87.1.7 operator>()	498
16.87.1.8 operator>=()	498
16.88IPAddress.hpp	498
16.89LICENSE.md File Reference	499
16.90LICENSE.md	499
16.91Logger.cpp File Reference	503
16.92Logger.cpp	504

16.93Logger.hpp File Reference	505
16.94Logger.hpp	506
16.95logger.py File Reference	507
16.96logger.py	507
16.97macros.hpp File Reference	509
16.98macros.hpp	509
16.99mainpage.md File Reference	510
16.100mainpage.md	510
16.101make_targets.md File Reference	510
16.102make_targets.md	510
16.103MAVAddress.cpp File Reference	512
16.104MAVAddress.cpp	512
16.105MAVAddress.hpp File Reference	516
16.105.1.Function Documentation	517
16.105.1.1operator"!="()	517
16.105.1.2operator<()	517
16.105.1.3operator<<()	517
16.105.1.4operator<=()	517
16.105.1.5operator==()	518
16.105.1.6operator>()	518
16.105.1.7operator>=()	518
16.106MAVAddress.hpp	518
16.107mavlink.cpp File Reference	519
16.108mavlink.cpp	520
16.109mavlink.hpp File Reference	521
16.109.1.Macro Definition Documentation	522
16.109.1.1MAVLINK_USE_MESSAGE_INFO	522
16.110mavlink.hpp	523

16.11 MAVSubnet.cpp File Reference	524
16.11 MAVSubnet.cpp	524
16.11 MAVSubnet.hpp File Reference	529
16.113. Function Documentation	530
16.113.1.1operator"!="()	531
16.113.1.2operator<<()	531
16.113.1.3operator==()	531
16.11 MAVSubnet.hpp	531
16.11 navtables.cpp File Reference	532
16.115. Function Documentation	533
16.115.1.1main()	533
16.116navtables.cpp	534
16.11 Options.cpp File Reference	535
16.118Options.cpp	535
16.119Options.hpp File Reference	538
16.119. Function Documentation	539
16.119.1.1find_config()	539
16.120Options.hpp	540
16.12 Packet.cpp File Reference	540
16.122Packet.cpp	541
16.123Packet.hpp File Reference	543
16.123. Function Documentation	544
16.123.1.1operator"!="()	544
16.123.1.2operator<<()	544
16.123.1.3operator==()	544
16.124Packet.hpp	545
16.125packet_scripter.py File Reference	546
16.126packet_scripter.py	547

16.12 <code>P</code> acket <code>P</code> arser.cpp File Reference	553
16.12 <code>P</code> acket <code>P</code> arser.hpp	553
16.12 <code>P</code> acket <code>P</code> arser.hpp File Reference	556
16.13 <code>P</code> acket <code>P</code> arser.hpp	557
16.13 <code>P</code> acket <code>Q</code> ueue.cpp File Reference	558
16.13 <code>P</code> acket <code>Q</code> ueue.hpp	558
16.13 <code>P</code> acket <code>Q</code> ueue.hpp File Reference	561
16.13 <code>P</code> acket <code>Q</code> ueue.hpp	562
16.13 <code>P</code> acket <code>V</code> ersion1.cpp File Reference	563
16.13 <code>P</code> acket <code>V</code> ersion1.hpp	564
16.13 <code>P</code> acket <code>V</code> ersion1.hpp File Reference	567
16.13 <code>P</code> acket <code>V</code> ersion1.hpp	568
16.13 <code>P</code> acket <code>V</code> ersion2.cpp File Reference	569
16.14 <code>P</code> acket <code>V</code> ersion2.cpp	570
16.14 <code>P</code> acket <code>V</code> ersion2.hpp File Reference	574
16.14 <code>P</code> acket <code>V</code> ersion2.hpp	575
16.14 <code>P</code> arse <code>_</code> tree.hpp File Reference	576
16.14 <code>P</code> arse <code>_</code> tree.hpp	577
16.14 <code>P</code> artialSendError.cpp File Reference	582
16.14 <code>P</code> artialSendError.hpp	582
16.14 <code>P</code> artialSendError.hpp File Reference	583
16.14 <code>P</code> artialSendError.hpp	584
16.14 <code>Q</code> ueuedPacket.cpp File Reference	584
16.15 <code>Q</code> ueuedPacket.hpp	585
16.15 <code>Q</code> ueuedPacket.hpp File Reference	588
16.151. Function Documentation	589
16.151.1. <code>operator"!=()</code>	589
16.151.1.2. <code>operator<()</code>	589

16.151.1.3operator<<()	589
16.151.1.4operator<=()	589
16.151.1.5operator==()	589
16.151.1.6operator>()	590
16.151.1.7operator>=()	590
16.152QueuedPacket.hpp	590
16.153README.md File Reference	591
16.154README.md	591
16.155RecursionData.hpp File Reference	593
16.156RecursionData.hpp	594
16.157RecursionError.cpp File Reference	595
16.158RecursionError.cpp	595
16.159RecursionError.hpp File Reference	596
16.160RecursionError.hpp	596
16.161RecursionGuard.cpp File Reference	597
16.162RecursionGuard.cpp	598
16.163RecursionGuard.hpp File Reference	598
16.164RecursionGuard.hpp	599
16.165Reject.cpp File Reference	600
16.166Reject.cpp	601
16.167Reject.hpp File Reference	602
16.168Reject.hpp	603
16.169Rule.cpp File Reference	604
16.170Rule.cpp	604
16.171Rule.hpp File Reference	605
16.171.1Function Documentation	606
16.171.1.1operator<<()	606
16.172Rule.hpp	607

16.17 8 un_tests.sh File Reference	608
16.17 9 unit_tests/run_tests.sh	608
16.17 5 un_tests.sh File Reference	609
16.17 6 integration_tests/run_tests.sh	609
16.17 8 Semaphore.cpp File Reference	615
16.17 9 Semaphore.cpp	616
16.17 9 Semaphore.hpp File Reference	616
16.18 0 Semaphore.hpp	617
16.18 1 SerialInterface.cpp File Reference	619
16.18 2 SerialInterface.cpp	619
16.18 3 SerialInterface.hpp File Reference	621
16.18 4 SerialInterface.hpp	622
16.18 5 SerialPort.cpp File Reference	623
16.18 6 SerialPort.cpp	623
16.18 7 SerialPort.hpp File Reference	625
16.187. Function Documentation	626
16.187.1. operator<<()	626
16.18 8 SerialPort.hpp	627
16.18 9 test_Accept.cpp File Reference	628
16.189. Function Documentation	628
16.189.1.1TEST_CASE() [1/8]	629
16.189.1.2TEST_CASE() [2/8]	629
16.189.1.3TEST_CASE() [3/8]	629
16.189.1.4TEST_CASE() [4/8]	629
16.189.1.5TEST_CASE() [5/8]	630
16.189.1.6TEST_CASE() [6/8]	630
16.189.1.7TEST_CASE() [7/8]	630
16.189.1.8TEST_CASE() [8/8]	630

16.19 test _Accept.cpp	631
16.19 test _Action.cpp File Reference	634
16.191.1.Function Documentation	634
16.191.1.1TEST_CASE() [1/10]	635
16.191.1.2TEST_CASE() [2/10]	635
16.191.1.3TEST_CASE() [3/10]	635
16.191.1.4TEST_CASE() [4/10]	636
16.191.1.5TEST_CASE() [5/10]	636
16.191.1.6TEST_CASE() [6/10]	636
16.191.1.7TEST_CASE() [7/10]	637
16.191.1.8TEST_CASE() [8/10]	637
16.191.1.9TEST_CASE() [9/10]	638
16.191.1.10TEST_CASE() [10/10]	638
16.191.2.Variable Documentation	638
16.191.2.1action_a	638
16.191.2.2action_b	639
16.19 test _Action.cpp	639
16.19 test _AddressPool.cpp File Reference	641
16.193.1.Function Documentation	642
16.193.1.1TEST_CASE() [1/4]	642
16.193.1.2TEST_CASE() [2/4]	642
16.193.1.3TEST_CASE() [3/4]	642
16.193.1.4TEST_CASE() [4/4]	642
16.19 test _AddressPool.cpp	643
16.19 test _Call.cpp File Reference	647
16.195.1.Function Documentation	648
16.195.1.1call()	648
16.195.1.2TEST_CASE() [1/6]	648

16.195.1.3TEST_CASE() [2/6]	649
16.195.1.4TEST_CASE() [3/6]	649
16.195.1.5TEST_CASE() [4/6]	650
16.195.1.6TEST_CASE() [5/6]	650
16.195.1.7TEST_CASE() [6/6]	650
16.195.2Variable Documentation	650
16.195.2.1call	650
16.195.2.2ping	651
16.195.2.3rule	651
16.196test_Call.cpp	651
16.197test_Chain.cpp File Reference	655
16.197.1Function Documentation	656
16.197.1.1append() [1/2]	656
16.197.1.2append() [2/2]	656
16.197.1.3TEST_CASE() [1/9]	657
16.197.1.4TEST_CASE() [2/9]	657
16.197.1.5TEST_CASE() [3/9]	657
16.197.1.6TEST_CASE() [4/9]	658
16.197.1.7TEST_CASE() [5/9]	658
16.197.1.8TEST_CASE() [6/9]	658
16.197.1.9TEST_CASE() [7/9]	659
16.197.1.10TEST_CASE() [8/9]	659
16.197.1.11TEST_CASE() [9/9]	660
16.197.2Variable Documentation	660
16.197.2.1chain_a	660
16.197.2.2chain_a_compare	660
16.197.2.3chain_b	660
16.197.2.4chain_b_compare	660

16.19 test_Chain.cpp	661
16.19 test_config_grammar.cpp File Reference	664
16.199. Function Documentation	665
16.199.1.1 TEST_CASE() [1/28]	666
16.199.1.2 TEST_CASE() [2/28]	666
16.199.1.3 TEST_CASE() [3/28]	666
16.199.1.4 TEST_CASE() [4/28]	667
16.199.1.5 TEST_CASE() [5/28]	667
16.199.1.6 TEST_CASE() [6/28]	667
16.199.1.7 TEST_CASE() [7/28]	667
16.199.1.8 TEST_CASE() [8/28]	667
16.199.1.9 TEST_CASE() [9/28]	668
16.199.1.10 TEST_CASE() [10/28]	668
16.199.1.11 TEST_CASE() [11/28]	668
16.199.1.12 TEST_CASE() [12/28]	668
16.199.1.13 TEST_CASE() [13/28]	668
16.199.1.14 TEST_CASE() [14/28]	669
16.199.1.15 TEST_CASE() [15/28]	669
16.199.1.16 TEST_CASE() [16/28]	669
16.199.1.17 TEST_CASE() [17/28]	669
16.199.1.18 TEST_CASE() [18/28]	669
16.199.1.19 TEST_CASE() [19/28]	670
16.199.1.20 TEST_CASE() [20/28]	670
16.199.1.21 TEST_CASE() [21/28]	670
16.199.1.22 TEST_CASE() [22/28]	670
16.199.1.23 TEST_CASE() [23/28]	670
16.199.1.24 TEST_CASE() [24/28]	671
16.199.1.25 TEST_CASE() [25/28]	671

16.199.1.2TEST_CASE() [26/28]	671
16.199.1.2TEST_CASE() [27/28]	671
16.199.1.2TEST_CASE() [28/28]	671
16.200test_config_grammar.cpp	672
16.201test_ConfigParser.cpp File Reference	706
16.201.1Function Documentation	707
16.201.1.1TEST_CASE() [1/11]	707
16.201.1.2TEST_CASE() [2/11]	707
16.201.1.3TEST_CASE() [3/11]	708
16.201.1.4TEST_CASE() [4/11]	708
16.201.1.5TEST_CASE() [5/11]	708
16.201.1.6TEST_CASE() [6/11]	709
16.201.1.7TEST_CASE() [7/11]	709
16.201.1.8TEST_CASE() [8/11]	709
16.201.1.9TEST_CASE() [9/11]	710
16.201.1.10TEST_CASE() [10/11]	710
16.201.1.11TEST_CASE() [11/11]	710
16.202test_ConfigParser.cpp	710
16.203test_Connection.cpp File Reference	724
16.203.1Function Documentation	725
16.203.1.1TEST_CASE() [1/10]	725
16.203.1.2TEST_CASE() [2/10]	726
16.203.1.3TEST_CASE() [3/10]	726
16.203.1.4TEST_CASE() [4/10]	727
16.203.1.5TEST_CASE() [5/10]	728
16.203.1.6TEST_CASE() [6/10]	728
16.203.1.7TEST_CASE() [7/10]	729
16.203.1.8TEST_CASE() [8/10]	729

16.203.1.9TEST_CASE() [9/10]	730
16.203.1.10TEST_CASE() [10/10]	731
16.204test_Connection.cpp	731
16.205test_ConnectionFactory.cpp File Reference	747
16.205.1.Function Documentation	747
16.205.1.1TEST_CASE() [1/3]	748
16.205.1.2TEST_CASE() [2/3]	748
16.205.1.3TEST_CASE() [3/3]	749
16.206test_ConnectionFactory.cpp	749
16.207test_ConnectionPool.cpp File Reference	751
16.207.1.Function Documentation	752
16.207.1.1TEST_CASE() [1/5]	752
16.207.1.2TEST_CASE() [2/5]	752
16.207.1.3TEST_CASE() [3/5]	753
16.207.1.4TEST_CASE() [4/5]	753
16.207.1.5TEST_CASE() [5/5]	754
16.208test_ConnectionPool.cpp	754
16.209test_DNSLookupError.cpp File Reference	757
16.209.1.Function Documentation	757
16.209.1.1TEST_CASE() [1/2]	757
16.209.1.2TEST_CASE() [2/2]	758
16.210test_DNSLookupError.cpp	758
16.211test_Filesystem.cpp File Reference	759
16.211.1.Function Documentation	759
16.211.1.1TEST_CASE()	759
16.212test_Filesystem.cpp	760
16.213test_Filter.cpp File Reference	760
16.213.1.Function Documentation	761

16.213.1.1TEST_CASE() [1/6]	761
16.213.1.2TEST_CASE() [2/6]	761
16.213.1.3TEST_CASE() [3/6]	762
16.213.1.4TEST_CASE() [4/6]	762
16.213.1.5TEST_CASE() [5/6]	762
16.213.1.6TEST_CASE() [6/6]	762
16.213.2Variable Documentation	762
16.213.2.1filter_a	763
16.213.2.2filter_b	763
16.214test_Filter.cpp	763
16.215test_GoTo.cpp File Reference	765
16.215.1Function Documentation	766
16.215.1.1goto_()	766
16.215.1.2TEST_CASE() [1/6]	766
16.215.1.3TEST_CASE() [2/6]	767
16.215.1.4TEST_CASE() [3/6]	767
16.215.1.5TEST_CASE() [4/6]	768
16.215.1.6TEST_CASE() [5/6]	768
16.215.1.7TEST_CASE() [6/6]	768
16.215.2Variable Documentation	768
16.215.2.1goto_	768
16.215.2.2ping	769
16.215.2.3rule	769
16.216test_GoTo.cpp	769
16.217test_If.cpp File Reference	773
16.217.1Function Documentation	774
16.217.1.1TEST_CASE() [1/10]	774
16.217.1.2TEST_CASE() [2/10]	774

16.217.1.3TEST_CASE() [3/10]	775
16.217.1.4TEST_CASE() [4/10]	775
16.217.1.5TEST_CASE() [5/10]	775
16.217.1.6TEST_CASE() [6/10]	775
16.217.1.7TEST_CASE() [7/10]	776
16.217.1.8TEST_CASE() [8/10]	776
16.217.1.9TEST_CASE() [9/10]	776
16.217.1.10TEST_CASE() [10/10]	776
16.217.2Variable Documentation	776
16.217.2.1if_a	777
16.217.2.2f_b	777
16.218test_If.cpp	777
16.219test_Interface.cpp File Reference	782
16.219.Function Documentation	782
16.219.1.1TEST_CASE() [1/4]	782
16.219.1.2TEST_CASE() [2/4]	783
16.219.1.3TEST_CASE() [3/4]	783
16.219.1.4TEST_CASE() [4/4]	784
16.220test_Interface.cpp	784
16.221test_InterfaceThreader.cpp File Reference	786
16.221.Function Documentation	787
16.221.1.1TEST_CASE() [1/2]	787
16.221.1.2TEST_CASE() [2/2]	787
16.222test_InterfaceThreader.cpp	787
16.223test_IPAddress.cpp File Reference	789
16.223.Function Documentation	790
16.223.1.1TEST_CASE() [1/10]	790
16.223.1.2TEST_CASE() [2/10]	791

16.223.1.3TEST_CASE() [3/10]	791
16.223.1.4TEST_CASE() [4/10]	791
16.223.1.5TEST_CASE() [5/10]	791
16.223.1.6TEST_CASE() [6/10]	792
16.223.1.7TEST_CASE() [7/10]	792
16.223.1.8TEST_CASE() [8/10]	792
16.223.1.9TEST_CASE() [9/10]	792
16.223.1.10TEST_CASE() [10/10]	793
16.223.2Variable Documentation	793
16.223.2.1address_a	793
16.223.2.2address_b	793
16.224test_IPAddress.cpp	793
16.225test_LOGGER.cpp File Reference	798
16.225Function Documentation	799
16.225.1.1TEST_CASE() [1/3]	799
16.225.1.2TEST_CASE() [2/3]	799
16.225.1.3TEST_CASE() [3/3]	799
16.226test_LOGGER.cpp	800
16.227test_MAVAddress.cpp File Reference	801
16.227Function Documentation	802
16.227.1.1TEST_CASE() [1/9]	802
16.227.1.2TEST_CASE() [2/9]	802
16.227.1.3TEST_CASE() [3/9]	802
16.227.1.4TEST_CASE() [4/9]	803
16.227.1.5TEST_CASE() [5/9]	803
16.227.1.6TEST_CASE() [6/9]	803
16.227.1.7TEST_CASE() [7/9]	803
16.227.1.8TEST_CASE() [8/9]	804

16.227.1.9TEST_CASE() [9/9]	804
16.227.2Variable Documentation	804
16.227.2.1address_a	804
16.227.2.2address_b	804
16.228test_MAVAddress.cpp	805
16.229test_mavlink.cpp File Reference	808
16.229.Function Documentation	809
16.229.1.1TEST_CASE()	809
16.230test_mavlink.cpp	809
16.231test_MAVSubnet.cpp File Reference	810
16.231.Function Documentation	811
16.231.1.1TEST_CASE() [1/9]	811
16.231.1.2TEST_CASE() [2/9]	812
16.231.1.3TEST_CASE() [3/9]	812
16.231.1.4TEST_CASE() [4/9]	812
16.231.1.5TEST_CASE() [5/9]	812
16.231.1.6TEST_CASE() [6/9]	813
16.231.1.7TEST_CASE() [7/9]	813
16.231.1.8TEST_CASE() [8/9]	813
16.231.1.9TEST_CASE() [9/9]	813
16.231.2Variable Documentation	813
16.231.2.1subnet_a	814
16.231.2.2subnet_b	814
16.232test_MAVSubnet.cpp	814
16.233test_Options.cpp File Reference	819
16.233.Function Documentation	820
16.233.1.1TEST_CASE() [1/7]	820
16.233.1.2TEST_CASE() [2/7]	820

16.233.1.3TEST_CASE() [3/7]	820
16.233.1.4TEST_CASE() [4/7]	820
16.233.1.5TEST_CASE() [5/7]	821
16.233.1.6TEST_CASE() [6/7]	821
16.233.1.7TEST_CASE() [7/7]	821
16.234test_Options.cpp	821
16.235test_Packet.cpp File Reference	827
16.235.1Function Documentation	828
16.235.1.1packet_b()	828
16.235.1.2TEST_CASE() [1/13]	828
16.235.1.3TEST_CASE() [2/13]	829
16.235.1.4TEST_CASE() [3/13]	829
16.235.1.5TEST_CASE() [4/13]	829
16.235.1.6TEST_CASE() [5/13]	829
16.235.1.7TEST_CASE() [6/13]	830
16.235.1.8TEST_CASE() [7/13]	830
16.235.1.9TEST_CASE() [8/13]	830
16.235.1.10TEST_CASE() [9/13]	830
16.235.1.11TEST_CASE() [10/13]	830
16.235.1.12TEST_CASE() [11/13]	831
16.235.1.13TEST_CASE() [12/13]	831
16.235.1.14TEST_CASE() [13/13]	831
16.235.2Variable Documentation	831
16.235.2.1packet_a	831
16.236test_Packet.cpp	832
16.237test_PacketParser.cpp File Reference	834
16.237.1Function Documentation	835
16.237.1.1TEST_CASE() [1/4]	835

16.237.1.2TEST_CASE() [2/4]	835
16.237.1.3TEST_CASE() [3/4]	836
16.237.1.4TEST_CASE() [4/4]	836
16.238test_PacketParser.cpp	836
16.239test_PacketQueue.cpp File Reference	839
16.239.1.Function Documentation	840
16.239.1.1TEST_CASE() [1/7]	840
16.239.1.2TEST_CASE() [2/7]	840
16.239.1.3TEST_CASE() [3/7]	841
16.239.1.4TEST_CASE() [4/7]	841
16.239.1.5TEST_CASE() [5/7]	841
16.239.1.6TEST_CASE() [6/7]	842
16.239.1.7TEST_CASE() [7/7]	842
16.240test_PacketQueue.cpp	842
16.241test_PacketVersion1.cpp File Reference	846
16.241.1.Function Documentation	847
16.241.1.1TEST_CASE() [1/16]	847
16.241.1.2TEST_CASE() [2/16]	848
16.241.1.3TEST_CASE() [3/16]	848
16.241.1.4TEST_CASE() [4/16]	849
16.241.1.5TEST_CASE() [5/16]	849
16.241.1.6TEST_CASE() [6/16]	849
16.241.1.7TEST_CASE() [7/16]	850
16.241.1.8TEST_CASE() [8/16]	850
16.241.1.9TEST_CASE() [9/16]	850
16.241.1.10TEST_CASE() [10/16]	851
16.241.1.11TEST_CASE() [11/16]	851
16.241.1.12TEST_CASE() [12/16]	852

16.241.1.1 TEST_CASE() [13/16]	852
16.241.1.1 TEST_CASE() [14/16]	853
16.241.1.1 TEST_CASE() [15/16]	853
16.241.1.1 TEST_CASE() [16/16]	854
16.241.2 Variable Documentation	854
16.241.2.1 packet_a	854
16.241.2.2 packet_b	854
16.242 test_PacketVersion1.cpp	855
16.243 test_PacketVersion2.cpp File Reference	861
16.243.1 Function Documentation	862
16.243.1.1 TEST_CASE() [1/17]	862
16.243.1.2 TEST_CASE() [2/17]	862
16.243.1.3 TEST_CASE() [3/17]	863
16.243.1.4 TEST_CASE() [4/17]	863
16.243.1.5 TEST_CASE() [5/17]	864
16.243.1.6 TEST_CASE() [6/17]	864
16.243.1.7 TEST_CASE() [7/17]	865
16.243.1.8 TEST_CASE() [8/17]	865
16.243.1.9 TEST_CASE() [9/17]	865
16.243.1.10 TEST_CASE() [10/17]	865
16.243.1.11 TEST_CASE() [11/17]	866
16.243.1.12 TEST_CASE() [12/17]	866
16.243.1.13 TEST_CASE() [13/17]	867
16.243.1.14 TEST_CASE() [14/17]	867
16.243.1.15 TEST_CASE() [15/17]	868
16.243.1.16 TEST_CASE() [16/17]	868
16.243.1.17 TEST_CASE() [17/17]	869
16.243.2 Variable Documentation	869

16.243.2.1packet_a	869
16.243.2.2packet_b	869
16.244test_PacketVersion2.cpp	870
16.245test_PartialSendError.cpp File Reference	879
16.245.1.Function Documentation	880
16.245.1.1TEST_CASE() [1/2]	880
16.245.1.2TEST_CASE() [2/2]	880
16.246test_PartialSendError.cpp	881
16.247test_QueuedPacket.cpp File Reference	882
16.247.1.Function Documentation	882
16.247.1.1TEST_CASE() [1/8]	882
16.247.1.2TEST_CASE() [2/8]	883
16.247.1.3TEST_CASE() [3/8]	883
16.247.1.4TEST_CASE() [4/8]	883
16.247.1.5TEST_CASE() [5/8]	883
16.247.1.6TEST_CASE() [6/8]	883
16.247.1.7TEST_CASE() [7/8]	884
16.247.1.8TEST_CASE() [8/8]	884
16.248test_QueuedPacket.cpp	884
16.249test_RecursionError.cpp File Reference	887
16.249.1.Function Documentation	888
16.249.1.1TEST_CASE() [1/2]	888
16.249.1.2TEST_CASE() [2/2]	888
16.250test_RecursionError.cpp	888
16.251test_RecursionGuard.cpp File Reference	889
16.251.1.Function Documentation	890
16.251.1.1TEST_CASE() [1/2]	890
16.251.1.2TEST_CASE() [2/2]	890

16.25 test _RecursionGuard.cpp	890
16.25 test _Reject.cpp File Reference	891
16.253.1.Function Documentation	892
16.253.1.1reject()	892
16.253.1.2TEST_CASE() [1/4]	892
16.253.1.3TEST_CASE() [2/4]	892
16.253.1.4TEST_CASE() [3/4]	893
16.253.1.5TEST_CASE() [4/4]	893
16.253.2.Variable Documentation	893
16.253.2.1reject	893
16.253.2.2rule	894
16.25 test _Reject.cpp	894
16.25 test _Rule.cpp File Reference	896
16.255.1.Function Documentation	896
16.255.1.1TEST_CASE() [1/5]	896
16.255.1.2TEST_CASE() [2/5]	897
16.255.1.3TEST_CASE() [3/5]	897
16.255.1.4TEST_CASE() [4/5]	897
16.255.1.5TEST_CASE() [5/5]	897
16.25 test _Rule.cpp	898
16.25 test _Rule_comparison.cpp File Reference	900
16.257.1.Function Documentation	900
16.257.1.1TEST_CASE()	900
16.25 test _Rule_comparison.cpp	901
16.25 test _Semaphore.cpp File Reference	902
16.259.1.Function Documentation	902
16.259.1.1TEST_CASE() [1/4]	903
16.259.1.2TEST_CASE() [2/4]	903

16.259.1.3TEST_CASE() [3/4]	903
16.259.1.4TEST_CASE() [4/4]	903
16.260test_semaphore.cpp	904
16.261test_SerialInterface.cpp File Reference	907
16.261.Function Documentation	908
16.261.1.1TEST_CASE() [1/4]	908
16.261.1.2TEST_CASE() [2/4]	908
16.261.1.3TEST_CASE() [3/4]	909
16.261.1.4TEST_CASE() [4/4]	910
16.262test_SerialInterface.cpp	910
16.263test_SerialPort.cpp File Reference	915
16.263.Function Documentation	916
16.263.1.1TEST_CASE() [1/5]	916
16.263.1.2TEST_CASE() [2/5]	917
16.263.1.3TEST_CASE() [3/5]	917
16.263.1.4TEST_CASE() [4/5]	918
16.263.1.5TEST_CASE() [5/5]	918
16.264test_SerialPort.cpp	918
16.265test_UDPInterface.cpp File Reference	920
16.265.Function Documentation	921
16.265.1.1TEST_CASE() [1/7]	921
16.265.1.2TEST_CASE() [2/7]	922
16.265.1.3TEST_CASE() [3/7]	923
16.265.1.4TEST_CASE() [4/7]	923
16.265.1.5TEST_CASE() [5/7]	924
16.265.1.6TEST_CASE() [6/7]	924
16.265.1.7TEST_CASE() [7/7]	925
16.266test_UDPInterface.cpp	926

16.26 Test_UDPSSocket.cpp File Reference	937
16.267. Function Documentation	937
16.267.1. TEST_CASE() [1/5]	937
16.267.1.2. TEST_CASE() [2/5]	938
16.267.1.3. TEST_CASE() [3/5]	938
16.267.1.4. TEST_CASE() [4/5]	939
16.267.1.5. TEST_CASE() [5/5]	939
16.268 Test_UDPSSocket.cpp	940
16.269 Test_UinxSerialPort.cpp File Reference	942
16.269. Function Documentation	942
16.269.1. TEST_CASE() [1/5]	943
16.269.1.2. TEST_CASE() [2/5]	943
16.269.1.3. TEST_CASE() [3/5]	943
16.269.1.4. TEST_CASE() [4/5]	944
16.269.1.5. TEST_CASE() [5/5]	944
16.270 Test_UinxSerialPort.cpp	944
16.271 Test_UinxUDPSocket.cpp File Reference	955
16.271. Function Documentation	956
16.271.1. TEST_CASE() [1/4]	956
16.271.1.2. TEST_CASE() [2/4]	956
16.271.1.3. TEST_CASE() [3/4]	956
16.271.1.4. TEST_CASE() [4/4]	957
16.272 Test_UinxUDPSocket.cpp	957
16.273 Test_utility.cpp File Reference	966
16.273. Function Documentation	967
16.273.1. TEST_CASE() [1/6]	967
16.273.1.2. TEST_CASE() [2/6]	967
16.273.1.3. TEST_CASE() [3/6]	968

16.273.1.4TEST_CASE() [4/6]	968
16.273.1.5TEST_CASE() [5/6]	968
16.273.1.6TEST_CASE() [6/6]	968
16.274test_utility.cpp	969
16.275UDPIInterface.cpp File Reference	972
16.276UDPIInterface.cpp	972
16.277UDPIInterface.hpp File Reference	974
16.278UDPIInterface.hpp	975
16.279UDPSocket.cpp File Reference	976
16.280UDPSocket.cpp	977
16.281UDPSocket.hpp File Reference	979
16.281.1Function Documentation	980
16.281.1.1operator<<()	980
16.282UDPSocket.hpp	980
16.283nit_tests.cpp File Reference	981
16.283.1Macro Definition Documentation	982
16.283.1.1CATCH_CONFIG_MAIN	982
16.284nit_tests.cpp	982
16.285UnixSerialPort.cpp File Reference	982
16.286UnixSerialPort.cpp	983
16.287UnixSerialPort.hpp File Reference	987
16.288UnixSerialPort.hpp	988
16.289UnixSyscalls.cpp File Reference	989
16.290UnixSyscalls.cpp	990
16.291UnixSyscalls.hpp File Reference	992
16.292UnixSyscalls.hpp	993
16.293UnixUDPSocket.cpp File Reference	994
16.294UnixUDPSocket.cpp	995
16.295UnixUDPSocket.hpp File Reference	998
16.296UnixUDPSocket.hpp	999
16.297user_manual.md File Reference	1000
16.298user_manual.md	1000
16.299utility.cpp File Reference	1003
16.300utility.cpp	1004
16.301utility.hpp File Reference	1004
16.302utility.hpp	1005
Index	1009

Chapter 1

mavtables

- [Build Status](#)
- [Coverage Status](#)

Introduction

A MAVLink router and firewall. It can connect 2 or more MAVLink endpoints such as autopilots, ground control software, loggers, image capture systems, etc over serial and UDP. MAVLink packets will be routed to specific components when they have a destination address. Any packet, targeted or broadcasted, can be filtered based on source system/component, destination system/component and message type. The filter can also apply a priority to packets allowing more important packets to take priority over lower priority packets when an endpoint is choked.

- [User Manual](#)
- [Configuration](#)
- [README](#)
- [LICENSE](#)
- [CONTRIBUTING](#)
- [Makefile Targets](#)

Links

- [HTML Documentation](#)
- [PDF Documentation](#)
- [Download](#)
- [GitHub](#)
- [Theory of MAVLink Routing](#)

Install

In order to compile you will need the following packages:

- GCC 7+ or Clang 5+ (needed for C++17 support)
- `CMake v3.3+`
- `Boost v1.54+`

mavtables can be easily installed using the standard procedure of

```
$ make  
# make install
```

The installation prefix is `/usr/local` by default but can be changed with

```
$ make  
# make PREFIX=/desired/install/path install
```

See the [README](#) for further documentation.

Chapter 2

Configuration

- `MAVLink Addressing`
 - `MAVLink Subnets`
- `Basic Grammar`
 - `Identifiers`
 - `Statements`
 - `Blocks`
 - `Comments`
- `Global Settings`
- `udp block`
 - `port statement`
 - `address statement`
 - `max_bitrate statement`
- `serial block`
 - `device statement`
 - `baudrate statement`
 - `flow_control statement`
 - `preload statement`
- `chain block`
 - `Rules`
 - `Action`
 - `Priority`
 - `Condition`
 - * `<packet type>`
 - * `<source address>`
 - * `<dest address>`

mavtables' configuration files are used to define default actions, interfaces, and filter chains. A full example configuration file can be found at `../examples/mavtables.conf`.

MAVLink Addressing

A MAVLink address consists of 2 octets. The first being the system ID and the second the component ID. Therefore, a MAVLink address looks like half of an IPv4 address.

```
<system ID>.<component ID>
```

MAVLink Subnets

MAVLink subnets are similar to IP subnets and thus it is recommended that the reader understand IP subnets before continuing.

There are four representations of MAVLink subnets:

1. "\<System ID>.<Component ID>:\<System ID mask>.\<Component ID mask>"
2. "\<System ID>.<Component ID>\&<bits>"
3. "\<System ID>.<Component ID>\|\<bits>"
4. "\<System ID>.<Component ID>"

The first form is self explanatory, but the 2nd and 3rd are not as simple. In the 2nd case the number of bits (0 - 16) is the number of bits from the left that must match for an address to be in the subnet. The 3rd form is like the 2nd, but does not require the system ID (first octet) to match and starts with the number of bits of the component ID (0 - 8) that must match from the left for an address to be in the subnet. The last form is shorthand for "<System ID>.<Component ID>/16", that is an exact match.

Below is a table relating the slash postfix to the subnet mask in <System mask>.<Component mask> notation.

Mask with /	Mask with \	Postfix (#bits)
255.255	out of range	16
255.254	out of range	15
255.252	out of range	14
255.248	out of range	13
255.240	out of range	12
255.224	out of range	11
255.192	out of range	10
255.128	out of range	9
255.0	0.255	8
254.0	0.254	7
252.0	0.252	6
248.0	0.248	5
240.0	0.240	4
224.0	0.224	3
192.0	0.192	2
128.0	0.128	1
0.0	0	0

Basic Grammar

There are four major types of grammar elements in configuration files. These are identifiers, statements, blocks, and comments. Configuration files have a similar syntax to the C programming language and like C they are not whitespace sensitive.

Identifiers

An identifier is a name containing any combination of letters (upper and lower), digits, and the underscores ('_'). However, it must not start with a number. This is the same definition as identifiers in the C programming language.

Statements

A statement is defined as an identifier, followed by a value, and then the *end of statement* character ';'. Therefore, it takes the form of

```
<identifier> <value>;
```

An example would be:

```
baodrate 57600;
```

where 'baodrate' is the identifier, '57600' is the value (in this case a number) and it ends with ; .

Blocks

A block is a sectioning construct that can contain one or more statements, or in the special case of filter blocks, a rule. Blocks consist of a type and optionally an identifier followed by '{', one or more statements (or rules) and finally ending in '}'. The format is:

```
<type> <optional identifier> {  
    <statement 1>;  
    <statement 2>;  
    <statement ...>;  
    <statement n>;  
}
```

An example would be:

```
udp {  
    port 14555;  
    address 127.0.0.1;  
}
```

Comments

The comment character is '#'. Everything after this character to the end of the line is considered a comment and will be ignored. An example of using comments is:

```
udp {  # UDP Interface
    port 14555;      # port number, the default is 14500
    address 127.0.0.1; # listen on localhost only, the default is any address
}
```

Global Settings

The following sections document the components of a configuration file.

default_action statement

Set the default action to take when the filter does not determine what to do with a packet. The options are:

- accept - to accept packets by default
- reject - to reject packets by default

To accept packets by default use:

```
default_action accept;
```

To reject packets by default use:

```
default_action reject;
```

udp block

The `udp` block defines a UDP interface to listen for connections on. An example is:

```
udp {
    port 14555;
    address 127.0.0.1;
    max_bitrate 8388608;
}
```

There is no limit to the number of UDP interfaces that can be defined.

port statement

A statement that sets the port number to listen on. The format is:

```
port <port number>;
```

An example is:

```
port 14555;
```

This is the only required statement in a `udp` block.

address statement (optional)

A statement that sets the IP address to listen on. The format is:

```
address <IP v4 address>;
```

An example is:

```
address 127.0.0.1;
```

which restricts mavtables to only listening for connections from `localhost`.

If not provided the default is to listen on every address and therefore to accept connections from remote systems (subject to the firewall's running on the host operating system).

max_bitrate statement (optional)

A statement that sets the bitrate limit (in bits per second) when transmitting packets over UDP. The format is:

```
max_bitrate <maximum bitrate>;
```

An example is:

```
max_bitrate 8388608; # 8 Mbps
```

If not provided there will not be any limit on the rate of packet transmission over UDP.

If downstream components tend to lose packets due to the operating system's UDP buffers filling up, consider slowing down mavtables by providing a bitrate limit. This feature was added because mavtables is much faster than components written in interpreted languages such as Python and thus must be limited so it can store packets in its own buffers to avoid overflowing the operating system buffers.

serial block

The `serial` block defines a serial port interface to listen for connections on. An example is:

```
serial {  
    device /dev/ttyUSB0;  
    baudrate 115200;  
    flow_control yes;  
    preload 1.1;  
}
```

There is no limit to the number of serial port interfaces that can be defined.

device statement

A statement that sets the serial device to listen on. The format is:

```
device <device string>;
```

where the `<device string>` is the path to the serial device.

An example is:

```
device /dev/ttyUSB0;
```

This is the only required statement in a `serial` block.

baudrate statement (optional)

A statement that sets the bitrate (in bits per second) of the serial port. The format is:

```
baudrate <bitrate>;
```

An example is:

```
baudrate 115200;
```

which sets the baudrate to 112.5 kbps.

If not provided the default is to use a baudrate of 9600 (9.4 kbps)

flow_control statement (optional)

A statement that enables/disables hardware flow control on the serial port. The format is:

```
flow_control <yes/no>;
```

To turn on hardware flow control:

```
flow_control yes;
```

To turn off hardware flow control:

```
flow_control no;
```

If not provided the default is to disable hardware flow control.

preload statement (optional)

A statement that can preload a MAVLink address on the serial port. Typically mavtables learns about a component when that component sends a packet to the router. In some cases a component may not send a packet until it receives one, in this case the component's address can be preloaded.

One particular case is when two mavtables are in use. Neither instance knows about the other because mavtables will not send any packets over a serial port if nothing has been sent to it over said serial port. By preloading addresses on both instances for the serial port connection they share, the packets from other components will be sent to the other mavtables instance and thus the instances can discover each other's components.

The format is:

```
preload <MAVLink address>;
```

An example is:

```
preload 1.1;
```

This indicates that system 1 and component 1 can be reached on this serial port, even before that component sends a packet to the router.

Any number of `preload` statements are allowed. Every address listed will be added to the serial port.

chain block

A **chain block** defines a filter chain consisting of one or more rules. [Filter](#) chains are used to define firewall rules and collect them into groups. The format is:

```
chain <chain name> {
    <rule 1>;
    <rule 2>;
    <rule ...>;
    <rule n>;
}
```

An example is:

```
chain default
{
    reject if ENCAPSULATED_DATA;
    accept;
}
```

which rejects the ENCAPSULATED_DATA packet, but accepts all others.

The 'default' chain is a reserved name and it must exist. The 'default' chain is the one used to filter packets before re-transmitting them. All other chains are called from the default chain or another chain via the 'call' or 'goto' rules.

There is no limit to the number of filter chains, or to the number of rules in a filter chain.

Rules

A rule has the form

```
<action> <optional priority> <optional condition>;
```

an example is

```
accept with priority 3 if ENCAPSULATED_DATA from 192.168 to 172.0/8;
```

When evaluating the filter, the first rule a packet matches will decide the fate of the packet. [If](#) a condition is not provided the rule will match all packets, otherwise the condition decides whether the rule matches or not.

Action

There are 4 types of actions a rule can have.

- accept - accept the packet
- reject - reject the packet
- call <chain name> - delegate accept/reject decision to another chain, <chain name>. Returns to parent chain if no match in the target chain.
- goto <chain name> - delegate accept/reject decision to another chain, <chain name>. [If](#) the target chain never matches the packet then the default action, set in default_action is used.

Priority

The priority form is:

```
with priority <priority level>
```

where `<priority level>` is a positive or negative integer. An example is:

```
with priority -4
```

A greater integer indicates a higher priority. Packets have a priority of 0 by default. If communications slow down and mavtables starts to build a queue of packets to send it will send out the higher priority packets first.

Condition

A condition determines whether or not a packet and the address it is being sent to matches a rule. Conditions have the form:

```
if <packet type> from <source address> to <dest address>;
```

where `<packet type>`, `from <source address>`, and `to <dest address>` are all optional, but one is required, otherwise the rule should not have a condition at all.

`<packet type>`

The name of the packet type to match (upper case). If left out all packet types will match.

`<source address>`

A MAVLink subnet to test for containment of the source MAVLink address of the packet.

`<dest address>`

A MAVLink subnet to test for containment of the destination MAVLink address of the packet. The destination address is not regarded as the destination contained in the packet (as that may not exist) but is the MAVLink address of where mavtables is attempting to send the packet.

Chapter 3

Makefile Targets

While mavtables uses [CMake](#) for building a simple Makefile (which calls `cmake`) is included for convenience. The available targets are listed below.

release (default)

Build `mavtables` using the *Release* option which generates an optimized (for speed) executable without debug symbols. `mavtables` is placed at `build/mavtables`.

Unit tests are built as well under `build/unit_tests` and are built with optimizations.

debug

Build `mavtables` and unit tests using the *Debug* option which will enable debug flags and turn off optimizations. The executables are located at `build/mavtables` and `build/unit_tests`.

test

Build and run all tests. Runs both `test/unit_tests/run_tests.sh` and `test/integration_tests/run_tests.sh`. The latter requires Python with the packages located in `test/integration_tests/requirements.txt`.

unit_tests

Build and run unit tests only.

integration_tests

Build and run integration tests only. Requires Python with the packages located in `test/integration_tests/requirements.txt`.

coverage

Build and run all tests and compute test coverage. This option requires `lcov` and `gcovr` to be installed and will only work when the compiler is `g++`.

The coverage per file will be printed to the terminal and a detailed, line by line, report generated with `lcov` at `build/lcov/html/selected_targets/index.html`.

linecheck

Prints all lines exceeding 80 characters.

style

Fix the style of C++ source code and header files. The original files are backed up with a `.orig` extension. [Artistic Style](#) is required to use this target. This will also call the linecheck target.

html

Generate html documentation with [Doxygen](#) and place it at `build/doc/html/index.html`.

doc

Generate html and pdf documentation with [Doxygen](#) and place it at `build/doc/html/index.html` and `build/doc/html/mavtables.pdf` respectively.

gh-pages

Generate html and pdf documentation and publish to <https://shamuproject.github.io/mavtables>.

clean

Clean up the project directory. This does not remove git submodules.

remove-subs

Remove all git submodules. They will be re-downloaded if needed.

Chapter 4

MAVTables User Manual

- [Project](#)
- [Installing](#)
 - [Requirements](#)
 - [CMake Installation](#)
 - [Manual Installation](#)
- [Usage](#)
 - [Running](#)
 - [Log Level](#)
 - * [-loglevel 0](#)
 - * [-loglevel 1](#)
 - * [-loglevel 2](#)
 - * [-loglevel 3](#)
 - [Abstract Syntax Tree](#)
- [Configuration](#)

Installing

Requirements

In order to compile you will need the following packages:

- GCC 7+ or Clang 5+ (needed for C++17 support)
- [CMake v3.3+](#)
- [Boost v1.54+](#)

on either

- Linux
- Mac OS X
- BSD

mavtables is tested on both Linux and Mac OS X and should work on any unix compatible system.

CMake Installation

To install via CMake simple run:

```
$ make  
$ make install
```

The installation prefix is `/usr/local` by default but can be changed with

```
$ make PREFIX=/desired/install/path  
# make DESTDIR=/desired/install/path install
```

`PREFIX` changes where mavtables expects to be installed while `DESTDIR` changes where it will actually be installed.

To change the MAVLink dialect use the `DIALECT` environment variable. For example, to use the `common` dialect:

```
$ make DIALECT=common  
# make install
```

To change the MAVLink implementation completely use the `MDIR` environment variable to set the path to the MAVLink library instead of downloading the upstream implementation. For example, if a custom implementation is at `/tmp/mavlink/v2` then:

```
$ make MDIR=/tmp/mavlink/v2  
# make install
```

Both the `MDIR` and `DIALECT` variables can be used together.

Manual Installation

To manually install mavtables first make a release build:

```
$ make
```

Remembering to use the `PREFIX` option if the final installation destination is not `/usr/local`.

Copy binary, configuration, and systemd unit files to their proper locations

```
# cp build/mavtables /usr/local/bin/  
# cp examples/mavtables.conf /usr/local/etc/  
# cp build/mavtables.service /usr/local/lib/systemd/system/
```

Usage

Running

To run mavtables and begin routing packets

```
$ mavtables
```

This will use the first configuration file it finds in the configuration file priority order:

1. The target of the MAVTABLES_CONFIG_PATH environment variable.
2. .mavtablesrc in the current directory.
3. .mavtablesrc at \$HOME/.mavtablesrc.
4. The main configuration file at /etc/mavtables.conf.

To specify the configuration file use

```
$ mavtables --config path/to/config/file
```

If the configuration file contains an error, mavtables will exit immediately and print an error message to stderr.

The installer will install a system wide configuration file at PREFIX/etc/mavtables.conf and a systemd unit file at PREFIX/lib/systemd/system/mavtables.service. Therefore, on Linux (with systemd) mavtables can be run as a daemon with

```
# systemctl start mavtables
```

or enabled for startup on each boot with

```
# systemctl enable mavtables
```

systemd will keep mavtables running (even if it crashes) and will use the PREFIX/etc/mavtables.conf configuration file. It will also use loglevel 1, discussed in the [Log Level](#) section below.

Log Level

The --loglevel flag can be used to set the logging level. This is the verbosity to print to stdout when mavtables is running. Each loglevel, 0 through 3, is documented below.

-loglevel 0

Do not log anything to stdcout.

-loglevel 1

Log each new component. Therefore, every time a new system/component address is connected to the router it will be printed to stdcout. An example log output is:

```
new component 127.1 on 127.0.0.1
new component 192.168 on 127.0.0.1
new component 172.128 on ./ttyS0
new component 10.10 on 127.0.0.1
```

-loglevel 2

In addition to everything in --loglevel 1 this will log received packets to stdcout. An example log output is:

```
new component 127.1 on 127.0.0.1
received HEARTBEAT (#0) from 127.1 (v2.0) source 127.0.0.1
new component 192.168 on 127.0.0.1
received HEARTBEAT (#0) from 192.168 (v2.0) source 127.0.0.1
new component 172.128 on ./ttyS0
received HEARTBEAT (#0) from 172.128 (v2.0) source ./ttyS0
new component 10.10 on 127.0.0.1
received HEARTBEAT (#0) from 10.10 (v2.0) source 127.0.0.1
received POWER_STATUS (#125) from 10.10 (v2.0) source 127.0.0.1
received SYS_STATUS (#1) from 10.10 (v2.0) source 127.0.0.1
received GPS_RAW_INT (#24) from 10.10 (v2.0) source 127.0.0.1
received LOCAL_POSITION_NED_COV (#64) from 10.10 (v2.0) source 127.0.0.1
received MISSION_REQUEST_PARTIAL_LIST (#37) from 10.10 to 172.168 (v2.0) source 127.0.0.1
received LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET (#89) from 10.10 (v2.0) source 127.0.0.1
received MANUAL_CONTROL (#69) from 10.10 (v2.0) source 127.0.0.1
received GPS_INPUT (#232) from 10.10 (v2.0) source 127.0.0.1
received VFR_HUD (#74) from 10.10 (v2.0) source 127.0.0.1
received LANDING_TARGET (#149) from 10.10 (v2.0) source 127.0.0.1
received SCALED_IMU (#26) from 10.10 (v2.0) source 127.0.0.1
received PARAM_REQUEST_READ (#20) from 10.10 to 192.168 (v2.0) source 127.0.0.1
```

-loglevel 3

In addition to everything in --loglevel 1 and --loglevel 2 this will log routed packets, indicating whether they are accepted or rejected, to stdcout. An example log output is:

```
new component 127.1 on 127.0.0.1
received HEARTBEAT (#0) from 127.1 (v2.0) source 127.0.0.1
rejected HEARTBEAT (#0) from 127.1 (v2.0) source 127.0.0.1 dest ./ttyS0
new component 192.168 on 127.0.0.1
received HEARTBEAT (#0) from 192.168 (v2.0) source 127.0.0.1
rejected HEARTBEAT (#0) from 192.168 (v2.0) source 127.0.0.1 dest ./ttyS0
rejected HEARTBEAT (#0) from 192.168 (v2.0) source 127.0.0.1 dest 127.0.0.1
new component 172.128 on ./ttyS0
received HEARTBEAT (#0) from 172.128 (v2.0) source ./ttyS0
rejected HEARTBEAT (#0) from 172.128 (v2.0) source ./ttyS0 dest 127.0.0.1
rejected HEARTBEAT (#0) from 172.128 (v2.0) source ./ttyS0 dest 127.0.0.1
new component 10.10 on 127.0.0.1
received HEARTBEAT (#0) from 10.10 (v2.0) source 127.0.0.1
rejected HEARTBEAT (#0) from 10.10 (v2.0) source 127.0.0.1 dest ./ttyS0
rejected HEARTBEAT (#0) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
rejected HEARTBEAT (#0) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
received POWER_STATUS (#125) from 10.10 (v2.0) source 127.0.0.1
rejected POWER_STATUS (#125) from 10.10 (v2.0) source 127.0.0.1 dest ./ttyS0
accepted POWER_STATUS (#125) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
rejected POWER_STATUS (#125) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
received SYS_STATUS (#1) from 10.10 (v2.0) source 127.0.0.1
accepted SYS_STATUS (#1) from 10.10 (v2.0) source 127.0.0.1 dest ./ttyS0
accepted SYS_STATUS (#1) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
accepted SYS_STATUS (#1) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
received GPS_RAW_INT (#24) from 10.10 (v2.0) source 127.0.0.1
rejected GPS_RAW_INT (#24) from 10.10 (v2.0) source 127.0.0.1 dest ./ttyS0
accepted GPS_RAW_INT (#24) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
accepted GPS_RAW_INT (#24) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
```

Abstract Syntax Tree

mavtables can print the abstract syntax tree of the configuration file instead of routing packets. When run in this mode it will return immediately This is accomplished with:

```
$ mavtables --ast
```

or

```
$ mavtables --ast --config path/to/config/file
```

which will print something similar to

```
===== /etc/mavtables.conf =====
:005: default_action
:005: | reject
:008: udp
:009: | port 14500
:010: | address 127.0.0.1
:022: chain default
:023: | accept
```

This feature can be used to debug configuration files.

For and explanation of configuration files see [Configuration](#).

Chapter 5

GNU General Public License

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: ****(1)**** copyright the software, and ****(2)**** offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. **If** the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program”, below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- **a)** You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- **b)** You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- **c)** If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the

author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

```
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License along
with this program; if not, write to the Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.
```

```
<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

Chapter 6

Contributing

When contributing to this repository, please first discuss the change you wish to make via issue, email, or any other method with the owners of this repository before making a change.

Pull Request Process

1. Create a pull request early in development processes with a checkbox list of features/changes you plan to make along with a description of what you intend to do. Prepend (WIP:) for "work in progress" to the title of this pull request until you are ready for review and merging.
2. Update README.rst and the Sphinx documentation with details of any changes to the interface, this includes new environment variables, exposed ports, and useful file locations.
3. Ensure test coverage did not drop below 90% and that all unit tests pass.
4. Ensure the package builds, installs, and runs using `pip install .` in a clean virtualenv. List any new external dependencies in the README.rst file.
5. Increase the version numbers in any examples files and the README.rst to the new version that this Pull Request would represent. The versioning scheme we use is [SemVer](#).
6. You may merge the Pull Request in once you have the sign-off of a core developer, or if you do not have permission to do that, you may request the reviewer merge it for you.

Git Etiquette

- Do not work on the main repository unless you are merging in changes, preparing for a version bump, or other repository management.
 - Create a fork, make a new branch and work on your changes there. Merge these changes in using the *Pull Request Process* above.
- Do not commit IDE/Editor specific files or any generated files. You should add these files to the `.gitignore` file to ensure you do not accidentally commit them.
- Do not commit application specific configuration files. If you wish to submit an example place it in the examples directory.

- Do not commit assets such as pictures or videos. The exception is small (under 300 kB) images used in the documentation.
- Do not commit other project's documentation such as PDF files.
 - Link to that projects documentation instead.
- Do not commit another project's source code.
 - Use a `Git submodule` instead.

Programming Style

- All C++17 features are allowed.
- Simply running the `style` rule of the Makefile will fix most style issues and should be used to ensure consistent style. This target will be ran before a merge to check for style consistency but does not check naming conventions. It does not always fix line length problems though. This is why the `style` rule also prints lines exceeding 80 characters that must be fixed manually.
- All type names such as classes, structs, type aliases, enums, and type template parameters should be CamelCase.
- Functions, variables, and namespaces should be `snake_case`.
- No name may begin with a single `_` (underscore) or contain `__` (double underscore).
- Private data members and functions should end in an underscore. have an `autodoc` entry in the Sphinx documentation.
- Each class should have it's own source and header file.
- When a file contains a class the file should be named for the class. All other file names should be lower case and should only include letters, numbers, and underscores.
- C++ source code files should use the `.cpp` extension and `.hpp` for header files. `.c` and `.h` should only be used for C files.
- All header files must have include guards and the name should be the file name (all uppercase) with `_HPP_` appended.
- All public functions and classes should contain `Doxygen style comments` and the recommended block format is

```
\**  
*  
*/
```

And the `\` (backslash) is recommended over the `@` (at symbol).

Chapter 7

mavtables

A MAVLink router and firewall. It can connect 2 or more MAVLink endpoints such as autopilots, ground control software, loggers, image capture systems, etc over serial and UDP. MAVLink packets will be routed to specific components when they have a destination address. Any packet, targeted or broadcasted, can be filtered based on source system/component, destination system/component and message type. The filter can also apply a priority to packets allowing more important packets to take priority over lower priority packets when an endpoint is choked.

Links

- [User Documentation](#)
- [HTML Developer Documentation](#)
- [PDF Developer Documentation](#)
- [Download](#)
- [GitHub](#)
- [Theory of MAVLink Routing](#)

Compilation and Installation

In order to compile you will need the following packages:

- GCC 7+ or Clang 5+ (needed for C++17 support)
- CMake v3.3+
- Boost v1.54+

Clang is recommended when contributing to mavtables as it's warnings are more comprehensive. However, GCC must be used when generating code coverage reports.

The following packages are only needed for development work:

- **Artistic Style** (used for checking/fixing the code style)
- **Gcovr** (coverage report)
- **LCOV** (detailed coverage html report)
- **socat** (for testing serial port communications)

mavtables can be easily installed using the standard procedure of

```
$ make
# make install
```

The installation prefix is `/usr/local` by default but can be changed with

```
$ make
# make PREFIX=/desired/install/path install
```

The makefile will download and use the default MAVLink implementation with the ArduPilot dialect. This can be overridden by setting the `MDIR` environment variable to the library path (containing the `DIALECT`) and/or the `DIALECT` environment variable to the MAVLink dialect to use. For instance the default value of `DIALECT` is `ardupilotmega`.

Running

To run mavtables and begin routing packets

```
$ mavtables
```

This will use the first configuration file it finds in the configuration file priority order given in the next section. To force a specific configuration file the `--config` flag may be used.

```
$ mavtables --config <path/to/config>
```

The inbuilt help may be accessed with the `-h` or `--help` flags.

```
$ mavtables --help
usage: mavtables:
-h [ --help ]           print this message
--config arg            specify configuration file
--ast                   print AST of configuration file (do not run)
--version               print version and license information
--loglevel arg          level of logging, between 0 and 3
```

Configuration File

Both interfaces and filter rules are defined in a configuration file. The format of this configuration file is documented in `'doc/configuration.md'` and an example is located at `examples/mavtables.conf` which is the same file that is installed at `/etc/mavtables.conf` when using `make install`. The configuration file used is the first one found in the following order:

1. The target of the `MAVTABLES_CONFIG_PATH` environment variable.
2. `.mavtablesrc` in the current directory.
3. `.mavtablesrc` at `$HOME/.mavtablesrc`.
4. The main configuration file at `/etc/mavtables.conf`.

If the `--config` flag is given then `mavtables` will only look for the given configuration file.

Contributing

Before contributing read the [CONTRIBUTING.md](#) file which gives guidelines that must be followed by all developers working on `mavtables`.

Chapter 8

Module Index

8.1 Modules

Here is a list of all modules:

Configuration functions	45
Macros	48
MAVLink Library and Helpers	49
Utility Functions	52

Chapter 9

Namespace Index

9.1 Namespace List

Here is a list of all namespaces with brief descriptions:

config	57
logger	58
mavlink	61
packet_scripter	61
packet_v1	64
packet_v2	67

Chapter 10

Hierarchical Index

10.1 Class Hierarchy

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

Action	78
AddressPool< TC >	92
AddressPool<>	92
App	96
Chain	105
ConfigParser	113
Connection	126
ConnectionFactory< C, AP, PQ >	132
ConnectionFactory<>	132
ConnectionPool	135
std::exception	
DNSLookupError	139
InvalidPacketIDError	174
PartialSendError	286
RecursionError	299
Filesystem	141
Filter	143
If	155
Interface	166
SerialInterface	320
UDPIInterface	334
InterfaceThreader	170
IPAddress	176
Logger	190
MAVAddress	194
MAVSubnet	208
MockCErr	217
MockCOut	219
Options	221
Packet	227
packet_v1::Packet	240

packet_v2::Packet	258
PacketParser	277
PacketQueue	282
QueuedPacket	289
RecursionData	296
RecursionGuard	301
Rule	310
Accept	71
Call	97
GoTo	148
Reject	304
semaphore	317
SerialPort	325
UnixSerialPort	345
UDPSocket	339
UnixUDPSocket	360
UnixSyscalls	353
mavlink::v1_header	367
mavlink::v2_header	369

Chapter 11

Class Index

11.1 Class List

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

Accept	71
Action	78
AddressPool< TC >	92
App	96
Call	97
Chain	105
ConfigParser	113
Connection	126
ConnectionFactory< C, AP, PQ >	132
ConnectionPool	135
DNSLookupError	139
Filesystem	141
Filter	143
GoTo	148
If	155
Interface	166
InterfaceThreader	170
InvalidPacketIDError	174
IPAddress	176
Logger	190
MAVAddress	194
MAVSubnet	208
MockCErr	217
MockCOut	219
Options	221
Packet	227
packet_v1::Packet	240
packet_v2::Packet	258
PacketParser	277
PacketQueue	282
PartialSendError	286

QueuedPacket	289
RecursionData	296
RecursionError	299
RecursionGuard	301
Reject	304
Rule	310
semaphore	317
SerialInterface	320
SerialPort	325
UDPIInterface	334
UDPSocket	339
UnixSerialPort	345
UnixSyscalls	353
UnixUDPSocket	360
mavlink::v1_header	367
mavlink::v2_header	369

Chapter 12

File Index

12.1 File List

Here is a list of all files with brief descriptions:

Accept.cpp	373
Accept.hpp	375
Action.cpp	377
Action.hpp	380
AddressPool.hpp	383
ansi_codes.sh	386
App.cpp	387
App.hpp	389
Call.cpp	390
Call.hpp	393
Chain.cpp	395
Chain.hpp	399
common.hpp	402
common_Packet.hpp	405
common_Rule.hpp	411
config_grammar.cpp	412
config_grammar.hpp	416
ConfigParser.cpp	422
ConfigParser.hpp	429
Connection.cpp	439
Connection.hpp	445
ConnectionFactory.hpp	447
ConnectionPool.cpp	450
ConnectionPool.hpp	451
DNSLookupError.cpp	454
DNSLookupError.hpp	456
Filesystem.cpp	457
Filesystem.hpp	458
Filter.cpp	460
Filter.hpp	461
GoTo.cpp	464

GoTo.hpp	467
If.cpp	469
If.hpp	473
Interface.cpp	477
Interface.hpp	478
InterfaceThreader.cpp	480
InterfaceThreader.hpp	482
InvalidPacketIDError.cpp	484
InvalidPacketIDError.hpp	486
IPAddress.cpp	487
IPAddress.hpp	494
Logger.cpp	503
Logger.hpp	505
logger.py	507
macros.hpp	509
MAVAddress.cpp	512
MAVAddress.hpp	516
mavlink.cpp	519
mavlink.hpp	521
MAVSubnet.cpp	524
MAVSubnet.hpp	529
mavtables.cpp	532
Options.cpp	535
Options.hpp	538
Packet.cpp	540
Packet.hpp	543
packet_scripter.py	546
PacketParser.cpp	553
PacketParser.hpp	556
PacketQueue.cpp	558
PacketQueue.hpp	561
PacketVersion1.cpp	563
PacketVersion1.hpp	567
PacketVersion2.cpp	569
PacketVersion2.hpp	574
parse_tree.hpp	576
PartialSendError.cpp	582
PartialSendError.hpp	583
QueuedPacket.cpp	584
QueuedPacket.hpp	588
RecursionData.hpp	593
RecursionError.cpp	595
RecursionError.hpp	596
RecursionGuard.cpp	597
RecursionGuard.hpp	598
Reject.cpp	600
Reject.hpp	602
Rule.cpp	604
Rule.hpp	605
unit_tests/run_tests.sh	608
integration_tests/run_tests.sh	609
semaphore.cpp	615
semaphore.hpp	616
SerialInterface.cpp	619

SerialInterface.hpp	621
SerialPort.cpp	623
SerialPort.hpp	625
test_Accept.cpp	628
test_Action.cpp	634
test_AddressPool.cpp	641
test_Call.cpp	647
test_Chain.cpp	655
test_config_grammar.cpp	664
test_ConfigParser.cpp	706
test_Connection.cpp	724
test_ConnectionFactory.cpp	747
test_ConnectionPool.cpp	751
test_DNSLookupError.cpp	757
test_Filesystem.cpp	759
test_Filter.cpp	760
test_GoTo.cpp	765
test_If.cpp	773
test_Interface.cpp	782
test_InterfaceThreader.cpp	786
test_IPAddress.cpp	789
test_Logger.cpp	798
test_MAVAddress.cpp	801
test_mavlink.cpp	808
test_MAVSubnet.cpp	810
test_Options.cpp	819
test_Packet.cpp	827
test_PacketParser.cpp	834
test_PacketQueue.cpp	839
test_PacketVersion1.cpp	846
test_PacketVersion2.cpp	861
test_PartialSendError.cpp	879
test_QueuedPacket.cpp	882
test_RecursionError.cpp	887
test_RecursionGuard.cpp	889
test_Reject.cpp	891
test_Rule.cpp	896
test_Rule_comparison.cpp	900
test_semaphore.cpp	902
test_SerialInterface.cpp	907
test_SerialPort.cpp	915
test_UDPInterface.cpp	920
test_UDPSocket.cpp	937
test_UinxSerialPort.cpp	942
test_UinxUDPSocket.cpp	955
test_utility.cpp	966
UDPInterface.cpp	972
UDPInterface.hpp	974
UDPSocket.cpp	976
UDPSocket.hpp	979
unit_tests.cpp	981
UnixSerialPort.cpp	982
UnixSerialPort.hpp	987
UnixSyscalls.cpp	989

UnixSyscalls.hpp	992
UnixUDPSocket.cpp	994
UnixUDPSocket.hpp	998
utility.cpp	1003
utility.hpp	1004

Chapter 13

Module Documentation

13.1 Configuration functions.

Functions

- std::ostream & [config::print_node](#) (std::ostream &os, const config::parse_tree::node &node, bool print_location, const std::string &prefix)
- std::ostream & [operator<<](#) (std::ostream &os, const config::parse_tree::node &node)

13.1.1 Detailed Description

These functions are used to parse the configuration file.

13.1.2 Function Documentation

13.1.2.1 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const config::parse_tree::node & node )
```

Print AST node to the given output stream.

This is the same as calling [config::print_node](#) with the location option set to true.

Parameters

<code>os</code>	The output stream to print to.
<code>node</code>	The node to print, also prints it's children.

Returns

The output stream.

Definition at line 229 of file [config_grammar.cpp](#).

References [print_node\(\)](#).

Here is the call graph for this function:



13.1.2.2 print_node()

```
std::ostream & config::print_node (
    std::ostream & os,
    const config::parse_tree::node & node,
    bool print_location,
    const std::string & prefix )
```

Print an AST node and all its children.

Parameters

<i>os</i>	The output stream to print to.
<i>node</i>	The node to print, also prints its children.
<i>print_location</i>	Set to true to print file and line numbers of each AST node.
<i>prefix</i>	A string to prefix to each AST element. This is reserved for internal use.

Returns

The output stream.

Definition at line 166 of file [config_grammar.cpp](#).

Here is the caller graph for this function:



13.2 Macros

Macros

- `#define PACKED __attribute__((packed))`

13.2.1 Detailed Description

Macros used elsewhere.

13.2.2 Macro Definition Documentation

13.2.2.1 PACKED

```
#define PACKED __attribute__((packed))
```

Enforce a packed structure.

A packed structure will not have any padding regardless of the typical alignment restrictions.

Example:

```
struct PACKED a_packed_structure
{
    uint8_t a;
    uint16_t b;
    uint32_t c;
}
```

Definition at line [41](#) of file [macros.hpp](#).

13.3 MAVLink Library and Helpers

Classes

- struct `mavlink::v1_header`
- struct `mavlink::v2_header`

Functions

- `std::string mavlink::name (unsigned long id)`
- `unsigned long mavlink::id (std::string name)`

13.3.1 Detailed Description

MAVLink utility macros, defines, types and functions.

Including this file also includes the main MAVLink C library. This will be the reference implementation using the ArduPilot dialect by default.

The dialect can be changed by setting the `DIALECT` environment variable. The default is `ardupilotmega`.

The MAVLink library can be changed with the `MDIR` environment variable which should point to the directory where the `DIALECT` directory can be found.

13.3.2 Function Documentation

13.3.2.1 `id()`

```
unsigned long mavlink::id (
    std::string name )
```

Get message ID from message name.

Parameters

<code>name</code>	The name of the MAVLink message to get the numeric ID of.
-------------------	---

Returns

The numeric ID of the message.

Exceptions

<code>std::invalid_argument</code>	if the given message name is not valid.
------------------------------------	---

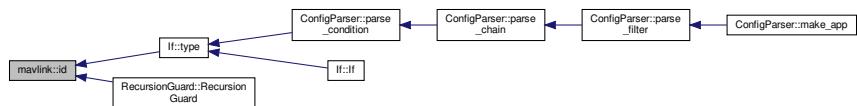
Definition at line 57 of file [mavlink.cpp](#).

References [name\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



13.3.2.2 name()

```
std::string mavlink::name (
    unsigned long id )
```

Get message name from numeric ID.

Parameters

<code>id</code>	The ID of the MAVLink message to get the name of.
-----------------	---

Returns

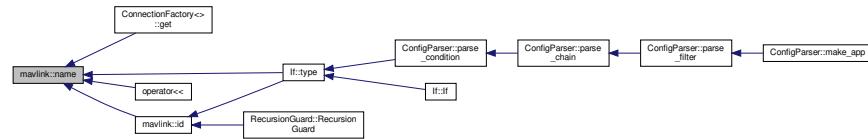
The name of the message.

Exceptions

<code>std::invalid_argument</code>	if the given <code>id</code> is not valid.
------------------------------------	--

Definition at line 35 of file [mavlink.cpp](#).

Here is the caller graph for this function:



13.4 Utility Functions

Functions

- std::string [to_lower](#) (std::string string)
- template<typename T>
std::vector< T >::iterator [append](#) (std::vector< T > &dest, const std::vector< T > &source)
- template<typename T>
std::vector< T >::iterator [append](#) (std::vector< T > &dest, std::vector< T > &&source)
- template<class T>
std::string [str](#) (const T &object)
- template<class ByteType = unsigned char, class T>
std::array< ByteType, sizeof(T)> [to_bytes](#) (T number)
- template<class T>
std::ostream & [operator<<](#) (std::ostream &os, const std::vector< T > &vector)

13.4.1 Detailed Description

Utility functions that don't warrant their own file.

13.4.2 Function Documentation

13.4.2.1 [append\(\)](#) [1/2]

```
template<typename T>
std::vector< T >::iterator append(
    std::vector< T > & dest,
    const std::vector< T > & source )
```

Append one vector to another.

Taken from <https://stackoverflow.com/a/37210097>

Parameters

<i>dest</i>	Vector to append to.
<i>source</i>	Vector to append the elements from.

Returns

Iterator pointing to the first element appended, or the end of the destination vector if the source vector is empty.

Definition at line [64](#) of file [utility.hpp](#).

13.4.2.2 `append()` [2/2]

```
template<typename T >
std::vector< T >::iterator append (
    std::vector< T > & dest,
    std::vector< T > && source )
```

Append one vector to another (move from source).

Taken from <https://stackoverflow.com/a/37210097>

Parameters

<i>dest</i>	Vector to append to.
<i>source</i>	Vector to append the elements from. <i>source</i> will be a valid empty vector after this call.

Returns

Iterator pointing to the first element appended, or the end of the destination vector if the source vector is empty.

Definition at line 96 of file [utility.hpp](#).

13.4.2.3 `operator<<()`

```
template<class T >
std::ostream & operator<< (
    std::ostream & os,
    const std::vector< T > & vector )
```

Print a vector to the given output stream.

Template Parameters

<i>T</i>	The type stored in the vector, it must support the <code><<</code> operator.
----------	--

Parameters

<i>os</i>	The output stream to print to.
<i>vector</i>	The vector of elements to print.

Returns

The output stream.

Definition at line 168 of file [utility.hpp](#).

13.4.2.4 str()

```
template<class T >
std::string str (
    const T & object )
```

Convert any object supporting the output stream operator (`<<`) to a string.

Template Parameters

<i>T</i>	Type of the object to convert to a string.
----------	--

Parameters

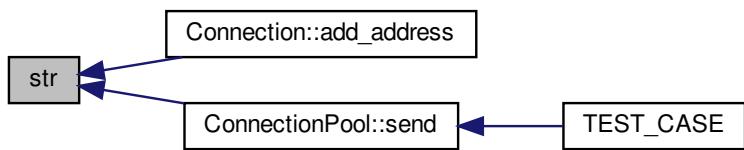
<i>object</i>	The object to convert to a string.
---------------	------------------------------------

Returns

The string representing the object.

Definition at line 128 of file [utility.hpp](#).

Here is the caller graph for this function:



13.4.2.5 to_bytes()

```
template<class ByteType = unsigned char, class T >
std::array< ByteType, sizeof(T)> to_bytes (
    T number )
```

Convert numeric types to bytes.

Template Parameters

<i>ByteType</i>	Numeric type to return in the array of bytes.
<i>T</i>	Type of the number being converted to bytes.

Parameters

<i>number</i>	Number to convert to bytes
---------------	----------------------------

Returns

The array of bytes from the given number, in LSB order.

Definition at line 145 of file [utility.hpp](#).

Here is the caller graph for this function:

**13.4.2.6 to_lower()**

```
std::string to_lower (
    std::string string )
```

Convert string to lower case.

Parameters

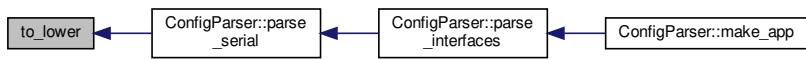
<i>string</i>	The string to convert to lower case.
---------------	--------------------------------------

Returns

The *string* converted to lower case.

Definition at line 37 of file [utility.cpp](#).

Here is the caller graph for this function:



Chapter 14

Namespace Documentation

14.1 config Namespace Reference

Functions

- std::ostream & `print_node` (std::ostream &os, const config::parse_tree::node &node, bool print_location, const std::string &prefix)
- template<typename Input>
std::unique_ptr<config::parse_tree::node> `parse` (Input &in)

14.1.1 Function Documentation

14.1.1.1 `parse()`

```
template<typename Input>
std::unique_ptr<config::parse_tree::node> config::parse (
    Input & in )
```

Parses given input into an abstract syntax tree.

Note

The returned AST is only valid while the input exists. Therefore, the input should be kept until the AST passes out of scope.

Template Parameters

<i>Input</i>	The type of input, usually either <code>read_input</code> or <code>string_input</code> (see PEGTL).
--------------	---

Parameters

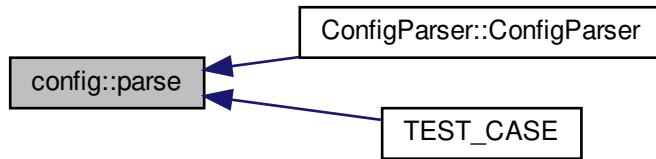
<i>in</i>	The input, from <code>read_input</code> etc, to parse.
-----------	--

Returns

The abstract syntax tree parsed from the input.

Definition at line 391 of file [config_grammar.hpp](#).

Here is the caller graph for this function:



14.2 logger Namespace Reference

Functions

- def [signal_handler](#) (signal, frame)
- def [parse_args](#) ()
- def [heartbeat](#) (mav)
- def [start_heartbeats](#) (mav)
- def [main](#) ()

14.2.1 Function Documentation

14.2.1.1 heartbeat()

```
def logger.heartbeat (
    mav )
```

Definition at line 42 of file [logger.py](#).

Here is the caller graph for this function:



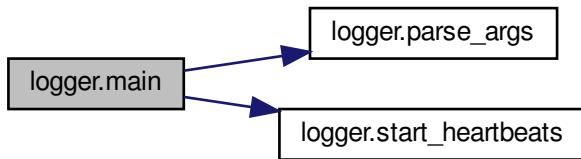
14.2.1.2 main()

```
def logger.main ( )
```

Definition at line 54 of file [logger.py](#).

References [parse_args\(\)](#), and [start_heartbeats\(\)](#).

Here is the call graph for this function:

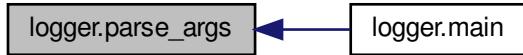


14.2.1.3 parse_args()

```
def logger.parse_args ( )
```

Definition at line 20 of file [logger.py](#).

Here is the caller graph for this function:



14.2.1.4 signal_handler()

```
def logger.signal_handler (
    signal,
    frame )
```

Definition at line 16 of file [logger.py](#).

14.2.1.5 start_heartbeats()

```
def logger.start_heartbeats (
    mav )
```

Definition at line 48 of file [logger.py](#).

Here is the caller graph for this function:



14.3 mavlink Namespace Reference

Classes

- struct [v1_header](#)
- struct [v2_header](#)

Functions

- std::string [name](#) (unsigned long [id](#))
- unsigned long [id](#) (std::string [name](#))

14.4 packet_scripter Namespace Reference

Functions

- def [send_packet](#) (mav, packet, system=0, component=0)
- def [parse_line](#) (line)
- def [parse_file](#) (filename)
- def [start_connection](#) (args)
- def [parse_args](#) ()
- def [main](#) ()

14.4.1 Function Documentation

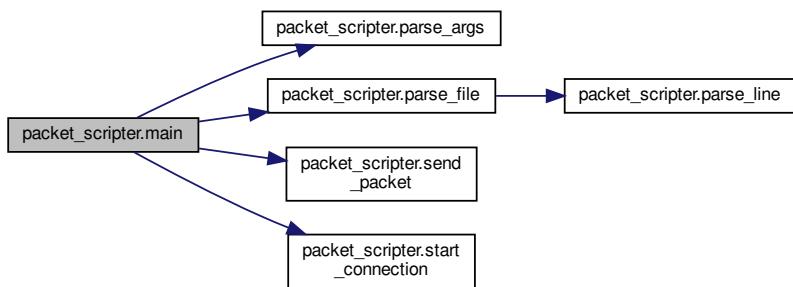
14.4.1.1 [main\(\)](#)

```
def packet_scripter.main ( )
```

Definition at line 480 of file [packet_scripter.py](#).

References [parse_args\(\)](#), [parse_file\(\)](#), [send_packet\(\)](#), and [start_connection\(\)](#).

Here is the call graph for this function:



14.4.1.2 parse_args()

```
def packet_scripter.parse_args ( )
```

Definition at line 458 of file [packet_scripter.py](#).

Here is the caller graph for this function:



14.4.1.3 parse_file()

```
def packet_scripter.parse_file ( filename )
```

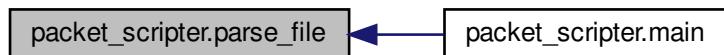
Definition at line 437 of file [packet_scripter.py](#).

References [parse_line\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

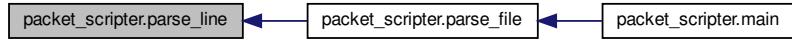


14.4.1.4 parse_line()

```
def packet_scripter.parse_line (
    line )
```

Definition at line 426 of file [packet_scripter.py](#).

Here is the caller graph for this function:



14.4.1.5 send_packet()

```
def packet_scripter.send_packet (
    mav,
    packet,
    system = 0,
    component = 0 )
```

Definition at line 16 of file [packet_scripter.py](#).

Here is the caller graph for this function:

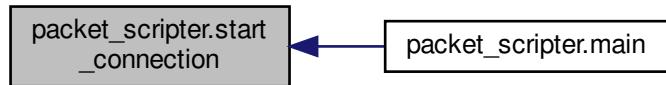


14.4.1.6 start_connection()

```
def packet_scripter.start_connection (
    args )
```

Definition at line 443 of file [packet_scripter.py](#).

Here is the caller graph for this function:



14.5 packet_v1 Namespace Reference

Classes

- class [Packet](#)

Variables

- const uint8_t [START_BYTE](#) = MAVLINK_STX_MAVLINK1
- const size_t [HEADER_LENGTH](#) = 1 + MAVLINK_CORE_HEADER_MAVLINK1_LEN
- const size_t [CHECKSUM_LENGTH](#) = MAVLINK_NUM_CHECKSUM_BYTES
- const ::Packet::Version [VERSION](#) = ::Packet::V1

14.5.1 Function Documentation

14.5.1.1 header()

```
const struct mavlink::v1_header* packet_v1::header (
    const std::vector< uint8_t > & data ) [related]
```

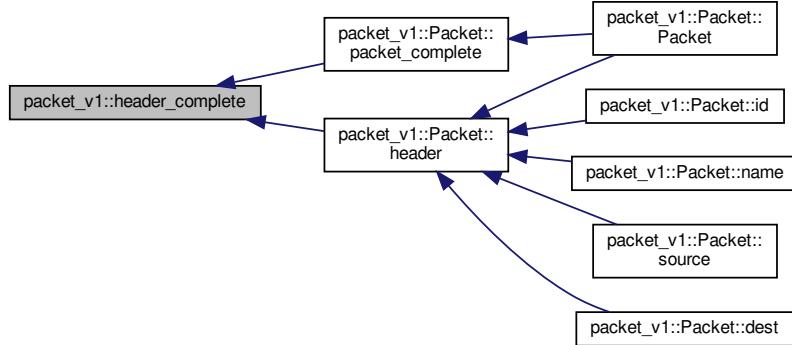
Here is the caller graph for this function:



14.5.1.2 header_complete()

```
bool packet_v1::header_complete (
    const std::vector< uint8_t > & data ) [related]
```

Here is the caller graph for this function:



14.5.1.3 packet_complete()

```
bool packet_v1::packet_complete (
    const std::vector< uint8_t > & data ) [related]
```

14.5.2 Variable Documentation

14.5.2.1 CHECKSUM_LENGTH

```
const size_t packet_v1::CHECKSUM_LENGTH = MAVLINK_NUM_CHECKSUM_BYTES
```

MAVLink v1.0 checksum length (2 bytes).

Definition at line [46](#) of file [PacketVersion1.hpp](#).

14.5.2.2 HEADER_LENGTH

```
const size_t packet_v1::HEADER_LENGTH = 1 + MAVLINK_CORE_HEADER_MAVLINK1_LEN
```

MAVLink v1.0 header length (6 bytes).

Definition at line [41](#) of file [PacketVersion1.hpp](#).

14.5.2.3 START_BYTE

```
const uint8_t packet_v1::START_BYTE = MAVLINK_STX_MAVLINK1
```

MAVLink v1.0 start byte (0xFE).

Definition at line [36](#) of file [PacketVersion1.hpp](#).

14.5.2.4 VERSION

```
const ::Packet::Version packet_v1::VERSION = ::Packet::V1
```

MAVLink v1.0 version.

Definition at line [51](#) of file [PacketVersion1.hpp](#).

14.6 packet_v2 Namespace Reference

Classes

- class [Packet](#)

Variables

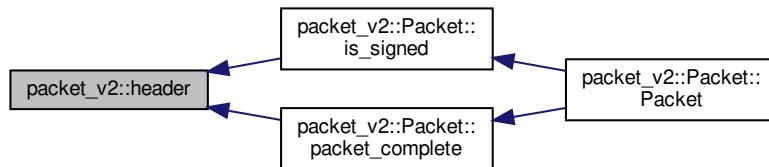
- const uint8_t [START_BYTE](#) = MAVLINK_STX
- const size_t [HEADER_LENGTH](#) = MAVLINK_NUM_HEADER_BYTES
- const size_t [CHECKSUM_LENGTH](#) = MAVLINK_NUM_CHECKSUM_BYTES
- const size_t [SIGNATURE_LENGTH](#) = MAVLINK_SIGNATURE_BLOCK_LEN
- const ::[Packet](#)::Version [VERSION](#) = ::[Packet](#)::V2

14.6.1 Function Documentation

14.6.1.1 [header\(\)](#)

```
const struct mavlink::v2\_header* packet\_v2::header (
    const std::vector< uint8_t > & data ) [related]
```

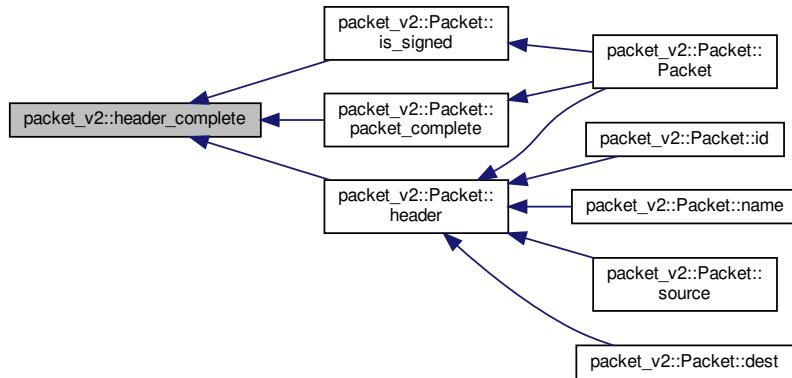
Here is the caller graph for this function:



14.6.1.2 header_complete()

```
bool packet_v2::header_complete (
    const std::vector< uint8_t > & data ) [related]
```

Here is the caller graph for this function:



14.6.1.3 is_signed()

```
bool packet_v2::is_signed (
    const std::vector< uint8_t > & data ) [related]
```

14.6.1.4 packet_complete()

```
bool packet_v2::packet_complete (
    const std::vector< uint8_t > & data ) [related]
```

14.6.2 Variable Documentation

14.6.2.1 CHECKSUM_LENGTH

```
const size_t packet_v2::CHECKSUM_LENGTH = MAVLINK_NUM_CHECKSUM_BYTES
```

MAVLink v2.0 checksum length (2 bytes).

Definition at line 46 of file [PacketVersion2.hpp](#).

14.6.2.2 HEADER_LENGTH

```
const size_t packet_v2::HEADER_LENGTH = MAVLINK_NUM_HEADER_BYTES
```

MAVLink v2.0 header length (10 bytes).

Definition at line 41 of file [PacketVersion2.hpp](#).

14.6.2.3 SIGNATURE_LENGTH

```
const size_t packet_v2::SIGNATURE_LENGTH = MAVLINK_SIGNATURE_BLOCK_LEN
```

MAVLink v2.0 signature length (13 bytes) if signed.

Definition at line 51 of file [PacketVersion2.hpp](#).

14.6.2.4 START_BYTE

```
const uint8_t packet_v2::START_BYTE = MAVLINK_STX
```

MAVLink v2.0 start byte (0xFD).

Definition at line 36 of file [PacketVersion2.hpp](#).

14.6.2.5 VERSION

```
const ::Packet::Version packet_v2::VERSION = ::Packet::V2
```

MAVLink v2.0 version..

Definition at line 56 of file [PacketVersion2.hpp](#).

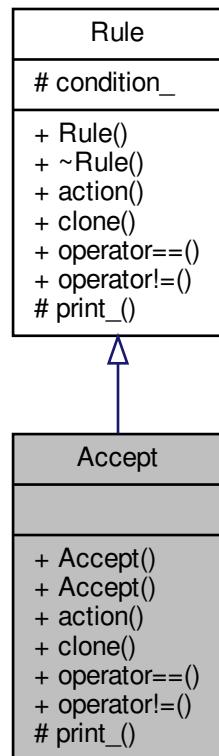
Chapter 15

Class Documentation

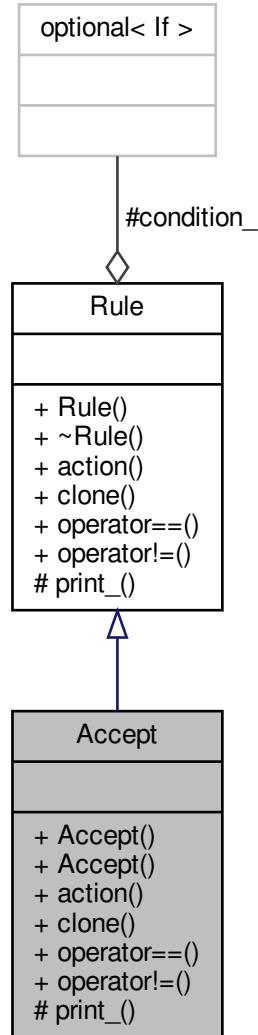
15.1 Accept Class Reference

```
#include <Accept.hpp>
```

Inheritance diagram for Accept:



Collaboration diagram for Accept:



Public Member Functions

- `Accept (std::optional< If > condition={})`
- `Accept (int priority, std::optional< If > condition={})`
- virtual `Action action (const Packet &packet, const MAVAddress &address) const`
- virtual `std::unique_ptr< Rule > clone () const`
- virtual `bool operator== (const Rule &other) const`
- virtual `bool operator!= (const Rule &other) const`

Protected Member Functions

- virtual std::ostream & [print_](#) (std::ostream &os) const

Protected Attributes

- std::optional< [If](#) > [condition_](#)

15.1.1 Detailed Description

[Rule](#) to accept a packet, optionally with a priority.

Definition at line 35 of file [Accept.hpp](#).

15.1.2 Constructor & Destructor Documentation

15.1.2.1 Accept() [1/2]

```
Accept::Accept (
    std::optional< If > condition = {} )
```

Construct an accept rule, without a priority.

An accept rule is used to accept packet/address combinations that match the condition of the rule.

Parameters

condition	The condition used to determine the rule matches a particular packet/address combination given to the action method. The default is {} which indicates the rule matches any packet/address combination.
---------------------------	---

See also

[action](#)

Definition at line 42 of file [Accept.cpp](#).

15.1.2.2 Accept() [2/2]

```
Accept::Accept (
    int priority,
    std::optional< If > condition = {} )
```

Construct an accept rule, with a priority.

An accept rule is used to accept packet/address combinations that match the condition of the rule.

Parameters

<i>condition</i>	The condition used to determine the rule matches a particular packet/address combination given to the action method. The default is {} which indicates the rule matches any packet/address combination.
<i>priority</i>	The priority to accept packets with. A higher number is more important and will be routed first.

See also

[action](#)

Definition at line 61 of file [Accept.cpp](#).

15.1.3 Member Function Documentation

15.1.3.1 [action\(\)](#)

```
Action Accept::action (
    const Packet & packet,
    const MAVAddress & address ) const [virtual]
```

Decide what to do with a [Packet](#).

Determine what action to take with the given packet sent to the given address. The possible actions are documented in the [Action](#) class. The continue object is always returned if the condition was set and does not match the packet/address combination.

Parameters

<i>packet</i>	The packet to determine whether to allow or not.
<i>address</i>	The address the <i>packet</i> will be sent out on if the action dictates it.

Returns

The action to take with the packet. If this is the accept object, it may also contain a priority for the packet.

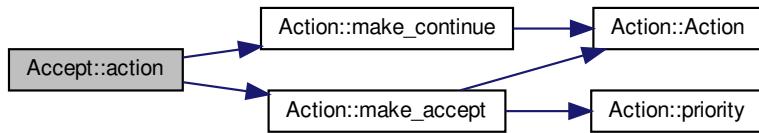
If the condition has not been set or it matches the given packet/address combination then it will return the accept [Action](#) object (with optional priority), otherwise it will return the continue [Action](#) object.

Implements [Rule](#).

Definition at line 99 of file [Accept.cpp](#).

References [Rule::condition_](#), [Action::make_accept\(\)](#), and [Action::make_continue\(\)](#).

Here is the call graph for this function:



15.1.3.2 clone()

```
std::unique_ptr< Rule > Accept::clone( ) const [virtual]
```

Return a copy of the [Rule](#) polymorphically.

This allows [Rule](#)'s to be copied without knowing the derived type.

Returns

A pointer to a new object with base type [Rule](#) which is an exact copy of this one.

Implements [Rule](#).

Definition at line 111 of file [Accept.cpp](#).

References [Rule::condition_](#).

15.1.3.3 operator"!=()

```
bool Accept::operator!=(
    const Rule & other ) const [virtual]
```

Inequality comparison.

Compares the type of the [Rule](#) and the condition ([If](#)) if set.

Parameters

<i>other</i>	The other rule to compare this to.
--------------	------------------------------------

Return values

<i>true</i>	if this rule is not the same as <i>other</i> .
<i>false</i>	if this rule is the same as <i>other</i> .

Compares the priority (if set) associated with the rule as well.

Implements [Rule](#).

Definition at line 138 of file [Accept.cpp](#).

References [Rule::condition_](#).

15.1.3.4 operator==()

```
bool Accept::operator== (
    const Rule & other ) const [virtual]
```

Equality comparison.

Compares the type of the [Rule](#) and the condition ([If](#)) if set.

Parameters

<i>other</i>	The other rule to compare this to.
--------------	------------------------------------

Return values

<i>true</i>	if this rule is the same as <i>other</i> .
<i>false</i>	if this rule is not the same as <i>other</i> .

Compares the priority (if set) associated with the rule as well.

Implements [Rule](#).

Definition at line 126 of file [Accept.cpp](#).

References [Rule::condition_](#).

15.1.3.5 print_()

```
std::ostream & Accept::print_ (
    std::ostream & os ) const [protected], [virtual]
```

Print the rule to the given output stream.

Parameters

<code>os</code>	The output stream to print to.
-----------------	--------------------------------

Returns

The output stream.

Prints "reject <Priority> <If Statement>" where the priority and if statement are only printed if the priority or condition is set, respectively.

Parameters

<code>os</code>	The output stream to print to.
-----------------	--------------------------------

Implements [Rule](#).

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

References [Rule::condition_](#).

15.1.4 Member Data Documentation

15.1.4.1 condition_

```
std::optional<If> Rule::condition_ [protected], [inherited]
```

Definition at line [91](#) of file [Rule.hpp](#).

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

- [Accept.hpp](#)
- [Accept.cpp](#)

15.2 Action Class Reference

```
#include <Action.hpp>
```

Collaboration diagram for Action:

Action
+ Action() + Action() + action() + priority() + priority() + operator=() + operator=() + make_accept() + make_reject() + make_continue() + make_default()

Public Types

- enum [Option](#) { [ACCEPT](#), [REJECT](#), [CONTINUE](#), [DEFAULT](#) }

Public Member Functions

- [Action \(const Action &other\)=default](#)
- [Action \(Action &&other\)=default](#)
- [Action::Option action \(\) const](#)
- [void priority \(int priority\)](#)
- [int priority \(\) const](#)
- [Action & operator= \(const Action &other\)=default](#)
- [Action & operator= \(Action &&other\)=default](#)

Static Public Member Functions

- static [Action make_accept \(std::optional< int > priority={}\)](#)
- static [Action make_reject \(\)](#)
- static [Action make_continue \(\)](#)
- static [Action make_default \(\)](#)

Related Functions

(Note that these are not member functions.)

- bool `operator==` (const `Action` &lhs, const `Action` &rhs)
- bool `operator!=` (const `Action` &lhs, const `Action` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `Action` &action)

15.2.1 Detailed Description

An action that is to be taken with a packet.

This is used as a return value to determine what to do with a packet.

Definition at line 30 of file [Action.hpp](#).

15.2.2 Member Enumeration Documentation

15.2.2.1 Option

```
enum Action::Option
```

Possible actions.

Enumerator

ACCEPT	The packet has been accepted, possibly with priority.
REJECT	The packet has been rejected.
CONTINUE	Continue evaluating rules.
DEFAULT	Use the default rule.

Definition at line 35 of file [Action.hpp](#).

15.2.3 Constructor & Destructor Documentation

15.2.3.1 Action() [1/2]

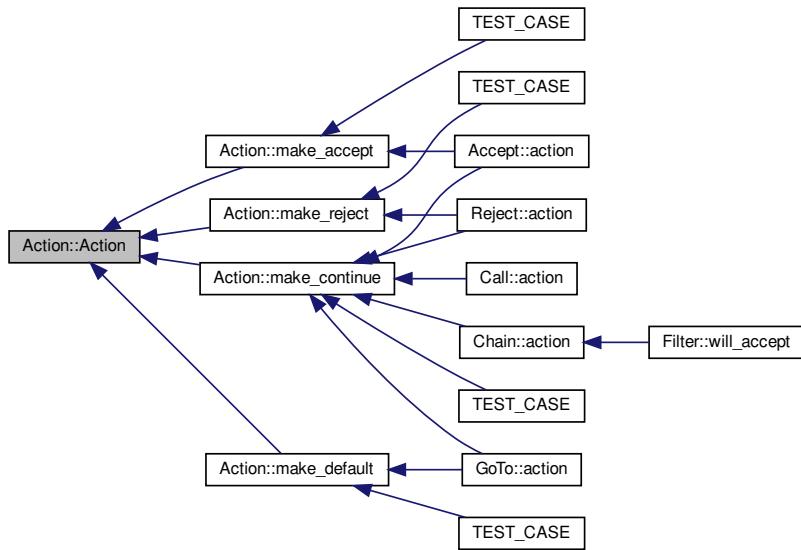
```
Action::Action (
    const Action & other ) [default]
```

Copy constructor.

Parameters

<i>other</i>	Action to copy from.
--------------	----------------------

Here is the caller graph for this function:

**15.2.3.2 Action() [2/2]**

```
Action::Action (
    Action && other ) [default]
```

Move constructor.

Parameters

<i>other</i>	Action to move from.
--------------	----------------------

15.2.4 Member Function Documentation

15.2.4.1 `action()`

```
Action::Option Action::action () const
```

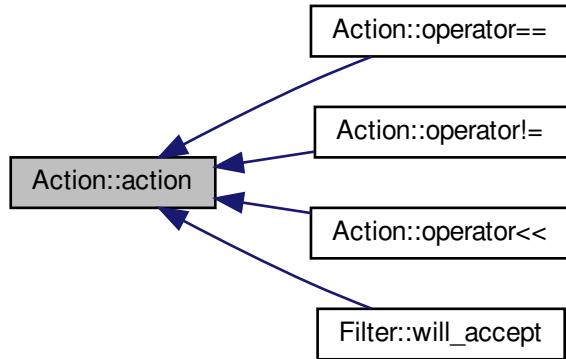
Return the action that has been chosen.

Returns

The `Action::Option` enum associated with this action.

Definition at line 45 of file `Action.cpp`.

Here is the caller graph for this function:



15.2.4.2 `make_accept()`

```
Action Action::make_accept (
    std::optional< int > priority = {} ) [static]
```

Make a new action result with the `Action::ACCEPT` action.

An accept action indicates that the packet/address combination this action is the response to should be accepted without any further processing.

Parameters

<code>priority</code>	The priority to accept the packet with. The default is to not apply a priority.
-----------------------	---

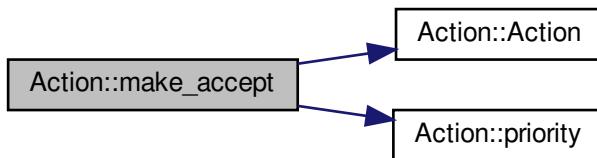
Returns

The new 'accept' action.

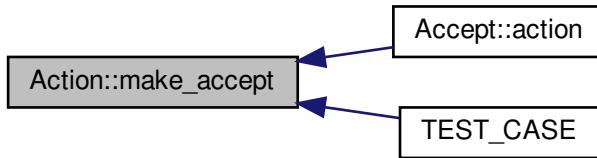
Definition at line 100 of file [Action.cpp](#).

References [ACCEPT](#), [Action\(\)](#), and [priority\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.2.4.3 make_continue()

[Action](#) `Action::make_continue () [static]`

Make a new action result with the [Action::CONTINUE](#) action.

A continue action indicates that filtering of the packet/address combination this action is the response should continue with the next [Rule](#).

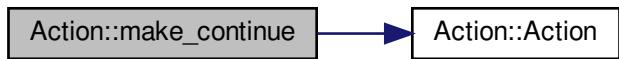
Returns

The new 'continue' action.

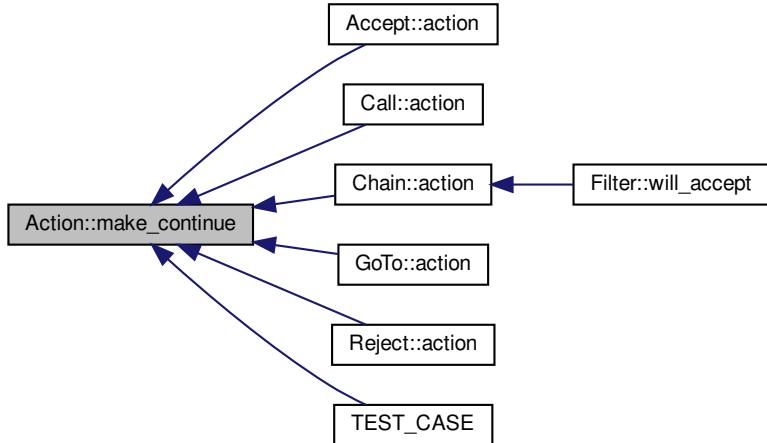
Definition at line 126 of file [Action.cpp](#).

References [Action\(\)](#), and [CONTINUE](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**15.2.4.4 make_default()**

[Action](#) `Action::make_default () [static]`

Make a new action result with the [Action::DEFAULT](#) action.

A default action indicates that the default action (defined in [Filter](#)) should be taken for the packet/address combination this action is the response to.

Returns

The new 'default' action.

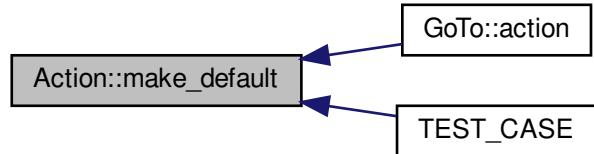
Definition at line 140 of file [Action.cpp](#).

References [Action\(\)](#), and [DEFAULT](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.2.4.5 make_reject()

[Action](#) `Action::make_reject () [static]`

Make a new action result with the [Action::REJECT](#) action.

A reject action indicates that the packet/address combination this action is the response to should be rejected without any further processing.

Returns

The new 'reject' action.

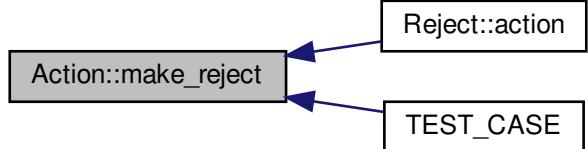
Definition at line 113 of file [Action.cpp](#).

References [Action\(\)](#), and [REJECT](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.2.4.6 operator=() [1/2]

```
Action& Action::operator= (
    const Action & other ) [default]
```

Assignment operator.

Parameters

<code>other</code>	<code>Action</code> to copy from.
--------------------	-----------------------------------

15.2.4.7 operator=() [2/2]

```
Action& Action::operator= (
    Action && other )  [default]
```

Assignment operator (by move semantics).

Parameters

<i>other</i>	Action to move from.
--------------	----------------------

15.2.4.8 priority() [1/2]

```
void Action::priority (
    int priority )
```

Set the priority of the action.

This only has an effect if the [action](#) is [Action::ACCEPT](#) and the priority has never been set before.

The default priority is 0. A higher priority will result in the packet being prioritized over other packets while a lower (negative) priority will de-prioritize the packet.

Parameters

<i>priority</i>	The priority to apply to the accept action.
-----------------	---

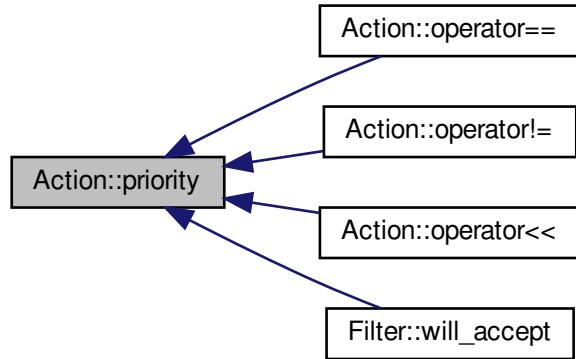
Definition at line [62](#) of file [Action.cpp](#).

References [ACCEPT](#), and [priority\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.2.4.9 priority() [2/2]

```
int Action::priority ( ) const
```

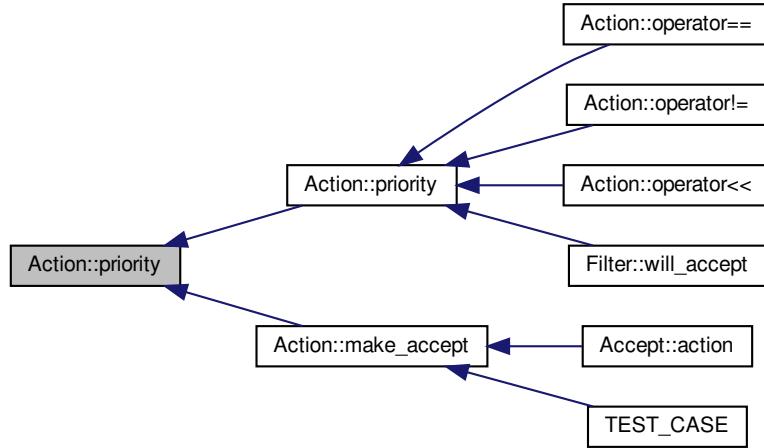
Return the priority if it exists.

Returns

The priority of the action. This will always be 0 if the `action` is not `Action::ACCEPT`. It will also be 0 (the default priority) if the priority has never been set.

Definition at line 80 of file [Action.cpp](#).

Here is the caller graph for this function:



15.2.5 Friends And Related Function Documentation

15.2.5.1 `operator"!="()`

```

bool operator!= (
    const Action & lhs,
    const Action & rhs ) [related]
  
```

Inequality comparison.

Parameters

<code>lhs</code>	The left hand side action.
<code>rhs</code>	The right hand side action.

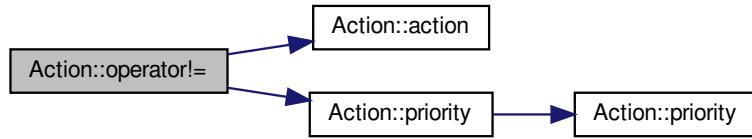
Return values

<code>true</code>	if <code>lhs</code> is not the same as <code>rhs</code> .
<code>false</code>	if <code>lhs</code> is the same as <code>rhs</code> .

Definition at line 168 of file [Action.cpp](#).

References [action\(\)](#), and [priority\(\)](#).

Here is the call graph for this function:



15.2.5.2 `operator<<()`

```
std::ostream & operator<< (
    std::ostream & os,
    const Action & action ) [related]
```

Print the action to the given output stream.

Some examples are:

- accept
- accept with priority 3
- reject
- continue
- default

Parameters

<code>os</code>	The output stream to print to.
<code>action</code>	The action result to print.

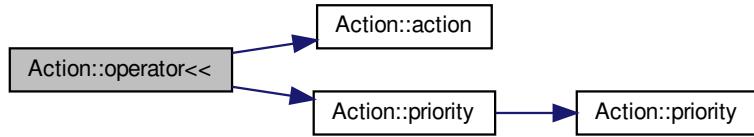
Returns

The output stream.

Definition at line 188 of file [Action.cpp](#).

References [ACCEPT](#), [action\(\)](#), [CONTINUE](#), [DEFAULT](#), [priority\(\)](#), and [REJECT](#).

Here is the call graph for this function:



15.2.5.3 operator==()

```
bool operator== (
    const Action & lhs,
    const Action & rhs ) [related]
```

Equality comparison.

Parameters

<i>lhs</i>	The left hand side action.
<i>rhs</i>	The right hand side action.

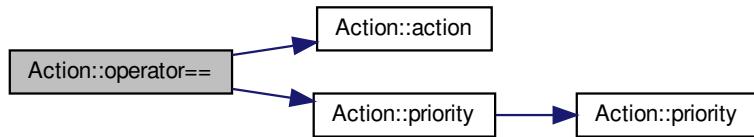
Return values

<i>true</i>	if <i>lhs</i> is the same as <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is not the same as <i>rhs</i> .

Definition at line 154 of file [Action.cpp](#).

References [action\(\)](#), and [priority\(\)](#).

Here is the call graph for this function:



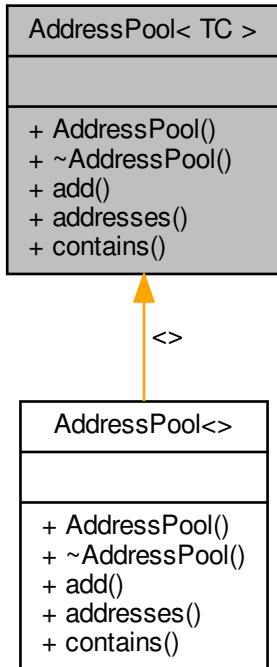
The documentation for this class was generated from the following files:

- [Action.hpp](#)
- [Action.cpp](#)

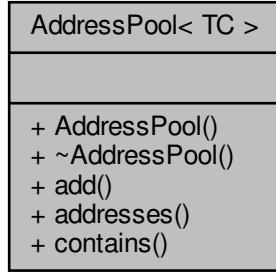
15.3 AddressPool< TC > Class Template Reference

```
#include <AddressPool.hpp>
```

Inheritance diagram for AddressPool< TC >:



Collaboration diagram for AddressPool< TC >:



Public Member Functions

- `AddressPool (std::chrono::milliseconds timeout=std::chrono::milliseconds(120000))`
- TEST_VIRTUAL `~AddressPool ()`=default
- TEST_VIRTUAL void `add (MAVAddress address)`
- TEST_VIRTUAL `std::vector< MAVAddress > addresses ()`
- TEST_VIRTUAL bool `contains (const MAVAddress &address)`

15.3.1 Detailed Description

```
template<class TC = std::chrono::steady_clock>
class AddressPool< TC >
```

A threadsafe container for addresses that expire after a given time.

Definition at line 34 of file [AddressPool.hpp](#).

15.3.2 Constructor & Destructor Documentation

15.3.2.1 AddressPool()

```
template<class TC >
AddressPool< TC >::AddressPool (
    std::chrono::milliseconds timeout = std::chrono::milliseconds(120000) )
```

Construct a new address pool.

Parameters

<i>timeout</i>	The amount of time (in milliseconds) before a component will be considered offline and removed from the pool, unless its time is updated with add .
----------------	---

Definition at line 60 of file [AddressPool.hpp](#).

15.3.2.2 ~AddressPool()

```
template<class TC = std::chrono::steady_clock>
TEST_VIRTUAL AddressPool< TC >::~AddressPool( ) [default]
```

15.3.3 Member Function Documentation**15.3.3.1 add()**

```
template<class TC >
void AddressPool< TC >::add(
    MAVAddress address)
```

Add a MAVLink address to the pool.

Note

Addresses will be removed after the timeout (set in the constructor) has run out. Re-adding the address (even before this time runs out) will reset the timeout.

Parameters

<i>address</i>	The MAVLink address to add or update the timeout for.
----------------	---

Remarks

Threadsafe (locking).

Definition at line 77 of file [AddressPool.hpp](#).

15.3.3.2 addresses()

```
template<class TC >
std::vector< MAVAddress > AddressPool< TC >::addresses ( )
```

Get a vector of all the addresses in the pool.

Note

A copy is returned instead of using iterators in order to make the call thread safe.

Returns

A vector of the addresses in the pool.

Remarks

Threadssafe (locking).

Definition at line 94 of file [AddressPool.hpp](#).

15.3.3.3 contains()

```
template<class TC >
bool AddressPool< TC >::contains (
    const MAVAddress & address )
```

Determine if the pool contains a given MAVLink address.

Parameters

<code>address</code>	The MAVLink address to test for.
----------------------	----------------------------------

Return values

<code>true</code>	If the pool contains address.
<code>false</code>	If the pool does not contain address.

Remarks

Threadssafe (locking).

Definition at line 129 of file [AddressPool.hpp](#).

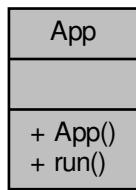
The documentation for this class was generated from the following file:

- [AddressPool.hpp](#)

15.4 App Class Reference

```
#include <App.hpp>
```

Collaboration diagram for App:



Public Member Functions

- [App \(std::vector< std::unique_ptr< Interface >> interfaces\)](#)
- [void run \(\)](#)

15.4.1 Detailed Description

The mavtables application class.

Definition at line 31 of file [App.hpp](#).

15.4.2 Constructor & Destructor Documentation

15.4.2.1 App()

```
App::App (
```

```
          std::vector< std::unique_ptr< Interface >> interfaces )
```

Construct mavtables application from a vector of interfaces.

Neither the interfaces, nor the application will be started until the [run](#) method is called.

Parameters

<i>interfaces</i>	A vector of interfaces.
-------------------	-------------------------

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

References [InterfaceThreader::DELAY_START](#).

15.4.3 Member Function Documentation

15.4.3.1 run()

```
void App::run ( )
```

Start the application.

This starts listening on all interfaces.

Exceptions

<i>std::system_error</i>	if an error is generated while waiting for Ctrl+C.
<i>std::runtime_error</i>	if run on Microsoft Windows.

Definition at line 59 of file [App.cpp](#).

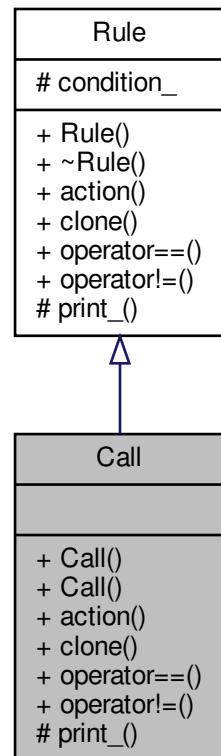
The documentation for this class was generated from the following files:

- [App.hpp](#)
- [App.cpp](#)

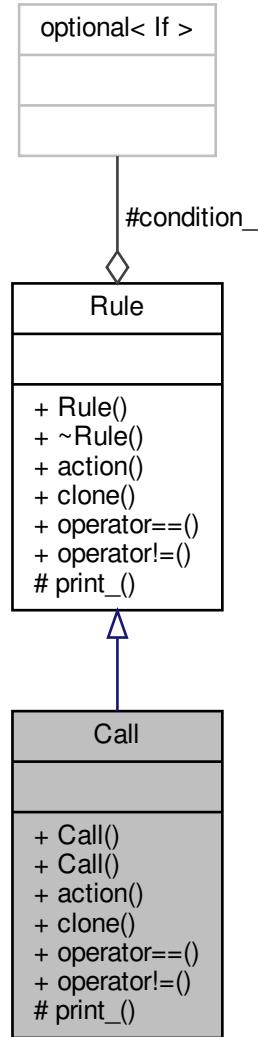
15.5 Call Class Reference

```
#include <Call.hpp>
```

Inheritance diagram for Call:



Collaboration diagram for Call:



Public Member Functions

- `Call (std::shared_ptr< Chain > chain, std::optional< If > condition={})`
- `Call (std::shared_ptr< Chain > chain, int priority, std::optional< If > condition={})`
- `virtual Action action (const Packet &packet, const MAVAddress &address) const`
- `virtual std::unique_ptr< Rule > clone () const`
- `virtual bool operator== (const Rule &other) const`
- `virtual bool operator!= (const Rule &other) const`

Protected Member Functions

- virtual std::ostream & [print_](#) (std::ostream &os) const

Protected Attributes

- std::optional< [If](#) > [condition_](#)

15.5.1 Detailed Description

Delegate decision on a packet to another [Chain](#).

[Rule](#) to delegate the decision on what to do with a packet to a filter [Chain](#). If this chain cannot make a determination (continue action returned), [Rule](#) evaluation should resume after this rule.

Definition at line 43 of file [Call.hpp](#).

15.5.2 Constructor & Destructor Documentation

15.5.2.1 Call() [1/2]

```
Call::Call (
    std::shared_ptr< Chain > chain,
    std::optional< If > condition = {} )
```

Construct a call rule given a chain to delegate to, without a priority.

A call rule is used to delegate the decision on whether to accept or reject a packet/address combination to another filter [Chain](#). [If](#) this called chain does not make a decision then rule evaluation should continue in the calling chain.

Parameters

<i>chain</i>	The chain to delegate decisions of whether to accept or reject a packet/address combination to. <code>nullptr</code> is not valid.
<i>condition</i>	The condition used to determine the rule matches a particular packet/address combination given to the action method. The default is <code>{}</code> which indicates the rule matches any packet/address combination.

Exceptions

<code>std::invalid_argument</code>	if the given <code>chain</code> pointer is null.
------------------------------------	--

See also

[action](#)

Definition at line 48 of file [Call.cpp](#).

15.5.2.2 Call() [2/2]

```
Call::Call (
    std::shared_ptr< Chain > chain,
    int priority,
    std::optional< If > condition = {} )
```

Construct a call rule given a chain to delegate to, with a priority.

A call rule is used to delegate the decision on whether to accept or reject a packet/address combination to another filter [Chain](#).

Parameters

<i>chain</i>	The chain to delegate decisions of whether to accept or reject a packet/address combination to. nullptr is not valid.
<i>priority</i>	The priority to accept packets with. A higher number is more important and will be routed first.
<i>condition</i>	The condition used to determine the rule matches a particular packet/address combination given to the action method. The default is {} which indicates the rule matches any packet/address combination.

Exceptions

<code>std::invalid_argument</code>	if the given pointer is null.
------------------------------------	-------------------------------

See also

[action](#)

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

15.5.3 Member Function Documentation

15.5.3.1 `action()`

```
Action Call::action (
    const Packet & packet,
    const MAVAddress & address ) const [virtual]
```

Decide what to do with a [Packet](#).

Determine what action to take with the given packet sent to the given address. The possible actions are documented in the [Action](#) class. The continue object is always returned if the condition was set and does not match the packet/address combination.

Parameters

<code>packet</code>	The packet to determine whether to allow or not.
<code>address</code>	The address the <code>packet</code> will be sent out on if the action dictates it.

Returns

The action to take with the packet. If this is the accept object, it may also contain a priority for the packet.

If the condition has not been set or it matches the given packet/address combination then the choice of [Action](#) will be delegated to the contained [Chain](#).

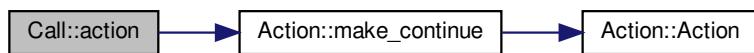
If the result from the chain is an accept [Action](#) object and no priority has been set on it but this [Rule](#) has a priority then the priority will be set.

Implements [Rule](#).

Definition at line 121 of file [Call.cpp](#).

References [Rule::condition_](#), and [Action::make_continue\(\)](#).

Here is the call graph for this function:



15.5.3.2 clone()

```
std::unique_ptr< Rule > Call::clone ( ) const [virtual]
```

Return a copy of the [Rule](#) polymorphically.

This allows [Rule](#)'s to be copied without knowing the derived type.

Returns

A pointer to a new object with base type [Rule](#) which is an exact copy of this one.

Implements [Rule](#).

Definition at line 142 of file [Call.cpp](#).

References [Rule::condition_](#).

15.5.3.3 operator"!=()

```
bool Call::operator!= (
    const Rule & other ) const [virtual]
```

Inequality comparison.

Compares the type of the [Rule](#) and the condition ([If](#)) if set.

Parameters

<i>other</i>	The other rule to compare this to.
--------------	------------------------------------

Return values

<i>true</i>	if this rule is not the same as <i>other</i> .
<i>false</i>	if this rule is the same as <i>other</i> .

Compares the chain and priority (if set) associated with the rule as well.

Implements [Rule](#).

Definition at line 170 of file [Call.cpp](#).

References [Rule::condition_](#).

15.5.3.4 operator==()

```
bool Call::operator== (
    const Rule & other ) const [virtual]
```

Equality comparison.

Compares the type of the [Rule](#) and the condition ([If](#)) if set.

Parameters

<i>other</i>	The other rule to compare this to.
--------------	------------------------------------

Returns

<i>true</i>	if this rule is the same as <i>other</i> .
<i>false</i>	if this rule is not the same as <i>other</i> .

Compares the chain and priority (if set) associated with the rule as well.

Implements [Rule](#).

Definition at line 157 of file [Call.cpp](#).

References [Rule::condition_](#).

15.5.3.5 print_()

```
std::ostream & Call::print_ (
    std::ostream & os ) const [protected], [virtual]
```

Print the rule to the given output stream.

Parameters

<i>os</i>	The output stream to print to.
-----------	--------------------------------

Returns

The output stream.

Prints "call <Chain Name> <If Statement>" or "call <Chain Name> with priority <If Statement> with priority <priority>" if the priority is given.

Implements [Rule](#).

Definition at line 93 of file [Call.cpp](#).

References [Rule::condition_](#).

15.5.4 Member Data Documentation

15.5.4.1 condition_

```
std::optional<If> Rule::condition_ [protected], [inherited]
```

Definition at line 91 of file [Rule.hpp](#).

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

- [Call.hpp](#)
- [Call.cpp](#)

15.6 Chain Class Reference

```
#include <Chain.hpp>
```

Collaboration diagram for Chain:

Chain
+ Chain() + Chain() + Chain() + ~Chain() + action() + append() + name() + operator=() + operator=()

Public Member Functions

- `Chain (const Chain &other)`
- `Chain (Chain &&other)=default`
- `Chain (std::string name_, std::vector< std::unique_ptr< Rule >> &&rules={})`
- `TEST_VIRTUAL ~Chain ()=default`
- `TEST_VIRTUAL Action action (const Packet &packet, const MAVAddress &address)`
- `void append (std::unique_ptr< Rule > rule)`
- `const std::string & name () const`
- `Chain & operator= (const Chain &other)`
- `Chain & operator= (Chain &&other)=default`

Friends

- `bool operator== (const Chain &lhs, const Chain &rhs)`
- `bool operator!= (const Chain &lhs, const Chain &rhs)`
- `std::ostream & operator<< (std::ostream &os, const Chain &chain)`

15.6.1 Detailed Description

A filter chain, containing a list of rules to check packets against.

Definition at line 37 of file [Chain.hpp](#).

15.6.2 Constructor & Destructor Documentation

15.6.2.1 `Chain()` [1/3]

```
Chain::Chain (
    const Chain & other )
```

Copy constructor.

Parameters

<code>other</code>	<code>Chain</code> to copy from.
--------------------	----------------------------------

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

References `Rule::clone()`, and `rule`.

Here is the call graph for this function:



15.6.2.2 Chain() [2/3]

```
Chain::Chain (
    Chain && other ) [default]
```

Move constructor.

Parameters

<i>other</i>	Chain to move from.
--------------	---------------------

15.6.2.3 Chain() [3/3]

```
Chain::Chain (
    std::string name,
    std::vector< std::unique_ptr< Rule >> && rules = {} )
```

Construct a new filter chain.

Note

No rule in the chain may contain a [GoTo](#) or [Call](#) that would directly or indirectly result in returning to this chain.

Parameters

<i>name</i>	The name of the filter chain. This is only used when printing the chain. The name cannot contain whitespace.
<i>rules</i>	A vector of the rules used in the filter chain. This must be moved from since the vector is made up of std::unique_ptr's.

Exceptions

<code>std::invalid_argument</code>	if the name contains whitespace.
------------------------------------	----------------------------------

Definition at line 59 of file [Chain.cpp](#).

15.6.2.4 ~Chain()

```
TEST_VIRTUAL Chain::~Chain ( ) [default]
```

15.6.3 Member Function Documentation

15.6.3.1 action()

```
Action Chain::action (
    const Packet & packet,
    const MAVAddress & address )
```

Decide what to do with a [Packet](#).

Determine what action to take with the given packet sent to the given address. The possible actions are documented in the [Action](#) class.

The filter chain will loop through all the rules and the first one that matches and returns something other than the continue [Action](#) will be taken as the result.

Note

An error will be thrown if any [Call](#) or [GoTo](#) rule matches that directly or indirectly loops back to this chain.

Parameters

<code>packet</code>	The packet to determine whether to allow or not.
<code>address</code>	The address the <code>packet</code> will be sent out on if the action allows it.

Returns

The action to take with the packet. If this is the accept [Action](#) object, it may also contain a priority for the packet.

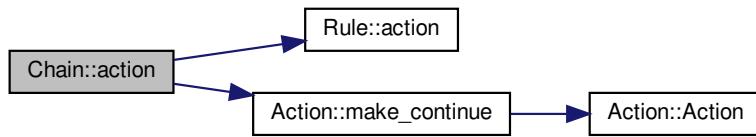
Exceptions

<i>RecursionError</i>	if a rule loops back to this chain.
-----------------------	-------------------------------------

Definition at line 90 of file [Chain.cpp](#).

References [Rule::action\(\)](#), [Action::CONTINUE](#), [Action::make_continue\(\)](#), and [rule](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.6.3.2 append()

```
void Chain::append (
    std::unique_ptr< Rule > rule )
```

Append a new rule to the filter chain.

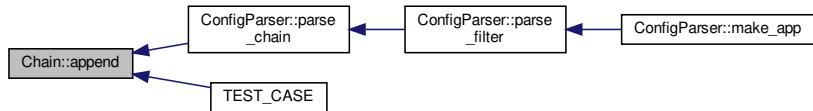
Parameters

<i>rule</i>	A new filter rule to append to the chain.
-------------	---

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

References rule.

Here is the caller graph for this function:



15.6.3.3 name()

```
const std::string & Chain::name( ) const
```

Return the name of the chain.

Note

This is only used when printing the chain.

Returns

The chain's name.

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

15.6.3.4 operator=() [1/2]

```
Chain & Chain::operator= (
    const Chain & other )
```

Assignment operator.

Parameters

<i>other</i>	Chain to copy from.
--------------	---------------------

Definition at line 138 of file [Chain.cpp](#).

References [Rule::clone\(\)](#), and [rule](#).

Here is the call graph for this function:



15.6.3.5 operator=() [2/2]

```
Chain& Chain::operator= (
    Chain && other ) [default]
```

Assignment operator (by move semantics).

Parameters

<i>other</i>	Chain to move from.
--------------	-------------------------------------

15.6.4 Friends And Related Function Documentation

15.6.4.1 operator!=

```
bool operator!= (
    const Chain & lhs,
    const Chain & rhs ) [friend]
```

Inequality comparison.

Compares the chain name and each [Rule](#) in the chain.

Parameters

<i>lhs</i>	The left hand side action.
<i>rhs</i>	The right hand side action.

Return values

<i>true</i>	if <i>lhs</i> is not the same as <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is the same as <i>rhs</i> .

Definition at line 200 of file [Chain.cpp](#).

15.6.4.2 operator<<

```
std::ostream & operator<< (
    std::ostream & os,
    const Chain & chain ) [friend]
```

Print the given filter chain to to the given output stream.

An example is:

```
chain default {
    reject if HEARTBEAT from 10.10;
    accept with priority -3 if GPS_STATUS to 172.0/8;
    accept if GLOBAL_POSITION_INT to 172.0/8;
    goto ap-in with priority 3 if from 192.168;
    call ap-out if to 192.168;
    reject;
}
```

Parameters

<i>os</i>	The output stream to print to.
<i>chain</i>	The filter chain to print.

Returns

The output stream.

Definition at line 225 of file [Chain.cpp](#).

15.6.4.3 operator==

```
bool operator== (
    const Chain & lhs,
    const Chain & rhs ) [friend]
```

Equality comparison.

Compares the chain name and each [Rule](#) in the chain.

Parameters

<i>lhs</i>	The left hand side filter chain.
<i>rhs</i>	The right hand side filter chain.

Return values

<i>true</i>	if <i>lhs</i> is the same as <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is not the same as <i>rhs</i> .

Definition at line 163 of file [Chain.cpp](#).

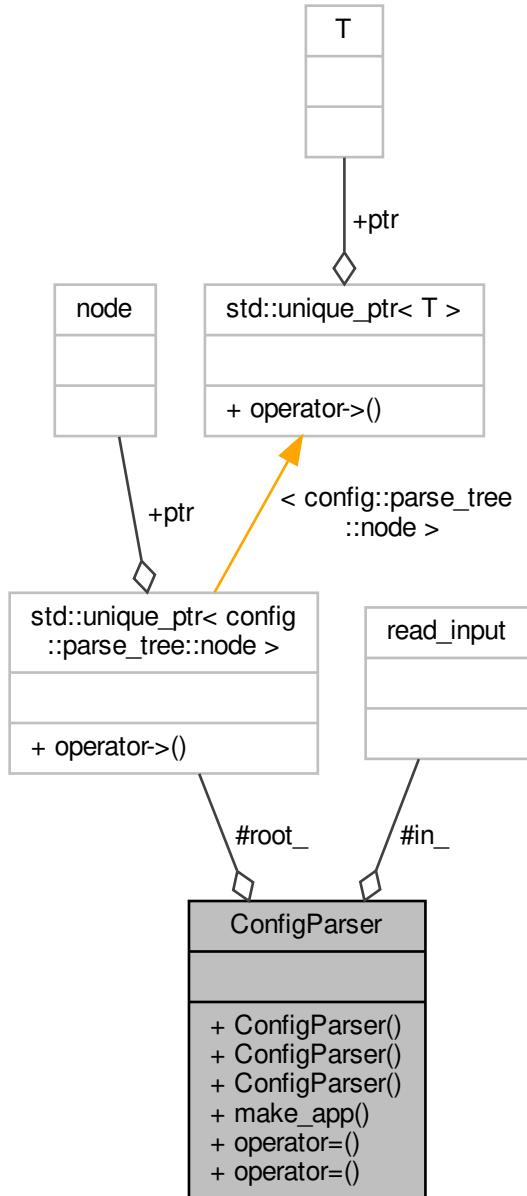
The documentation for this class was generated from the following files:

- [Chain.hpp](#)
- [Chain.cpp](#)

15.7 ConfigParser Class Reference

```
#include <ConfigParser.hpp>
```

Collaboration diagram for ConfigParser:



Public Member Functions

- `ConfigParser (std::string filename)`
- `ConfigParser (const ConfigParser &other)=delete`
- `ConfigParser (ConfigParser &&other)=delete`

- std::unique_ptr< App > make_app ()
- ConfigParser & operator= (const ConfigParser &other)=delete
- ConfigParser & operator= (ConfigParser &&other)=delete

Protected Attributes

- tao::pegtl::read_input *in_*
- std::unique_ptr< config::parse_tree::node > *root_*

Friends

- std::ostream & operator<< (std::ostream &os, const ConfigParser &config_parser)

Related Functions

(Note that these are not member functions.)

- std::map< std::string, std::shared_ptr< Chain > > init_chains (const config::parse_tree::node &root)
- std::unique_ptr< Rule > parse_action (const config::parse_tree::node &root, std::optional< int > priority, std::optional< If > condition, const std::map< std::string, std::shared_ptr< Chain > > &chains)
- void parse_chain (Chain &chain, const config::parse_tree::node &root, const std::map< std::string, std::shared_ptr< Chain > > &chains)
- If parse_condition (const config::parse_tree::node &root)
- std::unique_ptr< Filter > parse_filter (const config::parse_tree::node &root)
- std::vector< std::unique_ptr< Interface > > parse_interfaces (const config::parse_tree::node &root, std::unique_ptr< Filter > filter)
- std::unique_ptr< SerialInterface > parse_serial (const config::parse_tree::node &root, std::shared_ptr< Filter > filter, std::shared_ptr< ConnectionPool > pool)
- std::unique_ptr< UDPInterface > parse_udp (const config::parse_tree::node &root, std::shared_ptr< Filter > filter, std::shared_ptr< ConnectionPool > pool)

15.7.1 Detailed Description

Configuration file parser.

Used to parse a configuration file and create an instance of the mavtables application.

Definition at line 74 of file [ConfigParser.hpp](#).

15.7.2 Constructor & Destructor Documentation

15.7.2.1 ConfigParser() [1/3]

```
ConfigParser::ConfigParser (
    std::string filename )
```

Construct a configuration parser from a file.

Parameters

<i>filename</i>	The path of the configuration file to parse.
-----------------	--

Exceptions

<i>std::runtime_error</i>	if the configuration file cannot be parsed.
---------------------------	---

Definition at line 443 of file [ConfigParser.cpp](#).

References [in_](#), [config::parse\(\)](#), and [root_](#).

Here is the call graph for this function:



15.7.2.2 ConfigParser() [2/3]

```
ConfigParser::ConfigParser (
    const ConfigParser & other ) [delete]
```

15.7.2.3 ConfigParser() [3/3]

```
ConfigParser::ConfigParser (
    ConfigParser && other ) [delete]
```

15.7.3 Member Function Documentation

15.7.3.1 make_app()

```
std::unique_ptr< App > ConfigParser::make_app ( )
```

Build a mavtables application from the AST contained by the parser.

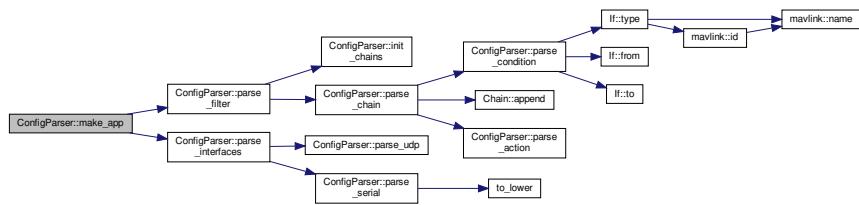
Returns

A mavtables application.

Definition at line 464 of file [ConfigParser.cpp](#).

References [parse_filter\(\)](#), [parse_interfaces\(\)](#), and [root_](#).

Here is the call graph for this function:



15.7.3.2 operator=() [1/2]

```
ConfigParser& ConfigParser::operator= (
    const ConfigParser & other ) [delete]
```

15.7.3.3 operator=() [2/2]

```
ConfigParser& ConfigParser::operator= (
    ConfigParser && other ) [delete]
```

15.7.4 Friends And Related Function Documentation

15.7.4.1 init_chains()

```
std::map< std::string, std::shared_ptr< Chain > > init_chains (
    const config::parse_tree::node & root ) [related]
```

Construct a map of non default chains.

Parameters

<code>root</code>	Root of configuration AST.
-------------------	----------------------------

Returns

Map of chain names to chains.

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

Here is the caller graph for this function:

**15.7.4.2 operator<<**

```
std::ostream & operator<< (
    std::ostream & os,
    const ConfigParser & config_parser ) [friend]
```

Print the configuration settings to the given output stream.

An example (that of test/mavtables.conf) is:

```
===== test/mavtables.conf =====
:001: default_action
:001: | accept
:004: udp
:005: | port 14500
:006: | address 127.0.0.1
:007: | max_bitrate 8388608
:011: serial
:012: | device ./ttyS0
:013: | baudrate 115200
:014: | flow_control yes
:015: | preload 1.1
:016: | preload 62.34
:020: chain default
:022: | call some_chain10
:022: | | condition
:022: | | | source 127.1
:022: | | | dest 192.0
:023: | reject
:027: chain some_chain10
:029: | accept
:029: | | priority 99
:029: | | condition
:029: | | | dest 192.0
:030: | accept
:030: | | condition
:030: | | | packet_type PING
:031: | accept
```

Parameters

<i>os</i>	The output stream to print to.
<i>config_parser</i>	The configuration parser to print.

Returns

The output stream.

Definition at line 512 of file [ConfigParser.cpp](#).

15.7.4.3 parse_action()

```
std::unique_ptr< Rule > parse_action (
    const config::parse_tree::node & root,
    std::optional< int > priority,
    std::optional< If > condition,
    const std::map< std::string, std::shared_ptr< Chain >> & chains ) [related]
```

Construct a [Rule](#) with action from AST, priority, and condition.

Parameters

<i>root</i>	AST action node.
<i>priority</i>	The priority to use when constructing the action. No priority if {}. If the AST node is a reject action the priority will be ignored.
<i>condition</i>	The condition to use when constructing the action.
<i>chains</i>	Map of chain names to chains for call and goto actions.

Returns

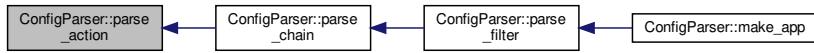
The action (or rule) parsed from the given AST node, priority, and condition.

Exceptions

<i>std::invalid_argument</i>	if the action attempts to call or goto the default chain.
<i>std::runtime_error</i>	if the action is not one of accept, reject, call, or goto.

Definition at line 92 of file [ConfigParser.cpp](#).

Here is the caller graph for this function:



15.7.4.4 parse_chain()

```

void parse_chain (
    Chain & chain,
    const config::parse_tree::node & root,
    const std::map< std::string, std::shared_ptr< Chain >> & chains ) [related]
  
```

Add Rule's from AST to a Chain.

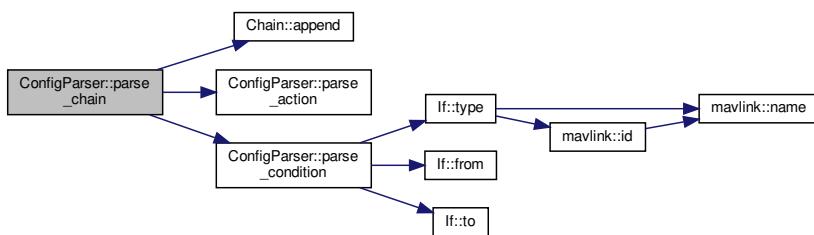
Parameters

<i>chain</i>	Chain to add rules to.
<i>root</i>	AST chain node containing rules.
<i>chains</i>	Map of chain names to chains for call and goto actions.

Definition at line 176 of file [ConfigParser.cpp](#).

References [Chain::append\(\)](#), [parse_action\(\)](#), and [parse_condition\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.7.4.5 parse_condition()

```
If parse_condition (
    const config::parse_tree::node & root ) [related]
```

Construct conditional ([If](#)) from AST.

Parameters

<i>root</i>	AST conditional node.
-------------	-----------------------

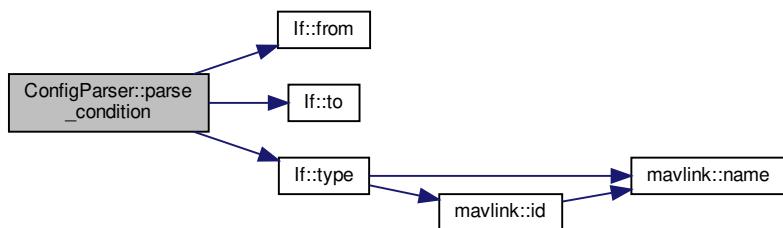
Returns

The conditional constructed from the given AST node.

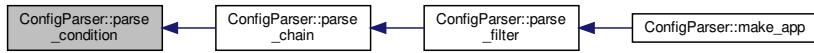
Definition at line [216](#) of file [ConfigParser.cpp](#).

References [If::from\(\)](#), [If::to\(\)](#), and [If::type\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.7.4.6 parse_filter()

```
std::unique_ptr< Filter > parse_filter (
    const config::parse_tree::node & root ) [related]
```

Parse [Filter](#) from AST.

Parameters

<i>root</i>	Root of configuration AST.
-------------	----------------------------

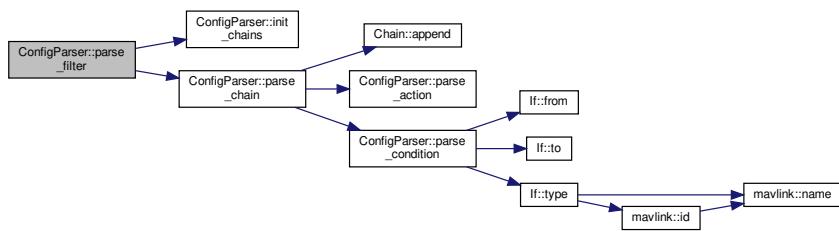
Returns

The [Filter](#) parsed from the AST.

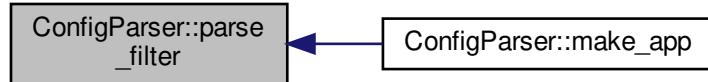
Definition at line 250 of file [ConfigParser.cpp](#).

References [init_chains\(\)](#), and [parse_chain\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.7.4.7 parse_interfaces()

```
std::vector< std::unique_ptr< Interface > > parse_interfaces (
    const config::parse_tree::node & root,
    std::unique_ptr< Filter > filter ) [related]
```

Parse UDP and serial port interfaces from AST root.

Parameters

<i>root</i>	The root of the AST to create Interface 's from.
<i>filter</i>	The packet Filter to use for the interfaces.

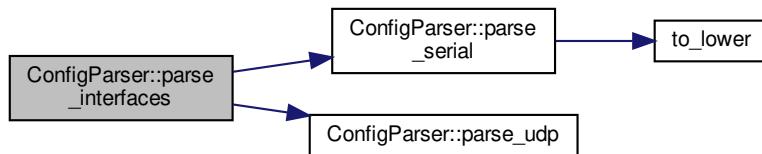
Returns

A vector of UDP and serial port interfaces.

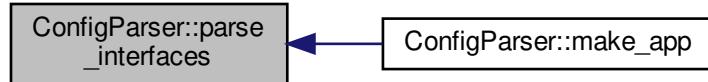
Definition at line 292 of file [ConfigParser.cpp](#).

References [parse_serial\(\)](#), and [parse_udp\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.7.4.8 parse_serial()

```
std::unique_ptr< SerialInterface > parse_serial (
    const config::parse_tree::node & root,
    std::shared_ptr< Filter > filter,
    std::shared_ptr< ConnectionPool > pool ) [related]
```

Parse a serial port interface from an AST.

Parameters

<i>root</i>	The serial port node to parse.
<i>filter</i>	The Filter to use for the SerialInterface .
<i>pool</i>	The connection pool to add the interface's connection to.

Returns

The serial port interface parsed from the AST and using the given filter and connection pool.

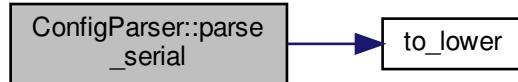
Exceptions

<i>std::invalid_argument</i>	if the device string is missing.
------------------------------	----------------------------------

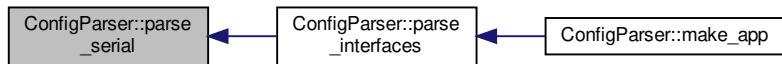
Definition at line 330 of file [ConfigParser.cpp](#).

References [SerialPort::DEFAULT](#), [SerialPort::FLOW_CONTROL](#), and [to_lower\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.7.4.9 parse_udp()

```
std::unique_ptr< UDPInterface > parse_udp (
    const config::parse_tree::node & root,
    std::shared_ptr< Filter > filter,
    std::shared_ptr< ConnectionPool > pool ) [related]
```

Parse a UPD interface from an AST.

Parameters

<i>root</i>	The UDP node to parse.
<i>filter</i>	The Filter to use for the UDPInterface .
<i>pool</i>	The connection pool to add the interface's connections to.

Returns

The UDP interface parsed from the AST and using the given filter and connection pool.

Definition at line 400 of file [ConfigParser.cpp](#).

Here is the caller graph for this function:



15.7.5 Member Data Documentation

15.7.5.1 in_

```
tao::pegtl::read_input ConfigParser::in_ [protected]
```

Definition at line 88 of file [ConfigParser.hpp](#).

15.7.5.2 root_

```
std::unique_ptr<config::parse_tree::node> ConfigParser::root_ [protected]
```

Definition at line 89 of file [ConfigParser.hpp](#).

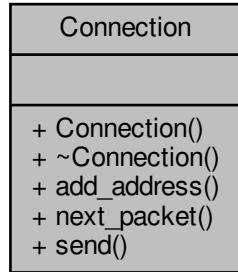
The documentation for this class was generated from the following files:

- [ConfigParser.hpp](#)
- [ConfigParser.cpp](#)

15.8 Connection Class Reference

```
#include <Connection.hpp>
```

Collaboration diagram for Connection:



Public Member Functions

- `Connection (std::string name, std::shared_ptr< Filter > filter, bool mirror=false, std::unique_ptr< AddressPool<>> pool=std::make_unique< AddressPool>(), std::unique_ptr< PacketQueue > queue=std::make_unique< PacketQueue >())`
- TEST_VIRTUAL `~Connection ()=default`
- TEST_VIRTUAL `void add_address (MAVAddress address)`
- TEST_VIRTUAL `std::shared_ptr< const Packet > next_packet (const std::chrono::nanoseconds &timeout=std::chrono::nanoseconds(0))`
- TEST_VIRTUAL `void send (std::shared_ptr< const Packet > packet)`

Friends

- `std::ostream & operator<< (std::ostream &os, const Connection &connection)`

15.8.1 Detailed Description

Represents a connection that packets can be sent over.

The connection class does not actually send anything. It filters and sorts packets in a queue for sending by an [Interface](#). It also maintains a list of MAVLink addresses reachable on this connection.

Definition at line 39 of file [Connection.hpp](#).

15.8.2 Constructor & Destructor Documentation

15.8.2.1 Connection()

```
Connection::Connection (
    std::string name,
    std::shared_ptr< Filter > filter,
    bool mirror = false,
    std::unique_ptr< AddressPool >> pool = std::make_unique<AddressPool>(),
    std::unique_ptr< PacketQueue > queue = std::make_unique<PacketQueue>() )
```

Construct a connection.

Parameters

<i>name</i>	The name of the connection, should be the device string for a serial connection or the IP address and port number for a UDP connection.
<i>filter</i>	The packet filter to use for determining whether and with what priority to add a packet to the queue for transmission.
<i>mirror</i>	Set to true if this is to be a mirror connection. A mirror connection is one that will receive all packets, regardless of destination address. The default is false.
<i>pool</i>	The AddressPool to use for keeping track of the addresses reachable by the connection. A default address pool will be used if none is given.
<i>queue</i>	The PacketQueue to use to hold packets awaiting transmission. A default packet queue will be used if none is given.

Exceptions

<code>std::invalid_argument</code>	if the given any of the <code>filter</code> , <code>pool</code> , or <code>queue</code> pointers are null.
------------------------------------	--

Remarks

If the given [AddressPool](#) and [PacketQueue](#) are threadsafe then the connection will also be threadsafe.

Definition at line 244 of file [Connection.cpp](#).

15.8.2.2 ~Connection()

```
TEST_VIRTUAL Connection::~Connection ( ) [default]
```

15.8.3 Member Function Documentation

15.8.3.1 add_address()

```
void Connection::add_address (
    MAVAddress address )
```

Add a MAVLink address to the connection.

This adds an address to the list of systems/components that can be reached on this connection.

Note

Addresses will be removed after the timeout set in the [AddressPool](#) given in the constructor. Re-adding the address (even before this time runs out) will reset the timeout.

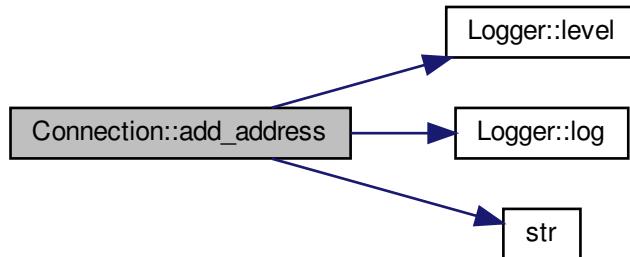
Parameters

<code>address</code>	The MAVLink address to add, or update the timeout for.
----------------------	--

Definition at line [281](#) of file [Connection.cpp](#).

References [Logger::level\(\)](#), [Logger::log\(\)](#), and [str\(\)](#).

Here is the call graph for this function:



15.8.3.2 next_packet()

```
std::shared_ptr< const Packet > Connection::next_packet (
    const std::chrono::nanoseconds & timeout = std::chrono::nanoseconds(0) )
```

Get next packet to send.

Blocks until a packet is ready to be sent or the `timeout` expires. Returns `nullptr` in the later case.

Parameters

<i>timeout</i>	How long to block waiting for a packet. Set to 0s for non blocking.
----------------	---

Returns

The next packet to send. Or nullptr if the call times out waiting on a packet.

Definition at line 303 of file [Connection.cpp](#).

15.8.3.3 send()

```
void Connection::send (
    std::shared_ptr< const Packet > packet )
```

Send a packet out on the connection.

Packets are ran through the contained [Filter](#) before being placed into the [PacketQueue](#) given in the constructor. Packets are read from the queue (for sending) by using the [next_packet](#) method.

Note

If the packet has a destination address that is not 0.0 (the broadcast address) it will only be sent if that system is reachable on this connection. It will still be sent even if the particular component cannot be found.

If this is a mirror connection then the destination address of the packet is ignored.

Parameters

<i>packet</i>	The packet to send.
---------------	---------------------

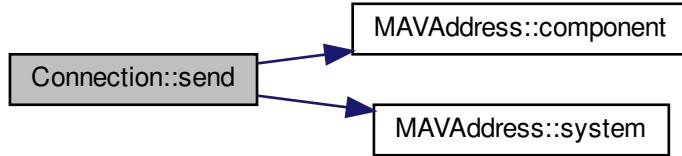
Exceptions

<i>std::invalid_argument</i>	if the <i>packet</i> pointer is null.
------------------------------	---------------------------------------

Definition at line 327 of file [Connection.cpp](#).

References [MAVAddress::component\(\)](#), and [MAVAddress::system\(\)](#).

Here is the call graph for this function:



15.8.4 Friends And Related Function Documentation

15.8.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Connection & connection ) [friend]
```

Print the connection name to the given output stream.

Some examples are:

- `/dev/ttyUSB0`
- `127.0.0.1:8000`
- `127.0.0.1:14550`

Definition at line 368 of file [Connection.cpp](#).

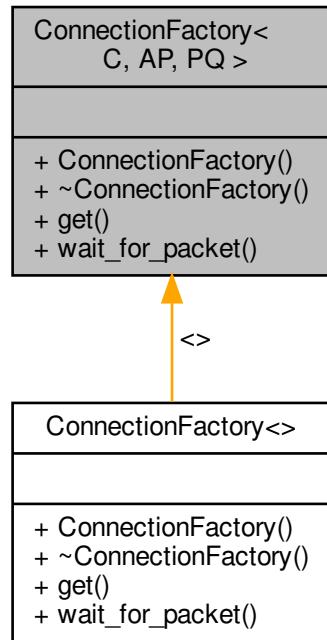
The documentation for this class was generated from the following files:

- [Connection.hpp](#)
- [Connection.cpp](#)

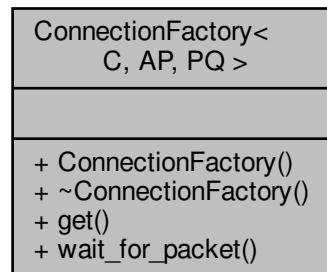
15.9 ConnectionFactory< C, AP, PQ > Class Template Reference

```
#include <ConnectionFactory.hpp>
```

Inheritance diagram for ConnectionFactory< C, AP, PQ >:



Collaboration diagram for ConnectionFactory< C, AP, PQ >:



Public Member Functions

- `ConnectionFactory` (`std::shared_ptr< Filter > filter, bool mirror=false)`
- `TEST_VIRTUAL ~ConnectionFactory ()=default`
- `TEST_VIRTUAL std::unique_ptr< C > get (std::string name="unknown")`
- `TEST_VIRTUAL bool wait_for_packet (const std::chrono::nanoseconds &timeout)`

15.9.1 Detailed Description

```
template<class C = Connection, class AP = AddressPool<>, class PQ = PacketQueue>
class ConnectionFactory< C, AP, PQ >
```

A factory for making related connections that use a common semaphore.

Definition at line 36 of file [ConnectionFactory.hpp](#).

15.9.2 Constructor & Destructor Documentation

15.9.2.1 ConnectionFactory()

```
template<class C , class AP , class PQ >
ConnectionFactory< C, AP, PQ >::ConnectionFactory (
    std::shared_ptr< Filter > filter,
    bool mirror = false )
```

Construct a connection factory.

Template Parameters

<i>C</i>	The Connection class (or derived class) to use.
<i>AP</i>	The AddressPool class (or derived class) to use.
<i>PQ</i>	The PacketQueue class (or derived class) to use, must accept a callback function in it's constructor.

Parameters

<i>filter</i>	The packet filter to use for determining whether and with what priority to add a packet to the queue for transmission. This will be given to each constructed Connection . Cannot be nullptr.
<i>mirror</i>	Set to true if all Connection 's made by this factory are to be mirror connections. A mirror connection is one that will receive all packets, regardless of destination address. The default is false.

Exceptions

<code>std::invalid_argument</code>	if the given <code>filter</code> pointer is null.
------------------------------------	---

Definition at line 67 of file [ConnectionFactory.hpp](#).

15.9.2.2 ~ConnectionFactory()

```
template<class C = Connection, class AP = AddressPool<>, class PQ = PacketQueue>
TEST_VIRTUAL ConnectionFactory< C, AP, PQ >::~ConnectionFactory() [default]
```

15.9.3 Member Function Documentation

15.9.3.1 get()

```
template<class C , class AP , class PQ >
std::unique_ptr< C > ConnectionFactory< C, AP, PQ >::get (
    std::string name = "unknown" )
```

Construct and return a new connection.

This connection will share a common semaphore with all other connections made by this factory instance.

Parameters

<code>name</code>	The name of the new connection.
-------------------	---------------------------------

Returns

The new connection.

Definition at line 87 of file [ConnectionFactory.hpp](#).

15.9.3.2 wait_for_packet()

```
template<class C , class AP , class PQ >
bool ConnectionFactory< C, AP, PQ >::wait_for_packet (
    const std::chrono::nanoseconds & timeout )
```

Wait for a packet to be available on any connection made by this factory.

Parameters

<i>timeout</i>	How long to block waiting for a packet. Set to 0s for non blocking.
----------------	---

Return values

<i>true</i>	There is a packet on at least one of the connections created by this factory instance.
<i>false</i>	The wait timed out, there is no packet available on any of the connections created by this factory instance.

Definition at line 109 of file [ConnectionFactory.hpp](#).

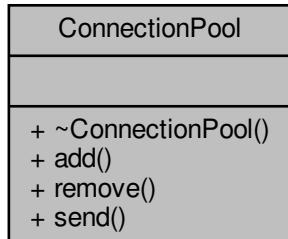
The documentation for this class was generated from the following file:

- [ConnectionFactory.hpp](#)

15.10 ConnectionPool Class Reference

```
#include <ConnectionPool.hpp>
```

Collaboration diagram for ConnectionPool:



Public Member Functions

- TEST_VIRTUAL [~ConnectionPool](#) ()=default
- TEST_VIRTUAL void [add](#) (std::weak_ptr< [Connection](#) > connection)
- TEST_VIRTUAL void [remove](#) (const std::weak_ptr< [Connection](#) > &connection)
- TEST_VIRTUAL void [send](#) (std::unique_ptr< const [Packet](#) > packet)

15.10.1 Detailed Description

A pool of [Connection](#)'s to send packets out on.

A connection pool stores a reference to all connections that packets can be sent out over.

Definition at line [36](#) of file [ConnectionPool.hpp](#).

15.10.2 Constructor & Destructor Documentation

15.10.2.1 `~ConnectionPool()`

```
TEST_VIRTUAL ConnectionPool::~ConnectionPool ( ) [default]
```

15.10.3 Member Function Documentation

15.10.3.1 `add()`

```
void ConnectionPool::add ( std::weak_ptr< Connection > connection )
```

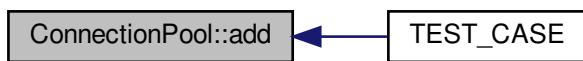
Add a connection to the pool.

Parameters

<code>connection</code>	The connection to add to the pool.
-------------------------	------------------------------------

Definition at line [35](#) of file [ConnectionPool.cpp](#).

Here is the caller graph for this function:



15.10.3.2 remove()

```
void ConnectionPool::remove (
    const std::weak_ptr< Connection > & connection )
```

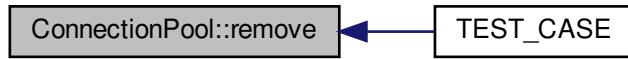
Remove a connection from the pool.

Parameters

<i>connection</i>	The connection to remove from the pool.
-------------------	---

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

Here is the caller graph for this function:



15.10.3.3 send()

```
void ConnectionPool::send (
    std::unique_ptr< const Packet > packet )
```

Send a packet to every connection.

Note

Each connection may decide to ignore the packet based on it's filter rules.

Parameters

<i>packet</i>	The packet to send to every connection, must not be nullptr.
---------------	--

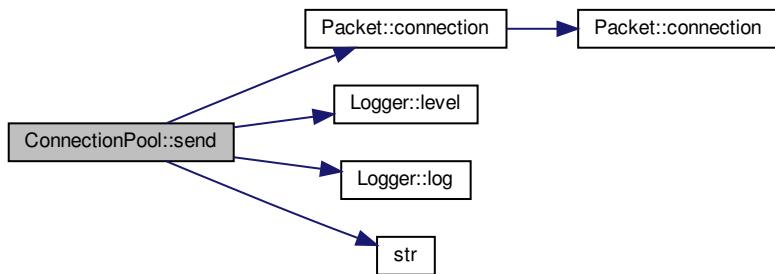
Exceptions

`std::invalid_argument` if the `packet` pointer is null.

Definition at line 61 of file [ConnectionPool.cpp](#).

References [Packet::connection\(\)](#), [Logger::level\(\)](#), [Logger::log\(\)](#), and [str\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



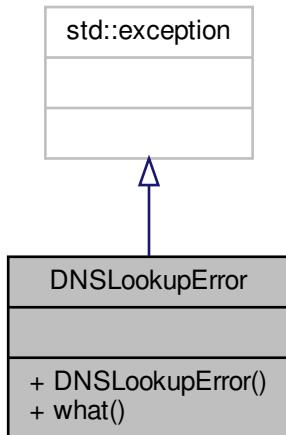
The documentation for this class was generated from the following files:

- [ConnectionPool.hpp](#)
- [ConnectionPool.cpp](#)

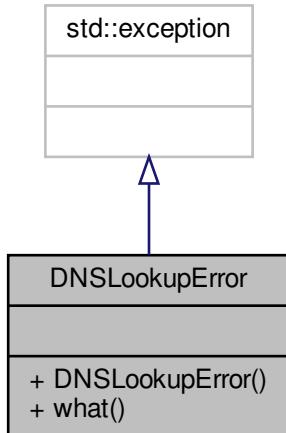
15.11 DNSLookupError Class Reference

```
#include <DNSLookupError.hpp>
```

Inheritance diagram for DNSLookupError:



Collaboration diagram for DNSLookupError:



Public Member Functions

- [DNSLookupError \(std::string url\)](#)
- [const char * what \(\) const noexcept](#)

15.11.1 Detailed Description

Exception type emitted when a DNS lookup fails.

Definition at line [28](#) of file [DNSLookupError.hpp](#).

15.11.2 Constructor & Destructor Documentation

15.11.2.1 DNSLookupError()

```
DNSLookupError::DNSLookupError (
    std::string url )
```

Construct a [DNSLookupError](#) given the unresolvable URL.

Parameters

<i>url</i>	The URL that could not be resolved.
------------	-------------------------------------

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

15.11.3 Member Function Documentation

15.11.3.1 what()

```
const char * DNSLookupError::what ( ) const [noexcept]
```

Return error message string.

Returns

Error string containing unresolvable hostname.

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

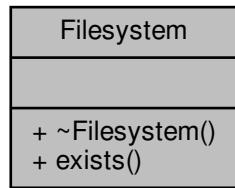
The documentation for this class was generated from the following files:

- [DNSLookupError.hpp](#)
- [DNSLookupError.cpp](#)

15.12 Filesystem Class Reference

```
#include <Filesystem.hpp>
```

Collaboration diagram for Filesystem:



Public Types

- using `path` = boost::filesystem::path

Public Member Functions

- TEST_VIRTUAL `~Filesystem ()`
- TEST_VIRTUAL bool `exists (const path &p) const`

15.12.1 Detailed Description

A class containing filesystem operations.

Note

This class exists for testing purposes and to provide a level of indirection so the underlying filesystem library can be changed to std::filesystem once it is moved out of experimental.

Definition at line 37 of file [Filesystem.hpp](#).

15.12.2 Member Typedef Documentation

15.12.2.1 path

```
using Filesystem::path = boost::filesystem::path
```

Definition at line [40](#) of file [Filesystem.hpp](#).

15.12.3 Constructor & Destructor Documentation

15.12.3.1 ~Filesystem()

```
Filesystem::~Filesystem ( )
```

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

15.12.4 Member Function Documentation

15.12.4.1 exists()

```
bool Filesystem::exists ( const path & p ) const
```

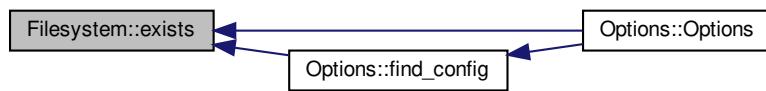
Returns true if the given path exists.

Parameters

<i>p</i>	The path to check for existence.
----------	----------------------------------

Definition at line [36](#) of file [Filesystem.cpp](#).

Here is the caller graph for this function:



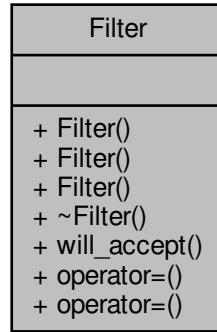
The documentation for this class was generated from the following files:

- [Filesystem.hpp](#)
- [Filesystem.cpp](#)

15.13 Filter Class Reference

```
#include <Filter.hpp>
```

Collaboration diagram for Filter:



Public Member Functions

- `Filter (const Filter &other)=default`
- `Filter (Filter &&other)=default`
- `Filter (Chain default_chain, bool accept_by_default=false)`
- TEST_VIRTUAL `~Filter ()=default`
- TEST_VIRTUAL `std::pair< bool, int > will_accept (const Packet &packet, const MAVAddress &address)`
- `Filter & operator= (const Filter &other)=default`
- `Filter & operator= (Filter &&other)=default`

Friends

- `bool operator== (const Filter &lhs, const Filter &rhs)`
- `bool operator!= (const Filter &lhs, const Filter &rhs)`

15.13.1 Detailed Description

The filter used to determine whether to accept or reject a packet.

See also

[Chain](#)
[Rule](#)
[If](#)

Definition at line 38 of file [Filter.hpp](#).

15.13.2 Constructor & Destructor Documentation

15.13.2.1 Filter() [1/3]

```
Filter::Filter (
    const Filter & other ) [default]
```

Copy constructor.

Parameters

<i>other</i>	Filter to copy from.
--------------	--------------------------------------

15.13.2.2 Filter() [2/3]

```
Filter::Filter (
    Filter && other ) [default]
```

Move constructor.

Parameters

<i>other</i>	Filter to move from.
--------------	--------------------------------------

15.13.2.3 Filter() [3/3]

```
Filter::Filter (
    Chain default_chain,
    bool accept_by_default = false )
```

Construct a new packet filter.

Parameters

<i>default_chain</i>	The Chain that all filtering begins with.
<i>accept_by_default</i>	Whether to accept (true) or reject (false) packets that don't match any rules in the default chain or any chains called by the default chain. The default value is false and thus to reject unmatched packets.

Definition at line 36 of file [Filter.cpp](#).

15.13.2.4 ~Filter()

```
TEST_VIRTUAL Filter::~Filter ( ) [default]
```

15.13.3 Member Function Documentation

15.13.3.1 operator=() [1/2]

```
Filter& Filter::operator= (
    const Filter & other ) [default]
```

Assignment operator.

Parameters

<i>other</i>	Filter to copy from.
--------------	--------------------------------------

15.13.3.2 operator=() [2/2]

```
Filter& Filter::operator= (
    Filter && other ) [default]
```

Assignment operator (by move semantics).

Parameters

<i>other</i>	Filter to move from.
--------------	--------------------------------------

15.13.3.3 will_accept()

```
std::pair< bool, int > Filter::will_accept (
    const Packet & packet,
    const MAVAddress & address )
```

Determine whether to accept or reject a packet/address combination.

Parameters

<i>packet</i>	The packet to determine whether to allow or not.
<i>address</i>	The address the packet will be sent out on if the action allows it.

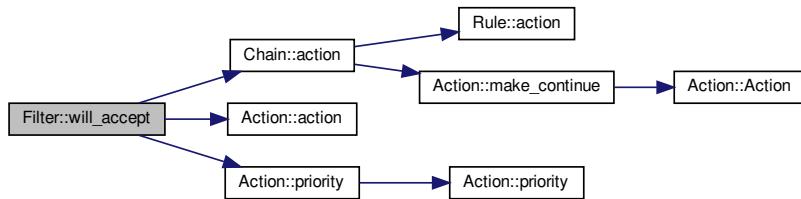
Returns

A pair (tuple) with the first value being whether to accept the packet or not and the second being the priority to use when sending the packet. The second value is only defined if the first value is true (accept).

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

References [Action::ACCEPT](#), [Chain::action\(\)](#), [Action::action\(\)](#), [Action::CONTINUE](#), [Action::DEFAULT](#), [Action::priority\(\)](#), and [Action::REJECT](#).

Here is the call graph for this function:



15.13.4 Friends And Related Function Documentation

15.13.4.1 operator!=

```
bool operator!= (
    const Filter & lhs,
    const Filter & rhs ) [friend]
```

Inequality comparison.

The default chain and default action are compared.

Parameters

<i>lhs</i>	The left hand side packet filter.
<i>rhs</i>	The right hand side packet filter.

Return values

<i>true</i>	if <i>lhs</i> is not the same as <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is the same as <i>rhs</i> .

Definition at line 102 of file [Filter.cpp](#).

15.13.4.2 operator==

```
bool operator== (
    const Filter & lhs,
    const Filter & rhs ) [friend]
```

Equality comparison.

The default chain and default action are compared.

Parameters

<i>lhs</i>	The left hand side packet filter.
<i>rhs</i>	The right hand side packet filter.

Return values

<i>true</i>	if <i>lhs</i> is the same as <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is not the same as <i>rhs</i> .

Definition at line 85 of file [Filter.cpp](#).

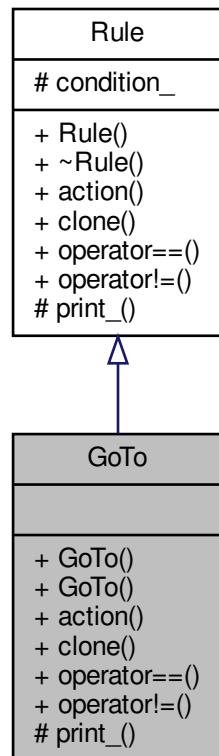
The documentation for this class was generated from the following files:

- [Filter.hpp](#)
- [Filter.cpp](#)

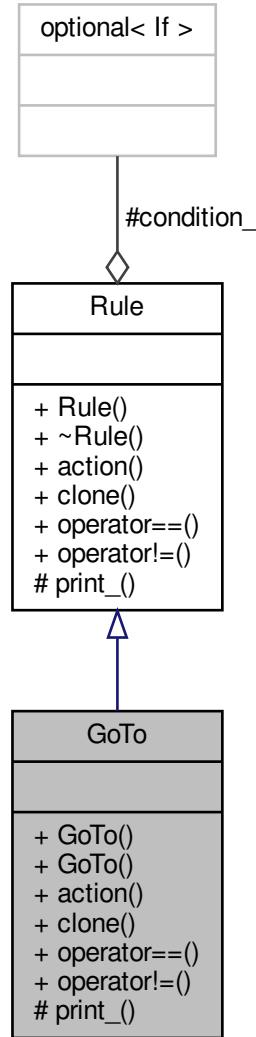
15.14 GoTo Class Reference

```
#include <GoTo.hpp>
```

Inheritance diagram for GoTo:



Collaboration diagram for GoTo:



Public Member Functions

- `GoTo (std::shared_ptr< Chain > chain, std::optional< If > condition={})`
- `GoTo (std::shared_ptr< Chain > chain, int priority, std::optional< If > condition={})`
- `virtual Action action (const Packet &packet, const MAVAddress &address) const`
- `virtual std::unique_ptr< Rule > clone () const`
- `virtual bool operator== (const Rule &other) const`
- `virtual bool operator!= (const Rule &other) const`

Protected Member Functions

- virtual std::ostream & `print_` (std::ostream &os) const

Protected Attributes

- std::optional< `If` > `condition_`

15.14.1 Detailed Description

Delegate decision on a packet to another [Chain](#).

[Rule](#) to delegate the decision on what to do with a packet to a filter [Chain](#). In particular, final decision is given to this [Chain](#). If the [Chain](#) cannot decide what to do with the [Packet](#) the global default action should be taken.

Definition at line 44 of file [GoTo.hpp](#).

15.14.2 Constructor & Destructor Documentation

15.14.2.1 GoTo() [1/2]

```
GoTo::GoTo (
    std::shared_ptr< Chain > chain,
    std::optional< If > condition = {} )
```

Construct a goto rule given a chain to delegate to, without a priority.

A goto rule is used to delegate the decision on whether to accept or reject a packet/address combination to another filter [Chain](#). [If](#) this called chain does not make a decision then the default action should be taken.

Parameters

<code>chain</code>	The chain to delegate decisions of whether to accept or reject a packet/address combination to. <code>nullptr</code> is not valid.
<code>condition</code>	The condition used to determine the rule matches a particular packet/address combination given to the action method. The default is <code>{}</code> which indicates the rule matches any packet/address combination.

Exceptions

<code>std::invalid_argument</code>	if the given pointer is null.
------------------------------------	-------------------------------

See also

[action](#)

Definition at line 47 of file [GoTo.cpp](#).

15.14.2.2 GoTo() [2/2]

```
GoTo::GoTo (
    std::shared_ptr< Chain > chain,
    int priority,
    std::optional< If > condition = {} )
```

Construct a goto rule given a chain to delegate to, with a priority.

A goto rule is used to delegate the decision on whether to accept or reject a packet/address combination to another filter [Chain](#). If this called chain does not make a decision then the default action should be taken.

Parameters

<i>chain</i>	The chain to delegate decisions of whether to accept or reject a packet/address combination to. nullptr is not valid.
<i>priority</i>	The priority to accept packets with. A higher number is more important and will be routed first.
<i>condition</i>	The condition used to determine the rule matches a particular packet/address combination given to the action method. The default is {} which indicates the rule matches any packet/address combination.

Exceptions

<code>std::invalid_argument</code>	if the given pointer is null.
------------------------------------	-------------------------------

See also

[action](#)

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

15.14.3 Member Function Documentation

15.14.3.1 action()

```
Action GoTo::action (
    const Packet & packet,
    const MAVAddress & address ) const [virtual]
```

Decide what to do with a [Packet](#).

Determine what action to take with the given packet sent to the given address. The possible actions are documented in the [Action](#) class. The continue object is always returned if the condition was set and does not match the packet/address combination.

Parameters

<i>packet</i>	The packet to determine whether to allow or not.
<i>address</i>	The address the <i>packet</i> will be sent out on if the action dictates it.

Returns

The action to take with the packet. If this is the accept object, it may also contain a priority for the packet.

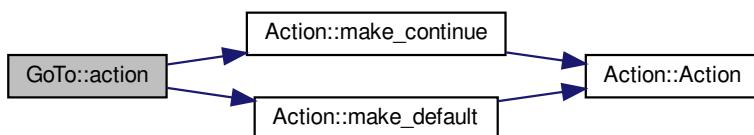
The [GoTo](#) class delegates the action choice to the contained [Chain](#). If the [Chain](#) decides on the continue action this method will return the default instead since final decision for a [GoTo](#) should be with the contained [Chain](#) or with the default action. In other words, once a [GoTo](#) rule matches, no further rule in the chain should ever be ran, regardless of the contained chain.

Implements [Rule](#).

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

References [Rule::condition_](#), [Action::CONTINUE](#), [Action::make_continue\(\)](#), and [Action::make_default\(\)](#).

Here is the call graph for this function:



15.14.3.2 clone()

```
std::unique_ptr< Rule > GoTo::clone ( ) const [virtual]
```

Return a copy of the [Rule](#) polymorphically.

This allows [Rule](#)'s to be copied without knowing the derived type.

Returns

A pointer to a new object with base type [Rule](#) which is an exact copy of this one.

Implements [Rule](#).

Definition at line 147 of file [GoTo.cpp](#).

References [Rule::condition_](#).

15.14.3.3 operator"!=()

```
bool GoTo::operator!= (
    const Rule & other ) const [virtual]
```

Inequality comparison.

Compares the type of the [Rule](#) and the condition ([If](#)) if set.

Parameters

<i>other</i>	The other rule to compare this to.
--------------	------------------------------------

Return values

<i>true</i>	if this rule is not the same as <i>other</i> .
<i>false</i>	if this rule is the same as <i>other</i> .

Compares the chain and priority (if set) associated with the rule as well.

Implements [Rule](#).

Definition at line 175 of file [GoTo.cpp](#).

References [Rule::condition_](#).

15.14.3.4 operator==()

```
bool GoTo::operator== (
    const Rule & other ) const [virtual]
```

Equality comparison.

Compares the type of the [Rule](#) and the condition ([If](#)) if set.

Parameters

<i>other</i>	The other rule to compare this to.
--------------	------------------------------------

Returns

<i>true</i>	if this rule is the same as <i>other</i> .
<i>false</i>	if this rule is not the same as <i>other</i> .

Compares the chain and priority (if set) associated with the rule as well.

Implements [Rule](#).

Definition at line 162 of file [GoTo.cpp](#).

References [Rule::condition_](#).

15.14.3.5 print_()

```
std::ostream & GoTo::print_ (
    std::ostream & os ) const [protected], [virtual]
```

Print the rule to the given output stream.

Parameters

<i>os</i>	The output stream to print to.
-----------	--------------------------------

Returns

The output stream.

Prints "goto <Chain Name> <If Statement>" or "goto <Chain Name> with priority <If Statement> with priority <priority>" if the priority is given.

Implements [Rule](#).

Definition at line 93 of file [GoTo.cpp](#).

References [Rule::condition_](#).

15.14.4 Member Data Documentation

15.14.4.1 condition_

```
std::optional<If> Rule::condition_ [protected], [inherited]
```

Definition at line 91 of file [Rule.hpp](#).

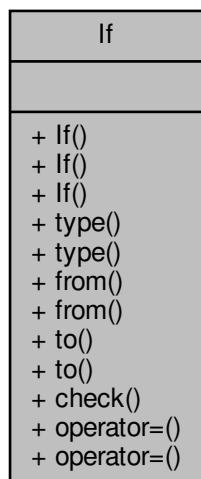
The documentation for this class was generated from the following files:

- [GoTo.hpp](#)
- [GoTo.cpp](#)

15.15 If Class Reference

```
#include <If.hpp>
```

Collaboration diagram for If:



Public Member Functions

- `If (std::optional< unsigned long > id={}, std::optional< MAVSubnet > source={}, std::optional< MAVSubnet > dest={})`
- `If (const If &other)=default`
- `If (If &&other)=default`
- `If & type (unsigned long id)`
- `If & type (const std::string &name)`
- `If & from (MAVSubnet subnet)`
- `If & from (const std::string &subnet)`
- `If & to (MAVSubnet subnet)`
- `If & to (const std::string &subnet)`
- `bool check (const Packet &packet, const MAVAddress &address) const`
- `If & operator= (const If &other)=default`
- `If & operator= (If &&other)=default`

Friends

- `bool operator== (const If &lhs, const If &rhs)`
- `bool operator!= (const If &lhs, const If &rhs)`
- `std::ostream & operator<< (std::ostream &os, const If &if_)`

15.15.1 Detailed Description

An if statement used to determine if a packet matches a rule.

This uses the type, source, and destination of a packet to determine if it matches.

Definition at line 35 of file `If.hpp`.

15.15.2 Constructor & Destructor Documentation

15.15.2.1 If() [1/3]

```
If::If (
    std::optional< unsigned long > id = {},
    std::optional< MAVSubnet > source = {},
    std::optional< MAVSubnet > dest = {} )
```

Construct an `If` statement.

The default is to allow any type of packet from any address to any address.

The `type`, `to`, and `from` methods can be used to apply conditions after construction. Some examples are:

- `If().type("PING").from("1.0/8").to("255.0");`
- `If().type("HEARTBEAT").from("255.0/8");`
- `If().type("SET_MODE").to("255.0/8");`
- `If().from("255.0/8");`

Parameters

<i>id</i>	The packet ID to match. If {} (std::nullopt) then any packet ID will match. The default is {}.
<i>source</i>	The subnet a source address must be in to match. If {} (std::nullopt) then any source address will match. The default is {}.
<i>dest</i>	The subnet a destination address must be in to match. If {} (std::nullopt) then any destination address will match. The default is {}.

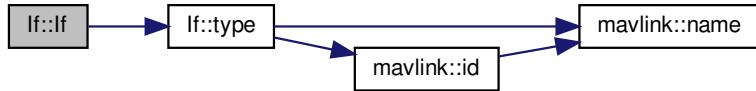
Exceptions

<code>std::invalid_argument</code>	if the given <code>id</code> is not valid.
------------------------------------	--

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

References [type\(\)](#).

Here is the call graph for this function:



15.15.2.2 If() [2/3]

```
If::If (
    const If & other ) [default]
```

Copy constructor.

Parameters

<i>other</i>	If to copy from.
--------------	------------------

15.15.2.3 If() [3/3]

```
If::If (
```

```
If && other ) [default]
```

Move constructor.

Parameters

<i>other</i>	If to move from.
--------------	------------------

15.15.3 Member Function Documentation

15.15.3.1 check()

```
bool If::check (
    const Packet & packet,
    const MAVAddress & address ) const
```

Check whether a [Packet](#) and [MAVAddress](#) combination matches.

Parameters

<i>packet</i>	The packet to check for a match.
<i>address</i>	The address the packet is to be sent to.

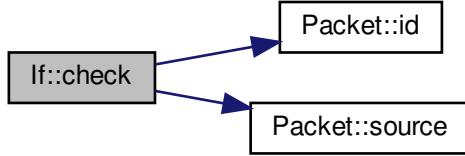
Return values

<i>true</i>	If the packet matches the type, source subnet (by MAVSubnet::contains), and destination subnet (by MAVSubnet::contains) of the if statement.
<i>false</i>	If any of the packet type, source subnet, or destination subnet does not match.

Definition at line 160 of file [If.cpp](#).

References [Packet::id\(\)](#), and [Packet::source\(\)](#).

Here is the call graph for this function:



15.15.3.2 from() [1/2]

```
If & If::from(
    MAVSubnet subnet )
```

Set subnet for source address matching using `MAVSubnet`.

Parameters

<code>subnet</code>	The subnet used for source address matching.
---------------------	--

Returns

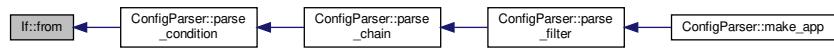
A reference to itself.

See also

`from(const std::string &subnet)`

Definition at line 98 of file `If.cpp`.

Here is the caller graph for this function:



15.15.3.3 from() [2/2]

```
If & If::from (
    const std::string & subnet )
```

Set subnet for source address matching by string.

See [MAVSubnet::MAVSubnet\(std::string address\)](#) for the acceptable formats and possible errors.

Parameters

<code>subnet</code>	The subnet used for source address matching.
---------------------	--

Returns

A reference to itself.

See also

[from\(MAVSubnet subnet\)](#)

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

15.15.3.4 operator=() [1/2]

```
If& If::operator= (
    const If & other ) [default]
```

Assignment operator.

Parameters

<code>other</code>	If to copy from.
--------------------	------------------

15.15.3.5 operator=() [2/2]

```
If& If::operator= (
    If && other ) [default]
```

Assignment operator (by move semantics).

Parameters

<code>other</code>	If to move from.
--------------------	------------------

15.15.3.6 to() [1/2]

```
If & If::to (
    MAVSubnet subnet )
```

Set subnet for destination address matching using [MAVSubnet](#).

Parameters

<code>subnet</code>	The subnet used for destination address matching.
---------------------	---

Returns

A reference to itself.

See also

[to\(const std::string &subnet\)](#)

Definition at line 127 of file [If.cpp](#).

Here is the caller graph for this function:

**15.15.3.7 to() [2/2]**

```
If & If::to (
    const std::string & subnet )
```

Set subnet for destination address matching by string.

See [MAVSubnet::MAVSubnet\(std::string address\)](#) for the acceptable formats and possible errors.

Parameters

<i>subnet</i>	The subnet used for destination address matching.
---------------	---

Returns

A reference to itself.

See also

[to\(MAVSubnet subnet\)](#)

Definition at line 143 of file [If.cpp](#).

15.15.3.8 type() [1/2]

```
If & If::type (
    unsigned long id )
```

Set the packet type to match, by ID.

Parameters

<i>id</i>	The packet ID to match.
-----------	-------------------------

Returns

A reference to itself.

Exceptions

<i>std::invalid_argument</i>	if the given <i>id</i> is not valid.
------------------------------	--------------------------------------

See also

[type\(const std::string &name\)](#)

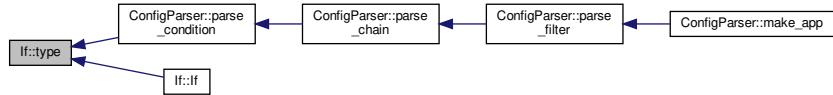
Definition at line 69 of file [If.cpp](#).

References [mavlink::id\(\)](#), and [mavlink::name\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.15.3.9 type() [2/2]

```
If & If::type (
    const std::string & name )
```

Set the packet type to match, by name.

Parameters

<i>name</i>	The packet name to match.
-------------	---------------------------

Returns

A reference to itself.

Exceptions

<i>std::invalid_argument</i>	if the given message name is not valid.
------------------------------	---

See also

[type\(unsigned long id\)](#)

Definition at line 85 of file [If.cpp](#).

References [mavlink::id\(\)](#), and [mavlink::name\(\)](#).

Here is the call graph for this function:



15.15.4 Friends And Related Function Documentation

15.15.4.1 operator"!=

```
bool operator!= (
    const If & lhs,
    const If & rhs ) [friend]
```

Inequality comparison.

Parameters

<i>lhs</i>	The left hand side if statement.
<i>rhs</i>	The right hand side if statement.

Return values

<i>true</i>	if <i>lhs</i> and <i>rhs</i> are not the same.
<i>false</i>	if <i>lhs</i> and <i>rhs</i> are the same.

Definition at line 209 of file [If.cpp](#).

15.15.4.2 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const If & if_ ) [friend]
```

Print the [If](#) statement to the given output stream.

Some examples are:

- if PING from 1.0/8 to 255.0
- if HEARTBEAT from 255.0/8
- if SET_MODE to 255.0
- if from 255.0/8

Definition at line [224](#) of file [If.cpp](#).

15.15.4.3 operator==

```
bool operator== (
    const If & lhs,
    const If & rhs ) [friend]
```

Equality comparison.

Parameters

<i>lhs</i>	The left hand side if statement.
<i>rhs</i>	The right hand side if statement.

Return values

<i>true</i>	if <i>lhs</i> and <i>rhs</i> are the same.
<i>false</i>	if <i>lhs</i> and <i>rhs</i> are not the same.

Definition at line [194](#) of file [If.cpp](#).

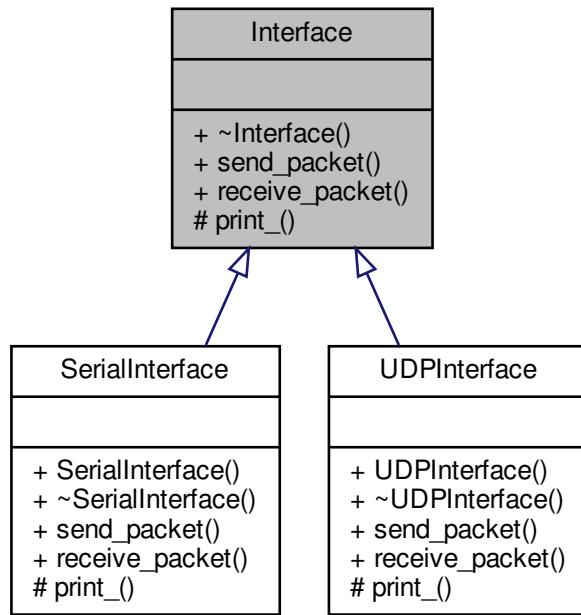
The documentation for this class was generated from the following files:

- [If.hpp](#)
- [If.cpp](#)

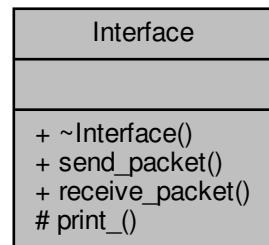
15.16 Interface Class Reference

```
#include <Interface.hpp>
```

Inheritance diagram for Interface:



Collaboration diagram for Interface:



Public Member Functions

- virtual ~Interface ()
- virtual void [send_packet](#) (const std::chrono::nanoseconds &timeout)=0
- virtual void [receive_packet](#) (const std::chrono::nanoseconds &timeout)=0

Protected Member Functions

- virtual std::ostream & [print_](#) (std::ostream &os) const =0

Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [Interface](#) &interface)

15.16.1 Detailed Description

The base class for all interfaces.

Derived classes should add one or more connections to queue packets for sending.

Definition at line [33](#) of file [Interface.hpp](#).

15.16.2 Constructor & Destructor Documentation

15.16.2.1 ~Interface()

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

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

15.16.3 Member Function Documentation

15.16.3.1 print_()

```
virtual std::ostream& Interface::print_ (
    std::ostream & os ) const [protected], [pure virtual]
```

Print the interface to the given output stream.

Parameters

<i>os</i>	The output stream to print to.
-----------	--------------------------------

Returns

The output stream.

Implemented in [UDPIInterface](#), and [SerialInterface](#).

15.16.3.2 receive_packet()

```
virtual void Interface::receive_packet (
    const std::chrono::nanoseconds & timeout ) [pure virtual]
```

Receive a packet on the interface.

Parameters

<i>timeout</i>	The maximum amount of time to wait for incoming data.
----------------	---

Implemented in [UDPIInterface](#), and [SerialInterface](#).

15.16.3.3 send_packet()

```
virtual void Interface::send_packet (
    const std::chrono::nanoseconds & timeout ) [pure virtual]
```

Send a packet from one of the interface's connections.

Note

Which connection to take this packet from is not defined but it must not starve any one of the [Interface](#)'s connections.

Parameters

<i>timeout</i>	The maximum amount of time to wait for a packet to be available for sending.
----------------	--

Implemented in [UDPIInterface](#), and [SerialInterface](#).

15.16.4 Friends And Related Function Documentation

15.16.4.1 operator<<

```
std::ostream & operator<< (
    std::ostream & os,
    const Interface & interface ) [friend]
```

Print the given [Interface](#) to the given output stream.

Note

This is a polymorphic print. Therefore, it can print any derived class as well.

Some examples are:

```
serial {
    device /dev/ttyUSB0;
    baudrate 115200;
    flow_control yes;
}
```

```
udp {
    port 14500;
    address 127.0.0.1;
}
```

Parameters

<i>os</i>	The output stream to print to.
<i>interface</i>	The interface (or any child of the Interface) to print.

Returns

The output stream.

Definition at line 58 of file [Interface.cpp](#).

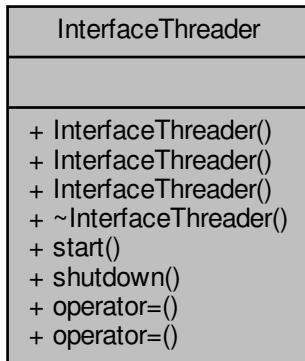
The documentation for this class was generated from the following files:

- [Interface.hpp](#)
- [Interface.cpp](#)

15.17 InterfaceThreader Class Reference

```
#include <InterfaceThreader.hpp>
```

Collaboration diagram for InterfaceThreader:



Public Types

- enum `Threads` { `START`, `DELAY_START` }

Public Member Functions

- `InterfaceThreader (std::unique_ptr< Interface > interface, std::chrono::microseconds=std::chrono::microseconds(100000), Threads start_threads=InterfaceThreader::START)`
- `InterfaceThreader (const InterfaceThreader &other)=delete`
- `InterfaceThreader (InterfaceThreader &&other)=delete`
- `~InterfaceThreader ()`
- `void start ()`
- `void shutdown ()`
- `InterfaceThreader & operator= (const InterfaceThreader &other)=delete`
- `InterfaceThreader & operator= (InterfaceThreader &&other)=delete`

15.17.1 Detailed Description

A multithreaded interface runner.

This class runs a given interface. It does this by calling the `Interface`'s `Interface::send_packet` and `Interface::receive_packet` methods repeatedly until the runner is shutdown. This is a threaded runner and thus it runs these methods in two separate threads which are managed by the `InterfaceThreader` class.

Call the `shutdown` method to stop running the interface and cleanup the threads.

Note

This is the only place where threading occurs in mavtables.

Definition at line 42 of file [InterfaceThreader.hpp](#).

15.17.2 Member Enumeration Documentation

15.17.2.1 Threads

```
enum InterfaceThreader::Threads
```

Enumerator

START	Start the interface (and worker threads) immediately.
DELAY_START	Delay starting, use start to launch threads.

Definition at line 45 of file [InterfaceThreader.hpp](#).

15.17.3 Constructor & Destructor Documentation

15.17.3.1 InterfaceThreader() [1/3]

```
InterfaceThreader::InterfaceThreader (
    std::unique_ptr< Interface > interface,
    std::chrono::microseconds timeout = std::chrono::microseconds(100000),
    Threads start_threads = InterfaceThreader::START )
```

Construct and optionally start an interface threadder.

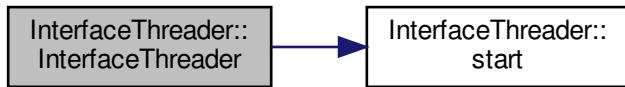
Parameters

<i>interface</i>	The Interface to run in TX/RX threads. It's Interface::send_packet and Interface::receive_packet methods will be called repeatedly in two separate worker threads.
<i>timeout</i>	The maximum amount of time to wait for incoming data or a packet to transmit. The default value is 100000 us (100 ms).
<i>start_threads</i>	Set to InterfaceThreader::START (the default value) to start the interface (including worker threads) on construction. Set to InterfaceThreader::DELAY_START to delay starting the interface (and worker threads) until the start method is called.

Definition at line 80 of file [InterfaceThreader.cpp](#).

References [START](#), and [start\(\)](#).

Here is the call graph for this function:



15.17.3.2 [InterfaceThreader\(\)](#) [2/3]

```
InterfaceThreader::InterfaceThreader (
    const InterfaceThreader & other ) [delete]
```

15.17.3.3 [InterfaceThreader\(\)](#) [3/3]

```
InterfaceThreader::InterfaceThreader (
    InterfaceThreader && other ) [delete]
```

15.17.3.4 [~InterfaceThreader\(\)](#)

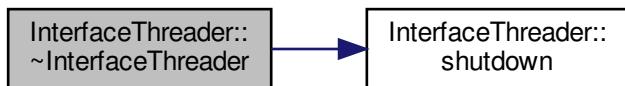
```
InterfaceThreader::~InterfaceThreader ( )
```

Shutdown the interface and its associated worker threads.

Definition at line 97 of file [InterfaceThreader.cpp](#).

References [shutdown\(\)](#).

Here is the call graph for this function:



15.17.4 Member Function Documentation

15.17.4.1 operator=() [1/2]

```
InterfaceThreader& InterfaceThreader::operator= (
    const InterfaceThreader & other ) [delete]
```

15.17.4.2 operator=() [2/2]

```
InterfaceThreader& InterfaceThreader::operator= (
    InterfaceThreader && other ) [delete]
```

15.17.4.3 shutdown()

```
void InterfaceThreader::shutdown ( )
```

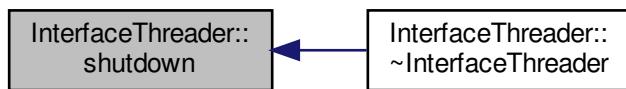
Shutdown the interface and its associated worker threads.

Note

This will always be called by the interface's destructor.

Definition at line 119 of file [InterfaceThreader.cpp](#).

Here is the caller graph for this function:



15.17.4.4 start()

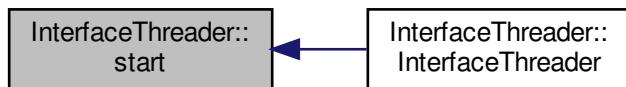
```
void InterfaceThreader::start ( )
```

Start the worker threads for the interface.

This starts the receiver and transmitter threads.

Definition at line 107 of file [InterfaceThreader.cpp](#).

Here is the caller graph for this function:



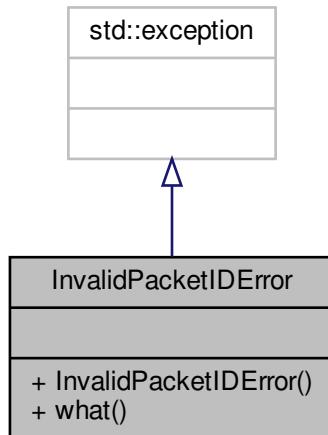
The documentation for this class was generated from the following files:

- [InterfaceThreader.hpp](#)
- [InterfaceThreader.cpp](#)

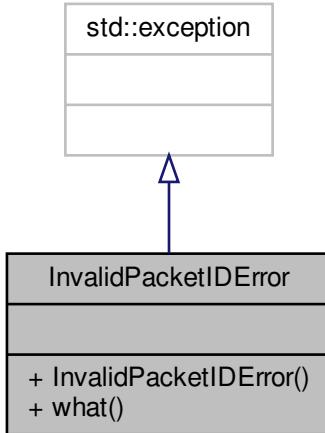
15.18 InvalidPacketIDError Class Reference

```
#include <InvalidPacketIDError.hpp>
```

Inheritance diagram for InvalidPacketIDError:



Collaboration diagram for InvalidPacketIDError:



Public Member Functions

- [InvalidPacketIDError](#) (unsigned long id)
- const char * [what\(\)](#) const noexcept

15.18.1 Detailed Description

Exception type emitted when parsing a packet with an invalid ID.

Definition at line [28](#) of file [InvalidPacketIDError.hpp](#).

15.18.2 Constructor & Destructor Documentation

15.18.2.1 InvalidPacketIDError()

```
InvalidPacketIDError::InvalidPacketIDError (
```

```
        unsigned long id )
```

Construct a [InvalidPacketIDError](#) given a packet ID.

Parameters

<i>id</i>	The packet ID.
-----------	----------------

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

15.18.3 Member Function Documentation

15.18.3.1 what()

```
const char * InvalidPacketIDError::what () const [noexcept]
```

Return error message string.

Returns

Error message string.

Definition at line 39 of file [InvalidPacketIDError.cpp](#).

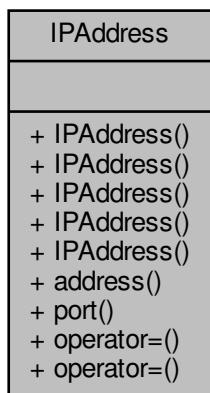
The documentation for this class was generated from the following files:

- [InvalidPacketIDError.hpp](#)
- [InvalidPacketIDError.cpp](#)

15.19 IPAddress Class Reference

```
#include <IPAddress.hpp>
```

Collaboration diagram for IPAddress:



Public Member Functions

- `IPAddress (const IPAddress &other)=default`
- `IPAddress (IPAddress &&other)=default`
- `IPAddress (const IPAddress &other, unsigned int port)`
- `IPAddress (unsigned long address, unsigned int port=0)`
- `IPAddress (std::string address)`
- `unsigned long address () const`
- `unsigned int port () const`
- `IPAddress & operator=(const IPAddress &other)=default`
- `IPAddress & operator=(IPAddress &&other)=default`

Friends

- `std::ostream & operator<< (std::ostream &os, const IPAddress &ipaddress)`

Related Functions

(Note that these are not member functions.)

- `bool operator==(const IPAddress &lhs, const IPAddress &rhs)`
- `bool operator!=(const IPAddress &lhs, const IPAddress &rhs)`
- `bool operator<(const IPAddress &lhs, const IPAddress &rhs)`
- `bool operator>(const IPAddress &lhs, const IPAddress &rhs)`
- `bool operator<=(const IPAddress &lhs, const IPAddress &rhs)`
- `bool operator>=(const IPAddress &lhs, const IPAddress &rhs)`
- `IPAddress dnslookup (const std::string &url)`

15.19.1 Detailed Description

An IP address with optional port number.

Definition at line 31 of file [IPAddress.hpp](#).

15.19.2 Constructor & Destructor Documentation

15.19.2.1 IPAddress() [1/5]

```
IPAddress::IPAddress (
    const IPAddress & other ) [default]
```

Copy constructor.

Parameters

<i>other</i>	IP address to copy from.
--------------	--------------------------

15.19.2.2 IPAddress() [2/5]

```
IPAddress::IPAddress (
    const IPAddress && other ) [default]
```

Move constructor.

Parameters

<i>other</i>	IP address to move from.
--------------	--------------------------

15.19.2.3 IPAddress() [3/5]

```
IPAddress::IPAddress (
    const IPAddress & other,
    unsigned int port )
```

Construct IP address from another IP address, changing the port number.

Copy constructor that also changes the port.

Parameters

<i>other</i>	IP address to copy from.
<i>port</i>	Port number (0 - 65535). A port number of 0 has the special meaning of no specific port.

Exceptions

<i>std::out_of_range</i>	if the port number is outside of the allowed 16 bit range.
--------------------------	--

Definition at line 88 of file [IPAddress.cpp](#).

References [port\(\)](#).

Here is the call graph for this function:



15.19.2.4 IPAddress() [4/5]

```
IPAddress::IPAddress (
    unsigned long address,
    unsigned int port = 0 )
```

Construct IP address from address and port number.

Parameters

<i>address</i>	32-bit IP address in system byte order (0x00000000 - 0xFFFFFFFF).
<i>port</i>	Port number (0 - 65535). A port number of 0 has the special meaning of no specific port.

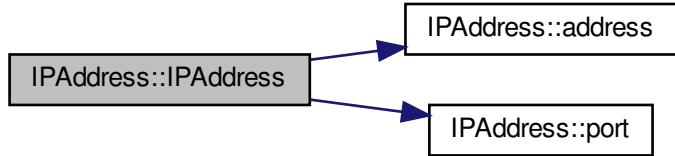
Exceptions

<i>std::out_of_range</i>	if either the IP address or the port number is outside of the respectively allowed 32 or 16 bit ranges.
--------------------------	---

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

References [address\(\)](#), and [port\(\)](#).

Here is the call graph for this function:



15.19.2.5 IPAddress() [5/5]

```
IPAddress::IPAddress (
    std::string address )
```

Construct IP address from a string.

Parse a string of the form "<IP Address>" or "<IP Address>:<Port Number>".

Some examples are:

- "127.0.0.1"
- "127.0.0.1:8888"
- "183.125.120.42:443"

If no port is given the 0 port is used which represents no specific port.

Parameters

<code>address</code>	String representing the IP address and optionally the port number.
----------------------	--

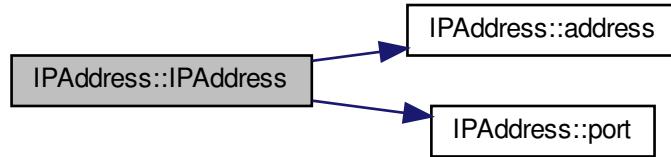
Exceptions

<code>std::invalid_argument</code>	if the string does not represent a valid IP address.
<code>std::out_of_range</code>	if an address octet or the port number is out of range.

Definition at line 127 of file [IPAddress.cpp](#).

References [address\(\)](#), and [port\(\)](#).

Here is the call graph for this function:



15.19.3 Member Function Documentation

15.19.3.1 address()

```
unsigned long IPAddress::address ( ) const
```

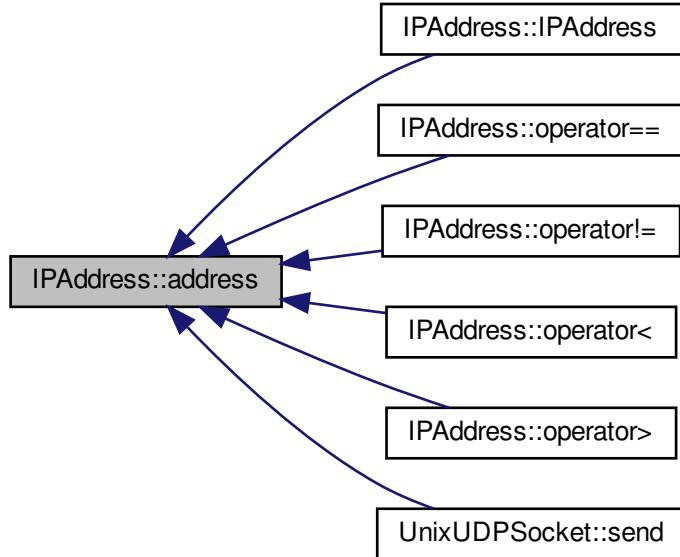
Return the IP address.

Returns

The 32-bit IP address (in system byte order) as an integer (0x00000000 - 0xFFFFFFFF).

Definition at line 205 of file [IPAddress.cpp](#).

Here is the caller graph for this function:



15.19.3.2 operator=() [1/2]

```
IPAddress& IPAddress::operator= (
    const IPAddress & other ) [default]
```

Assignment operator.

Parameters

<i>other</i>	IP address to copy from.
--------------	--------------------------

15.19.3.3 operator=() [2/2]

```
IPAddress& IPAddress::operator= (
    IPAddress && other ) [default]
```

Assignment operator (by move semantics).

Parameters

<i>other</i>	IP address to move from.
--------------	--------------------------

15.19.3.4 port()

```
unsigned int IPAddress::port ( ) const
```

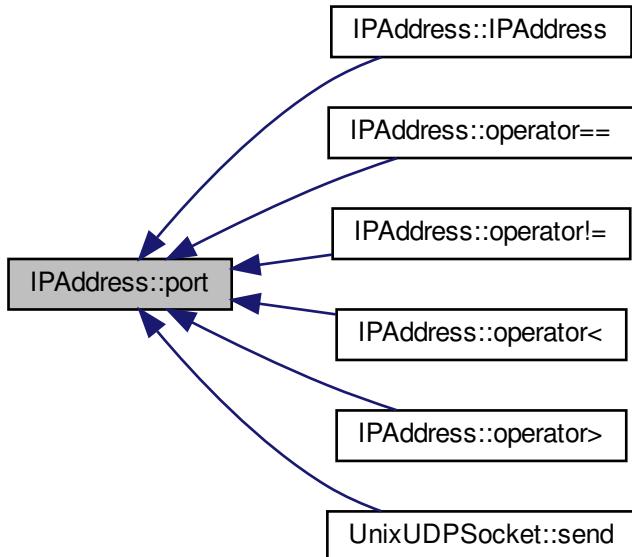
Return the port.

Returns

The port number (0 - 65535).

Definition at line 215 of file [IPAddress.cpp](#).

Here is the caller graph for this function:

**15.19.4 Friends And Related Function Documentation**

15.19.4.1 dnslookup()

```
IPAddress dnslookup (
    const std::string & url ) [related]
```

Lookup an IP address based on a hostname.

Warning

Currently only supports IPv4.

Note

Currently only UNIX based operating system are supported.

Parameters

<i>url</i>	The URL to get an IP address for.
------------	-----------------------------------

Returns

IP addresses corresponding to the given URL.

Exceptions

<i>DNSLookupError</i>	if the address cannot be found.
-----------------------	---------------------------------

Definition at line 371 of file [IPAddress.cpp](#).

15.19.4.2 operator"!="()

```
bool operator!= (
    const IPAddress & lhs,
    const IPAddress & rhs ) [related]
```

Inequality comparison.

Note

Compares address and port number.

Parameters

<i>lhs</i>	The left hand side IP address.
<i>rhs</i>	The right hand side IP address.

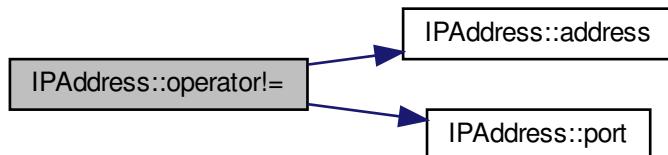
Return values

<i>true</i>	if <i>lhs</i> and <i>rhs</i> do not have the same address and port.
<i>false</i>	if <i>lhs</i> and <i>rhs</i> have the same address and port.

Definition at line 247 of file [IPAddress.cpp](#).

References [address\(\)](#), and [port\(\)](#).

Here is the call graph for this function:

**15.19.4.3 operator<()**

```
bool operator< (
    const IPAddress & lhs,
    const IPAddress & rhs ) [related]
```

Less than comparison.

Note

The address is considered first followed by the port.

Parameters

<i>lhs</i>	The left hand side IP address.
<i>rhs</i>	The right hand side IP address.

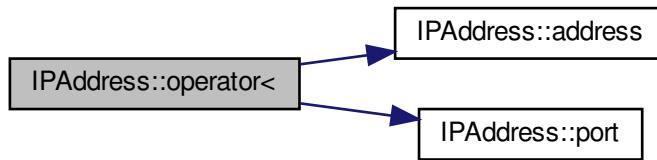
Return values

<code>true</code>	if <code>lhs</code> is less than <code>rhs</code> .
<code>false</code>	if <code>lhs</code> is not less than <code>rhs</code> .

Definition at line 263 of file [IPAddress.cpp](#).

References [address\(\)](#), and [port\(\)](#).

Here is the call graph for this function:



15.19.4.4 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const IPAddress & ipaddress ) [friend]
```

Print the IP address to the given output stream.

The format is "<IP Address>" or "<IP Address>:<Port Number>" if the port number is nonzero and "<IP Address>" if the port is 0.

Some examples are:

- 127.0.0.1
- 127.0.0.1:14555
- 183.125.120.42:443

Note

The string constructor [IPAddress\(std::string\)](#) and the output stream operator are inverses and thus:

```
std::string addr = "127.0.0.1:14555"
str(IPAddress(addr)) == addr
```

Parameters

<i>os</i>	The output stream to print to.
<i>ipaddress</i>	The IP address to print.

Returns

The output stream.

Definition at line 340 of file [IPAddress.cpp](#).

15.19.4.5 operator<=()

```
bool operator<= (
    const IPAddress & lhs,
    const IPAddress & rhs ) [related]
```

Less than or equal comparison.

Note

The address is considered first followed by the port.

Parameters

<i>lhs</i>	The left hand side IP address.
<i>rhs</i>	The right hand side IP address.

Return values

<i>true</i>	if <i>lhs</i> is less than or equal to <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is greater than <i>rhs</i> .

Definition at line 297 of file [IPAddress.cpp](#).

15.19.4.6 operator==()

```
bool operator== (
    const IPAddress & lhs,
    const IPAddress & rhs ) [related]
```

Equality comparison.

Note

Compares address and port number.

Parameters

<i>lhs</i>	The left hand side IP address.
<i>rhs</i>	The right hand side IP address.

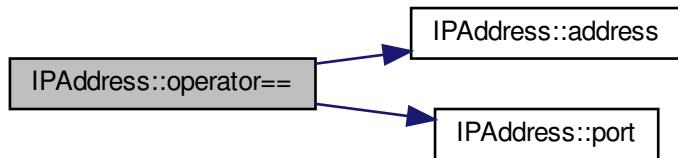
Return values

<i>true</i>	if <i>lhs</i> and <i>rhs</i> have the same address and port.
<i>false</i>	if <i>lhs</i> and <i>rhs</i> do not have the same address and port.

Definition at line 231 of file [IPAddress.cpp](#).

References [address\(\)](#), and [port\(\)](#).

Here is the call graph for this function:

**15.19.4.7 operator>()**

```
bool operator> (
    const IPAddress & lhs,
    const IPAddress & rhs ) [related]
```

Greater than comparison.

Note

The address is considered first followed by the port.

Parameters

<i>lhs</i>	The left hand side IP address.
<i>rhs</i>	The right hand side IP address.

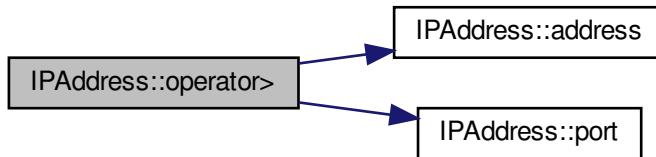
Return values

<i>true</i>	if <i>lhs</i> is greater than <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is not greater than <i>rhs</i> .

Definition at line 280 of file [IPAddress.cpp](#).

References [address\(\)](#), and [port\(\)](#).

Here is the call graph for this function:

**15.19.4.8 operator>=()**

```
bool operator>= (
    const IPAddress & lhs,
    const IPAddress & rhs ) [related]
```

Greater than comparison.

Note

The address is considered first followed by the port.

Parameters

<i>lhs</i>	The left hand side IP address.
<i>rhs</i>	The right hand side IP address.

Return values

<i>true</i>	if <code>lhs</code> is greater than or equal to <code>rhs</code> .
<i>false</i>	if <code>lhs</code> is less than <code>rhs</code> .

Definition at line 313 of file [IPAddress.cpp](#).

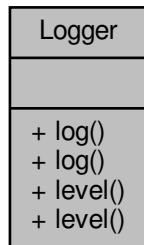
The documentation for this class was generated from the following files:

- [IPAddress.hpp](#)
- [IPAddress.cpp](#)

15.20 Logger Class Reference

```
#include <Logger.hpp>
```

Collaboration diagram for Logger:



Static Public Member Functions

- static void [log](#) (std::string message)
- static void [log](#) (unsigned int [level](#), std::string message)
- static void [level](#) (unsigned int level)
- static unsigned int [level](#) ()

15.20.1 Detailed Description

A global static logger for use by all of mavtables.

Note

Only supports writing to stdout.

Definition at line 30 of file [Logger.hpp](#).

15.20.2 Member Function Documentation

15.20.2.1 level() [1/2]

```
void Logger::level (
    unsigned int level )  [static]
```

Set the logging level.

A higher level indicates a higher verbosity of logging. A level of 0 will completely disable logging.

Parameters

<i>level</i>	The new logging level, valid values are 0 to 65535.
--------------	---

Definition at line 83 of file [Logger.cpp](#).

References [level\(\)](#).

Here is the call graph for this function:



15.20.2.2 level() [2/2]

```
unsigned int Logger::level ( )  [static]
```

Get the logging level.

Note

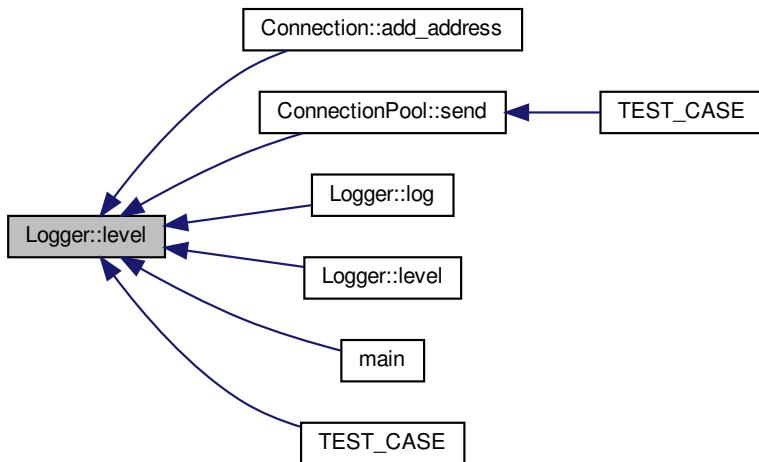
It is recommended to check the level before constructing a log message if the message is expensive to construct.

Returns

The current logging level.

Definition at line 96 of file [Logger.cpp](#).

Here is the caller graph for this function:

**15.20.2.3 log() [1/2]**

```
void Logger::log (
    std::string message ) [static]
```

Log a message with timestamp (at level 1).

This will log a message with the current date and time as the timestamp if the loglevel is set to at least 1.

Note

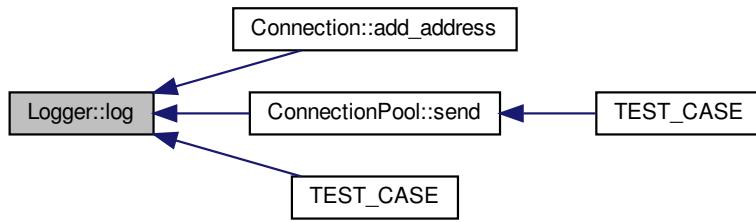
This is the only method (along with [log\(unsigned int, std::string\)](#)) that is threadsafe.

Parameters

<i>message</i>	The message to log.
----------------	---------------------

Definition at line 37 of file [Logger.cpp](#).

Here is the caller graph for this function:



15.20.2.4 log() [2/2]

```
void Logger::log (
    unsigned int level,
    std::string message ) [static]
```

Log a message with timestamp at the given level.

This will log a message with the current date and time as the timestamp if the loglevel is at least `level`.

Note

This is the only method (along with `log(std::string)`) that is threadsafe.

Parameters

<code>level</code>	The level to log the message at. If the logger's level is lower than this, the message will be discarded. The valid range is 1 to 65535, a value of 0 will be corrected to 1.
<code>message</code>	The message to log.

Remarks

Threadsafe (locking).

Definition at line 58 of file [Logger.cpp](#).

References [level\(\)](#).

Here is the call graph for this function:



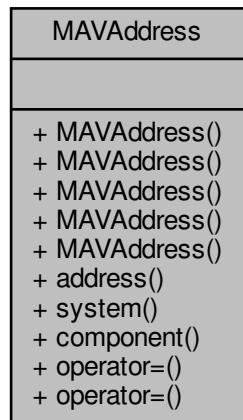
The documentation for this class was generated from the following files:

- [Logger.hpp](#)
- [Logger.cpp](#)

15.21 MAVAddress Class Reference

```
#include <MAVAddress.hpp>
```

Collaboration diagram for MAVAddress:



Public Member Functions

- `MAVAddress (const MAVAddress &other)=default`
- `MAVAddress (MAVAddress &&other)=default`
- `MAVAddress (unsigned int address)`
- `MAVAddress (unsigned int system, unsigned int component)`
- `MAVAddress (std::string address)`
- `unsigned int address () const`
- `unsigned int system () const`
- `unsigned int component () const`
- `MAVAddress & operator= (const MAVAddress &other)=default`
- `MAVAddress & operator= (MAVAddress &&other)=default`

Related Functions

(Note that these are not member functions.)

- `bool operator== (const MAVAddress &lhs, const MAVAddress &rhs)`
- `bool operator!= (const MAVAddress &lhs, const MAVAddress &rhs)`
- `bool operator< (const MAVAddress &lhs, const MAVAddress &rhs)`
- `bool operator> (const MAVAddress &lhs, const MAVAddress &rhs)`
- `bool operator<= (const MAVAddress &lhs, const MAVAddress &rhs)`
- `bool operator>= (const MAVAddress &lhs, const MAVAddress &rhs)`
- `std::ostream & operator<< (std::ostream &os, const MAVAddress &mavaddress)`

15.21.1 Detailed Description

A MAVLink address.

MAVLink addresses consist of a system and component and can be represented as two octets in the form:

`system.component`

Therefore, a system ID of 16 and a component ID of 8 can be represented as 16 . 8.

0 . 0 is reserved as the broadcast address.

Definition at line 38 of file [MAVAddress.hpp](#).

15.21.2 Constructor & Destructor Documentation

15.21.2.1 MAVAddress() [1/5]

```
MAVAddress::MAVAddress (
    const MAVAddress & other ) [default]
```

Copy constructor.

Parameters

<i>other</i>	MAVLink address to copy from.
--------------	-------------------------------

15.21.2.2 MAVAddress() [2/5]

```
MAVAddress::MAVAddress (
    MAVAddress && other ) [default]
```

Move constructor.

Parameters

<i>other</i>	MAVLink address to move from.
--------------	-------------------------------

15.21.2.3 MAVAddress() [3/5]

```
MAVAddress::MAVAddress (
    unsigned int address )
```

Construct a MAVLink address from an address in numeric representation.

The numeric representation of a MAVLink address is two bytes, the MSB contains the system ID and the LSB contains the component ID.

Parameters

<i>address</i>	Address (0 - 65535) with system ID encoded in MSB and component ID encoded in LSB.
----------------	--

Exceptions

<i>std::out_of_range</i>	if the address is not between 0 and 65535.
--------------------------	--

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

References [address\(\)](#).

Here is the call graph for this function:



15.21.2.4 MAVAddress() [4/5]

```
MAVAddress::MAVAddress (
    unsigned int system,
    unsigned int component )
```

Construct a MAVLink address from the system ID and component ID.

Note

component=0 and *system*=0 is the broadcast address.

Parameters

<i>system</i>	System ID (0 - 255).
<i>component</i>	Component ID (0 - 255).

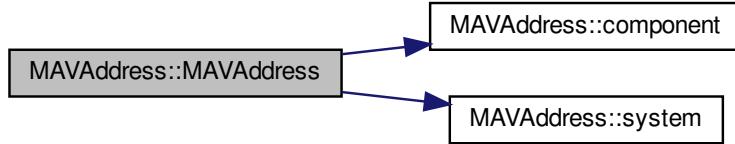
Exceptions

<i>std::out_of_range</i>	if either the <i>system</i> ID or the <i>component</i> ID is out of range.
--------------------------	--

Definition at line 87 of file [MAVAddress.cpp](#).

References [component\(\)](#), and [system\(\)](#).

Here is the call graph for this function:



15.21.2.5 `MAVAddress()` [5/5]

```
MAVAddress::MAVAddress (
    std::string address )
```

Construct MAVLink address from a string.

Parse a string of the form "<System ID>.<Component ID>".

Some examples are:

- "0.0"
- "16.8"
- "128.4"

Parameters

<code>address</code>	String representing the MAVLink address.
----------------------	--

Exceptions

<code>std::invalid_argument</code>	if the address string does not represent a valid MAVLink address.
<code>std::out_of_range</code>	if either the system ID or the component ID is out of range.

Definition at line 108 of file [MAVAddress.cpp](#).

References [address\(\)](#).

Here is the call graph for this function:



15.21.3 Member Function Documentation

15.21.3.1 address()

```
unsigned int MAVAddress::address ( ) const
```

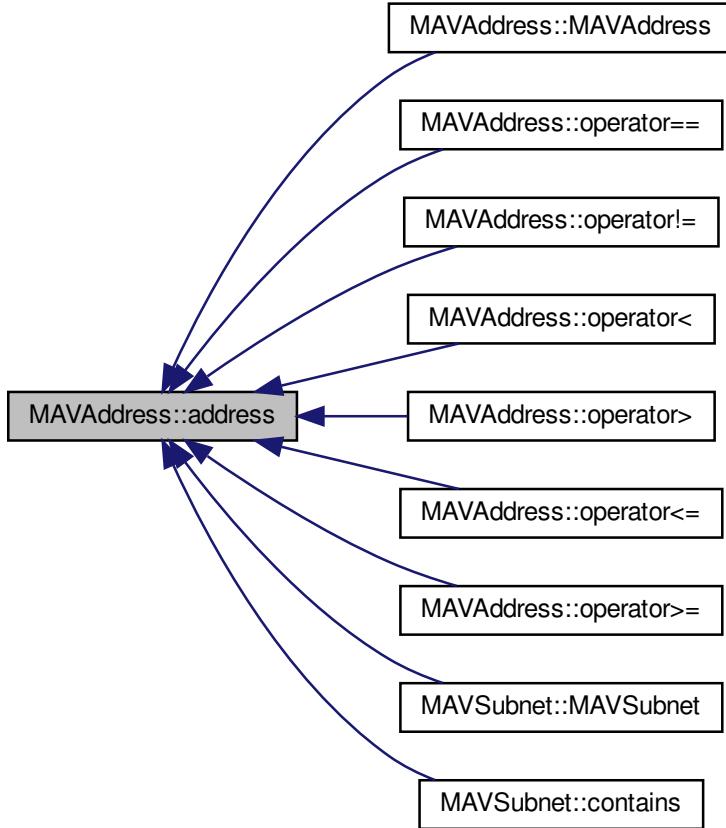
Return the MAVLink address in numeric form.

Returns

The MAVLink address as a two byte number with the system ID encoded in the MSB and the component ID in the LSB.

Definition at line 151 of file [MAVAddress.cpp](#).

Here is the caller graph for this function:



15.21.3.2 component()

```
unsigned int MAVAddress::component ( ) const
```

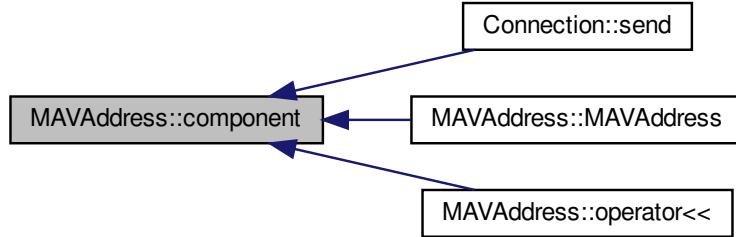
Return the Component ID.

Returns

The component ID (0 - 255).

Definition at line 171 of file [MAVAddress.cpp](#).

Here is the caller graph for this function:



15.21.3.3 operator=() [1/2]

```
MAVAddress& MAVAddress::operator= (
    const MAVAddress & other ) [default]
```

Assignment operator.

Parameters

<code>other</code>	MAVLink address to copy from.
--------------------	-------------------------------

15.21.3.4 operator=() [2/2]

```
MAVAddress& MAVAddress::operator= (
    MAVAddress && other ) [default]
```

Assignment operator (by move semantics).

Parameters

<code>other</code>	MAVLink address to move from.
--------------------	-------------------------------

15.21.3.5 system()

```
unsigned int MAVAddress::system ( ) const
```

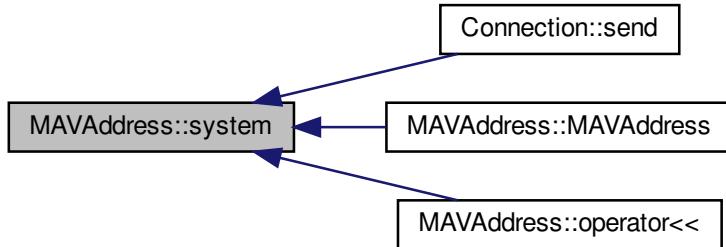
Return the System ID.

Returns

The system ID (0 - 255).

Definition at line 161 of file [MAVAddress.cpp](#).

Here is the caller graph for this function:



15.21.4 Friends And Related Function Documentation

15.21.4.1 operator"!=()

```
bool operator!= (
    const MAVAddress & lhs,
    const MAVAddress & rhs ) [related]
```

Inequality comparison.

Parameters

<i>lhs</i>	The left hand side MAVLink address.
<i>rhs</i>	The right hand side MAVLink address.

Return values

<i>true</i>	if <code>lhs</code> and <code>rhs</code> do not have the same system and component ID's
<i>false</i>	if <code>lhs</code> and <code>rhs</code> have the same system and component ID's.

Definition at line 201 of file [MAVAddress.cpp](#).

References [address\(\)](#).

Here is the call graph for this function:



15.21.4.2 operator<()

```
bool operator< (
    const MAVAddress & lhs,
    const MAVAddress & rhs ) [related]
```

Less than comparison.

Note

The System ID is considered first followed by the Component ID.

Parameters

<i>lhs</i>	The left hand side MAVLink address.
<i>rhs</i>	The right hand side MAVLink address.

Return values

<i>true</i>	if <code>lhs</code> is less than <code>rhs</code> .
<i>false</i>	if <code>lhs</code> is not less than <code>rhs</code> .

Definition at line 217 of file [MAVAddress.cpp](#).

References [address\(\)](#).

Here is the call graph for this function:



15.21.4.3 operator<<()

```
std::ostream & operator<< (
    std::ostream & os,
    const MAVAddress & mavaddress ) [related]
```

Print the MAVLink address to the given output stream.

The format is "<System ID>.<Component ID>".

Some examples are:

- 0.0
- 16.8
- 128.4

Note

The string constructor [MAVAddress\(std::string\)](#) and the output stream operator are inverses and thus:

```
std::string addr = "192.168"
str(MAVAddress(addr)) == addr
```

Parameters

<i>os</i>	The output stream to print to.
<i>mavaddress</i>	The MAVLink address to print.

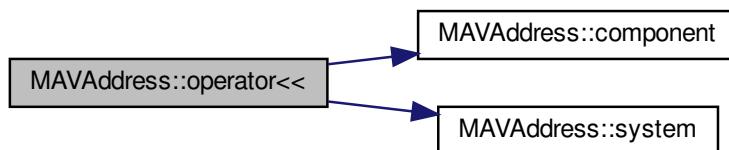
Returns

The output stream.

Definition at line 292 of file [MAVAddress.cpp](#).

References [component\(\)](#), and [system\(\)](#).

Here is the call graph for this function:

**15.21.4.4 operator<=()**

```
bool operator<= (
    const MAVAddress & lhs,
    const MAVAddress & rhs ) [related]
```

Less than or equal comparison.

Note

The System ID is considered first followed by the Component ID.

Parameters

<i>lhs</i>	The left hand side IP address.
<i>rhs</i>	The right hand side IP address.

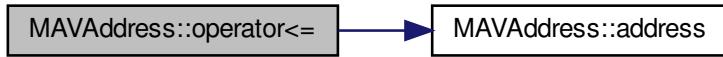
Return values

<i>true</i>	if <i>lhs</i> is less than or equal to <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is greater than <i>rhs</i> .

Definition at line 249 of file [MAVAddress.cpp](#).

References [address\(\)](#).

Here is the call graph for this function:



15.21.4.5 operator==()

```
bool operator== (
    const MAVAddress & lhs,
    const MAVAddress & rhs ) [related]
```

Equality comparison.

Parameters

<i>lhs</i>	The left hand side MAVLink address.
<i>rhs</i>	The right hand side MAVLink address.

Return values

<i>true</i>	if <i>lhs</i> and <i>rhs</i> have the same system and component ID's.
<i>false</i>	if <i>lhs</i> and <i>rhs</i> do not have the same system and component ID's.

Definition at line 186 of file [MAVAddress.cpp](#).

References [address\(\)](#).

Here is the call graph for this function:



15.21.4.6 operator>()

```
bool operator> (
    const MAVAddress & lhs,
    const MAVAddress & rhs ) [related]
```

Greater than comparison.

Note

The System ID is considered first followed by the Component ID.

Parameters

<i>lhs</i>	The left hand side MAVLink address.
<i>rhs</i>	The right hand side IP address.

Return values

<i>true</i>	if <i>lhs</i> is greater than <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is not greater than <i>rhs</i> .

Definition at line 233 of file [MAVAddress.cpp](#).

References [address\(\)](#).

Here is the call graph for this function:



15.21.4.7 operator>=()

```
bool operator>= (
    const MAVAddress & lhs,
    const MAVAddress & rhs ) [related]
```

Greater than comparison.

Note

The System ID is considered first followed by the Component ID.

Parameters

<i>lhs</i>	The left hand side IP address.
<i>rhs</i>	The right hand side IP address.

Return values

<i>true</i>	if <i>lhs</i> is greater than or equal to <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is less than <i>rhs</i> .

Definition at line 265 of file [MAVAddress.cpp](#).

References [address\(\)](#).

Here is the call graph for this function:



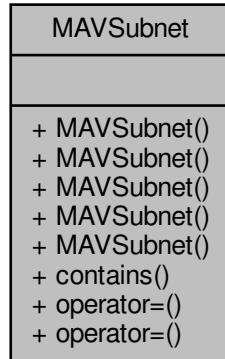
The documentation for this class was generated from the following files:

- [MAVAddress.hpp](#)
- [MAVAddress.cpp](#)

15.22 MAVSubnet Class Reference

```
#include <MAVSubnet.hpp>
```

Collaboration diagram for MAVSubnet:



Public Member Functions

- `MAVSubnet (const MAVSubnet &other)=default`
- `MAVSubnet (MAVSubnet &&other)=default`
- `MAVSubnet (const MAVAddress &address, unsigned int mask=0xFFFF)`
- `MAVSubnet (const MAVAddress &address, unsigned int system_mask, unsigned int component_mask)`
- `MAVSubnet (std::string address)`
- `bool contains (const MAVAddress &address) const`
- `MAVSubnet & operator= (const MAVSubnet &other)=default`
- `MAVSubnet & operator= (MAVSubnet &&other)=default`

Friends

- `bool operator== (const MAVSubnet &lhs, const MAVSubnet &rhs)`
- `bool operator!= (const MAVSubnet &lhs, const MAVSubnet &rhs)`
- `std::ostream & operator<< (std::ostream &os, const MAVSubnet &mavsubnet)`

15.22.1 Detailed Description

A MAVLink subnet.

Mavlink subnets work the same as IP subnets and allow the definition of a range of addresses. This is used to allow a single firewall rule to match multiple addresses.

See also

[MAVAddress](#)

Definition at line 36 of file [MAVSubnet.hpp](#).

15.22.2 Constructor & Destructor Documentation

15.22.2.1 MAVSubnet() [1/5]

```
MAVSubnet::MAVSubnet (
    const MAVSubnet & other ) [default]
```

Copy constructor.

Parameters

<i>other</i>	MAVLink subnet to copy from.
--------------	------------------------------

15.22.2.2 MAVSubnet() [2/5]

```
MAVSubnet::MAVSubnet (
    MAVSubnet && other ) [default]
```

Move constructor.

Parameters

<i>other</i>	MAVLink subnet to move from.
--------------	------------------------------

15.22.2.3 MAVSubnet() [3/5]

```
MAVSubnet::MAVSubnet (
    const MAVAddress & address,
    unsigned int mask = 0xFFFF )
```

Construct a MAVLink subnet from a MAVLink address and mask.

Parameters

<i>address</i>	MAVLink address of subnet.
<i>mask</i>	Two byte subnet mask, where the system mask is in the MSB and the component mask is in the LSB.

Exceptions

<code>std::out_of_range</code>	if the mask is not between 0x0000 and 0xFFFF.
--------------------------------	---

See also

Check if a [MAVAddress](#) is within the subnet with [contains](#).

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

15.22.2.4 MAVSubnet() [4/5]

```
MAVSubnet::MAVSubnet (
    const MAVAddress & address,
    unsigned int system_mask,
    unsigned int component_mask )
```

Construct MAVLink subnet from an address, system mask, and component mask.

Parameters

<code>address</code>	MAVLink address of subnet.
<code>system_mask</code>	One byte subnet system mask with the bits set that must match the subnet address for a MAVLink address to be contained within the subnet. Valid range is 0 to 255.
<code>component_mask</code>	One byte subnet component mask with the bits set that must match the subnet address for a MAVLink address to be contained within the subnet. Valid range is 0 to 255.

Exceptions

<code>std::out_of_range</code>	if the system and component masks are not each between 0x00 and 0xFF.
--------------------------------	---

See also

Check if a [MAVAddress](#) is within the subnet with [contains](#).

Definition at line 68 of file [MAVSubnet.cpp](#).

15.22.2.5 MAVSubnet() [5/5]

```
MAVSubnet::MAVSubnet (
    std::string subnet )
```

Construct a MAVLink subnet from a string.

There are four string forms of MAVLink subnets:

1. "<System ID>.<Component ID>:<System ID mask>.<Component ID mask>"
2. "<System ID>.<Component ID>/<bits>"
3. "<System ID>.<Component ID>\<bits>"
4. "<System ID>.<Component ID>"

The first form is self explanatory, but the 2nd and 3rd are not as simple. In the 2nd case the number of bits (0 - 16) is the number of bits from the left that must match for an address to be in the subnet. The 3rd form is like the 2nd, but does not require the system ID (first octet) to match and starts with the number of bits of the component ID (0 - 8) that must match from the left for an address to be in the subnet. The last form is shorthand for "<System ID>.<Component ID>/16", that is an exact match.

Below is a table relating the slash postfix to the subnet mask in <System mask>.<Component mask> notation.

Mask with /	Mask with \	Postfix (#bits)
255.255	out of range	16
255.254	out of range	15
255.252	out of range	14
255.248	out of range	13
255.240	out of range	12
255.224	out of range	11
255.192	out of range	10
255.128	out of range	9
255.0	0.255	8
254.0	0.254	7
252.0	0.252	6
248.0	0.248	5
240.0	0.240	4
224.0	0.224	3
192.0	0.192	2
128.0	0.128	1
0.0	0	0

Some examples are:

- "128.0/8" - Matches addresses with system ID 128 and any component ID.
- "128.0/9" - Matches addresses with system ID 128 and components ID's of 127 or less.
- "128.255/9" - Matches addresses with system ID 128 and components ID's of 128 or more.
- "128.0\1" - Matches addresses any system ID and components ID's of 127 or less.
- "128.255\1" - Matches addresses any system ID and components ID's of 128 or more.

- "255.0:128.240" - Matches system ID's 128 or greater and component ID's from 0 to 15.
- "255.16:128.240" - Matches system ID's 128 or greater and component ID's from 16 to 31.
- "255.16" - Matches only the address with system ID 255 and component ID 16.

Parameters

<code>subnet</code>	String representing the MAVLink subnet.
---------------------	---

Exceptions

<code>std::out_of_range</code>	if either the system ID or the component ID is out of range.
<code>std::out_of_range</code>	if the mask or slash bits are out of range.
<code>std::invalid_argument</code>	if the subnet string does not represent a valid MAVLink subnet.

See also

Check if a [MAVAddress](#) is within the subnet with [contains](#).

Definition at line 161 of file [MAVSubnet.cpp](#).

References [MAVAddress::address\(\)](#).

Here is the call graph for this function:



15.22.3 Member Function Documentation

15.22.3.1 contains()

```
bool MAVSubnet::contains (
    const MAVAddress & address ) const
```

Determine whether or not the subnet contains a given MAVLink address.

Parameters

<i>address</i>	The MAVLink address to test.
----------------	------------------------------

Return values

<i>true</i>	if <i>address</i> is part of the subnet.
-------------	--

Definition at line 250 of file [MAVSubnet.cpp](#).

References [MAVAddress::address\(\)](#).

Here is the call graph for this function:

**15.22.3.2 operator=() [1/2]**

```
MAVSubnet& MAVSubnet::operator= (
    const MAVSubnet & other ) [default]
```

Assignment operator.

Parameters

<i>other</i>	MAVLink subnet to copy from.
--------------	------------------------------

15.22.3.3 operator=() [2/2]

```
MAVSubnet& MAVSubnet::operator= (
    MAVSubnet && other ) [default]
```

Assignment operator (by move semantics).

Parameters

<i>other</i>	MAVLink subnet to move from.
--------------	------------------------------

15.22.4 Friends And Related Function Documentation**15.22.4.1 operator"!=**

```
bool operator!= (
    const MAVSubnet & lhs,
    const MAVSubnet & rhs ) [friend]
```

Inequality comparison.

Compares both address and mask.

Parameters

<i>lhs</i>	The left hand side MAVLink subnet.
<i>rhs</i>	The right hand side MAVLink subnet.

Return values

<i>true</i>	if <i>lhs</i> and <i>rhs</i> are not the same.
<i>false</i>	if <i>lhs</i> and <i>rhs</i> are the same.

Definition at line 282 of file [MAVSubnet.cpp](#).

15.22.4.2 operator<<

```
std::ostream & operator<< (
    std::ostream & os,
    const MAVSubnet & mavsubnet ) [friend]
```

Print the MAVLink subnet to the given output stream.

There are three string forms of MAVLink subnets.

1. "<System ID>.<Component ID>:<System ID mask>.<Component ID mask>"

2. "<System ID>.<Component ID>/<bits>"
3. "<System ID>.<Component ID>\<bits>"
4. "<System ID>.<Component ID>"

The slash notation is preferred. The last form is used when the mask requires all bits of a subnet to match an address.

See [MAVSubnet::MAVSubnet\(std::string address\)](#) for more information on the string format.

Note

The string constructor [MAVSubnet\(std::string\)](#) and the output stream operator are not inverses because of form preferences:

Parameters

<i>os</i>	The output stream to print to.
<i>mavsubnet</i>	The MAVLink subnet to print.

Returns

The output stream.

Definition at line 311 of file [MAVSubnet.cpp](#).

15.22.4.3 operator==

```
bool operator== (
    const MAVSubnet & lhs,
    const MAVSubnet & rhs ) [friend]
```

Equality comparison.

Compares both address and mask.

Parameters

<i>lhs</i>	The left hand side MAVLink subnet.
<i>rhs</i>	The right hand side MAVLink subnet.

Return values

<i>true</i>	if <i>lhs</i> and <i>rhs</i> are the same.
<i>false</i>	if <i>lhs</i> and <i>rhs</i> are not the same.

Definition at line 266 of file [MAVSubnet.cpp](#).

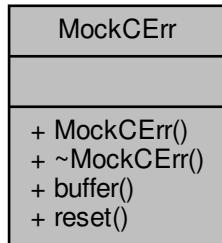
The documentation for this class was generated from the following files:

- [MAVSubnet.hpp](#)
- [MAVSubnet.cpp](#)

15.23 MockCErr Class Reference

```
#include <common.hpp>
```

Collaboration diagram for MockCErr:



Public Member Functions

- [MockCErr \(\)](#)
- [~MockCErr \(\)](#)
- std::string [buffer \(\)](#)
- void [reset \(\)](#)

15.23.1 Detailed Description

RAlI class to replace std::cerr with a mocked buffer.

Definition at line 94 of file [common.hpp](#).

15.23.2 Constructor & Destructor Documentation

15.23.2.1 MockCErr()

```
MockCErr::MockCErr ( ) [inline]
```

Replace std::cerr with this mock.

Definition at line 99 of file [common.hpp](#).

15.23.2.2 ~MockCErr()

```
MockCErr::~MockCErr ( ) [inline]
```

Restore std::cerr.

Definition at line 106 of file [common.hpp](#).

15.23.3 Member Function Documentation

15.23.3.1 buffer()

```
std::string MockCErr::buffer ( ) [inline]
```

Return the contents of the mocked std::cerr buffer as a string.

Returns

The contents of the mocked buffer.

Definition at line 114 of file [common.hpp](#).

15.23.3.2 reset()

```
void MockCErr::reset ( ) [inline]
```

Erase the mocked buffer.

Definition at line 120 of file [common.hpp](#).

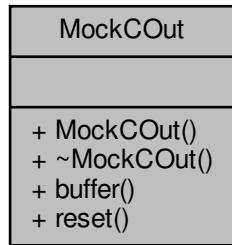
The documentation for this class was generated from the following file:

- [common.hpp](#)

15.24 MockCOut Class Reference

```
#include <common.hpp>
```

Collaboration diagram for MockCOut:



Public Member Functions

- [MockCOut \(\)](#)
- [~MockCOut \(\)](#)
- [std::string buffer \(\)](#)
- [void reset \(\)](#)

15.24.1 Detailed Description

RAII class to replace std::cout with a mocked buffer.

Definition at line [55](#) of file [common.hpp](#).

15.24.2 Constructor & Destructor Documentation

15.24.2.1 MockCOut()

```
MockCOut::MockCOut ( ) [inline]
```

Replace std::cout with this mock.

Definition at line [60](#) of file [common.hpp](#).

15.24.2.2 ~MockCOut()

```
MockCOut::~MockCOut ( ) [inline]
```

Restore std::cout.

Definition at line [67](#) of file [common.hpp](#).

15.24.3 Member Function Documentation

15.24.3.1 buffer()

```
std::string MockCOut::buffer ( ) [inline]
```

Return the contents of the mocked std::cout buffer as a string.

Returns

The contents of the mocked buffer.

Definition at line [75](#) of file [common.hpp](#).

15.24.3.2 reset()

```
void MockCOut::reset ( ) [inline]
```

Erase the mocked buffer.

Definition at line [81](#) of file [common.hpp](#).

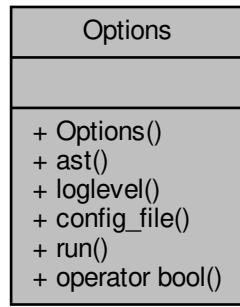
The documentation for this class was generated from the following file:

- [common.hpp](#)

15.25 Options Class Reference

```
#include <Options.hpp>
```

Collaboration diagram for Options:



Public Member Functions

- [Options \(int argc, const char *argv\[\], const **Filesystem** &filesystem=**Filesystem**\(\)\)](#)
- [bool **ast** \(\)](#)
- [unsigned int **loglevel** \(\)](#)
- [std::string **config_file** \(\)](#)
- [bool **run** \(\)](#)
- [operator bool \(\) const](#)

Related Functions

(Note that these are not member functions.)

- [std::optional< std::string > **find_config** \(const **Filesystem** &filesystem\)](#)

15.25.1 Detailed Description

An options class which is used to parse the command line arguments.

This class is what provides the command line help for mavtables.

```
usage: mavtables:
-h [ --help ]           print this message
--config arg            specify configuration file
--ast                   print AST of configuration file (do not run)
--version               print version and license information
--loglevel arg          level of logging, between 0 and 3
```

Definition at line 42 of file [Options.hpp](#).

15.25.2 Constructor & Destructor Documentation

15.25.2.1 Options()

```
Options::Options (
    int argc,
    const char * argv[],
    const Filesystem & filesystem = Filesystem() )
```

Construct an options object.

This will parse the command line arguments and construct an object for passing the result of these arguments to the application.

The first two arguments are designed to be taken directly from the inputs to the standard main function.

Parameters

<i>argc</i>	The number of command line arguments given.
<i>argv</i>	The command line arguments, as given in the arguments to the main function.
<i>filesystem</i>	A filesystem instance. The default is to construct an instance. This exists for testing purposes.

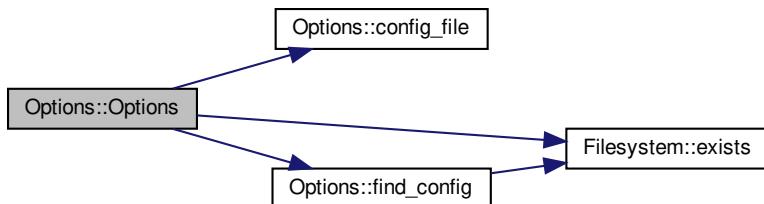
Exceptions

<i>std::runtime_error</i>	if no configuration file can be found.
---------------------------	--

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

References [config_file\(\)](#), [Filesystem::exists\(\)](#), and [find_config\(\)](#).

Here is the call graph for this function:



15.25.3 Member Function Documentation

15.25.3.1 ast()

```
bool Options::ast ( )
```

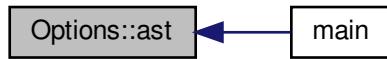
Determine whether to print the configuration file's AST or not.

Return values

<i>true</i>	Print abstract syntax tree of configuration file.
<i>false</i>	Don't print abstract syntax tree of configuration file.

Definition at line 135 of file [Options.cpp](#).

Here is the caller graph for this function:



15.25.3.2 config_file()

```
std::string Options::config_file ( )
```

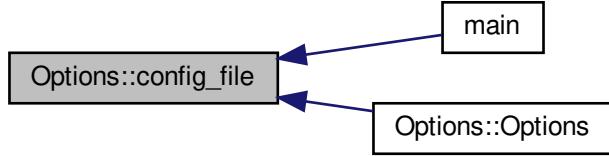
Get path to an existing configuration file.

Returns

The path to an existing, but not necessarily valid configuration file.

Definition at line 146 of file [Options.cpp](#).

Here is the caller graph for this function:



15.25.3.3 loglevel()

```
unsigned int Options::loglevel ( )
```

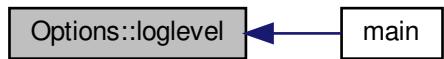
Get the log level.

Returns

The level to log at, between 0 and 3.

Definition at line 156 of file [Options.cpp](#).

Here is the caller graph for this function:



15.25.3.4 operator bool()

```
Options::operator bool ( ) const [explicit]
```

Determine if the [Options](#) object is valid.

Return values

<i>true</i>	If the options object successfully parsed the command line arguments.
<i>false</i>	If the program should exit immediately.

Definition at line 179 of file [Options.cpp](#).

15.25.3.5 run()

```
bool Options::run ( )
```

Determine whether to run the firewall/router or not.

Return values

<i>true</i>	Run the firewall/router.
<i>false</i>	Don't run the firewall/router.

Definition at line 167 of file [Options.cpp](#).

Here is the caller graph for this function:

**15.25.4 Friends And Related Function Documentation****15.25.4.1 find_config()**

```
std::optional< std::string > find_config (   
    const Filesystem & filesystem ) [related]
```

Find the configuration file.

Find the first configuration file in the list below:

1. The target of the MAVTABLES_CONFIG_PATH environment variable.
2. .mavtablesrc in the current directory.
3. .mavtablesrc at \$HOME/.mavtablesrc.
4. The main configuration file at PREFIX/etc/mavtables.conf.

Parameters

<i>filesystem</i>	A filesystem instance. The default is to construct an instance. This exists for testing purposes.
-------------------	---

Returns

The path to the first configuration file found. {} if no configuration file could be found.

Definition at line 199 of file [Options.cpp](#).

References [Filesystem::exists\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



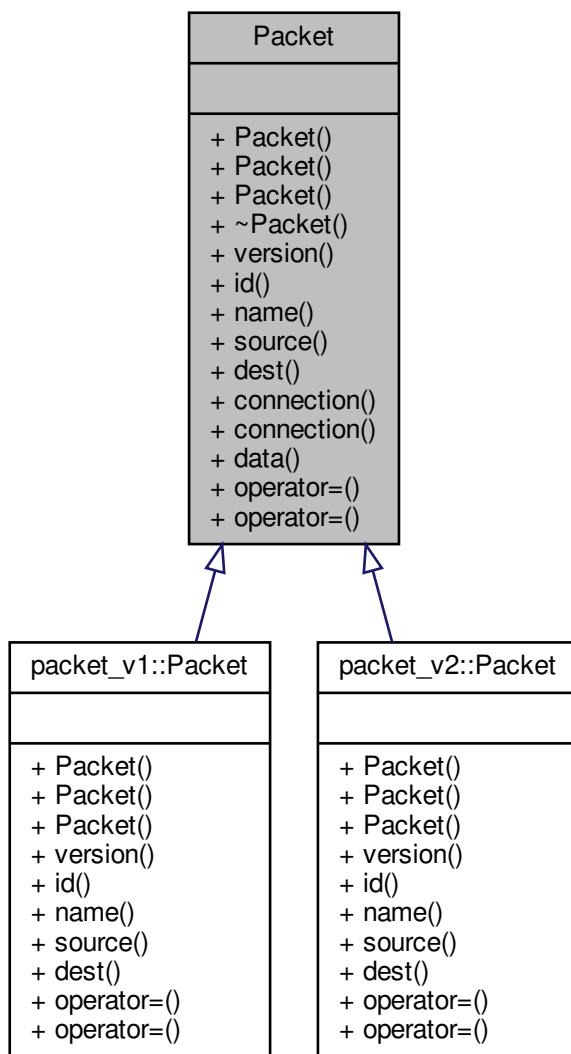
The documentation for this class was generated from the following files:

- [Options.hpp](#)
- [Options.cpp](#)

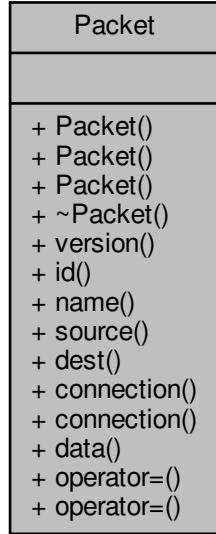
15.26 Packet Class Reference

```
#include <Packet.hpp>
```

Inheritance diagram for Packet:



Collaboration diagram for Packet:



Public Types

- enum `Version` { `V1` = 0x0100, `V2` = 0x0200 }

Public Member Functions

- `Packet (const Packet &other)=default`
- `Packet (Packet &&other)=default`
- `Packet (std::vector< uint8_t > data)`
- virtual `~Packet ()`
- virtual `Version version () const =0`
- virtual `unsigned long id () const =0`
- virtual `std::string name () const =0`
- virtual `MAVAddress source () const =0`
- virtual `std::optional< MAVAddress > dest () const =0`
- void `connection (std::weak_ptr< Connection > connection)`
- `const std::shared_ptr< Connection > connection () const`
- `const std::vector< uint8_t > & data () const`
- `Packet & operator= (const Packet &other)=default`
- `Packet & operator= (Packet &&other)=default`

Related Functions

(Note that these are not member functions.)

- bool `operator==` (const `Packet` &lhs, const `Packet` &rhs)
- bool `operator!=` (const `Packet` &lhs, const `Packet` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `Packet` &packet)

15.26.1 Detailed Description

A MAVLink packet.

This is an abstract class, it is meant to be overridden by classes implementing either version 1 or version 2 of the MAVLink packet wire protocol.

Definition at line 44 of file [Packet.hpp](#).

15.26.2 Member Enumeration Documentation

15.26.2.1 Version

```
enum Packet::Version
```

Enumerator

V1	MAVLink Version 1.0.
V2	MAVLink Version 2.0.

Definition at line 47 of file [Packet.hpp](#).

15.26.3 Constructor & Destructor Documentation

15.26.3.1 `Packet()` [1/3]

```
Packet::Packet (
```

```
    const Packet & other ) [default]
```

Copy constructor.

Parameters

<i>other</i>	Packet to copy from.
--------------	--------------------------------------

15.26.3.2 Packet() [2/3]

```
Packet::Packet (
    Packet && other )  [default]
```

Move constructor.

Parameters

<i>other</i>	Packet to move from.
--------------	--------------------------------------

15.26.3.3 Packet() [3/3]

```
Packet::Packet (
    std::vector< uint8_t > data )
```

Construct a packet.

Parameters

<i>data</i>	Raw packet data.
-------------	------------------

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

15.26.3.4 ~Packet()

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

Definition at line 42 of file [Packet.cpp](#).

15.26.4 Member Function Documentation

15.26.4.1 `connection()` [1/2]

```
void Packet::connection (
    std::weak_ptr< Connection > connection )
```

Set the source connection of the packet.

Parameters

<code>connection</code>	The source connection.
-------------------------	------------------------

See also

[connection\(\)](#)

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

References [connection\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.26.4.2 connection() [2/2]

```
const std::shared_ptr< Connection > Packet::connection ( ) const
```

Get the source connection of the packet.

Returns

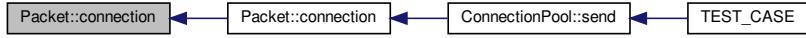
The source connection if set and it still exists, otherwise nullptr.

See also

[connection\(std::weak_ptr<Connection>\)](#)

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

Here is the caller graph for this function:



15.26.4.3 data()

```
const std::vector< uint8_t > & Packet::data ( ) const
```

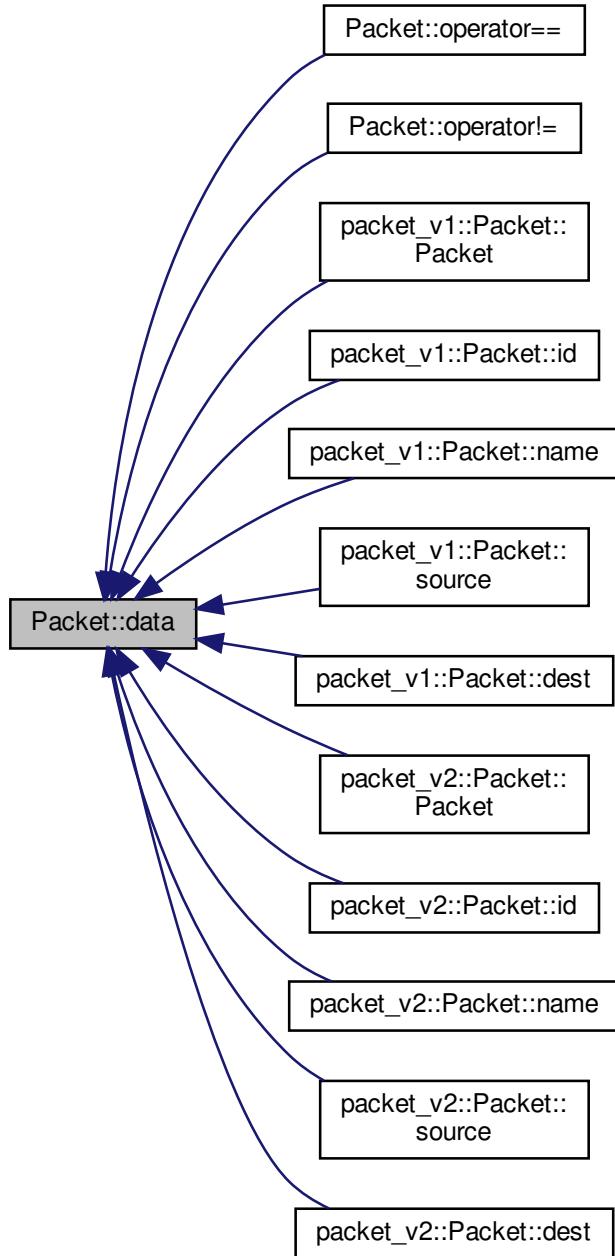
Return the packet data.

Returns

The packet data as a vector of bytes.

Definition at line [52](#) of file [Packet.cpp](#).

Here is the caller graph for this function:



15.26.4.4 dest()

```
virtual std::optional<MAVAddress> Packet::dest() const [pure virtual]
```

Return destination address.

Where the packet is sent to. This is optional because not all packets have a destination. If a system is specified, but not a component, a component ID of 0 will be used (the broadcast ID).

Returns

The destination MAVLink address of the packet if not a broadcast packet. If the packet does not have a destination address then {} will be returned.

Implemented in [packet_v2::Packet](#), and [packet_v1::Packet](#).

Here is the caller graph for this function:



15.26.4.5 id()

```
virtual unsigned long Packet::id() const [pure virtual]
```

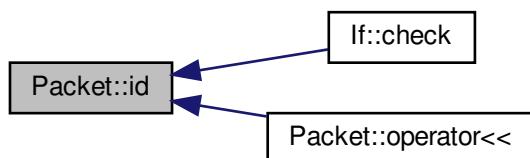
Return MAVLink message ID.

Returns

The numeric message ID of the MAVLink packet (0 to 255).

Implemented in [packet_v2::Packet](#), and [packet_v1::Packet](#).

Here is the caller graph for this function:



15.26.4.6 name()

```
virtual std::string Packet::name ( ) const [pure virtual]
```

Return MAVLink message name.

Returns

The message name of the MAVLink packet.

Implemented in [packet_v2::Packet](#), and [packet_v1::Packet](#).

Here is the caller graph for this function:



15.26.4.7 operator=() [1/2]

```
Packet& Packet::operator= (
    const Packet & other ) [default]
```

Assignment operator.

Parameters

<i>other</i>	Packet to copy from.
--------------	----------------------

15.26.4.8 operator=() [2/2]

```
Packet& Packet::operator= (
    Packet && other ) [default]
```

Assignment operator (by move semantics).

Parameters

<i>other</i>	Packet to move from.
--------------	--------------------------------------

15.26.4.9 source()

```
virtual MAVAddress Packet::source ( ) const [pure virtual]
```

Return source address.

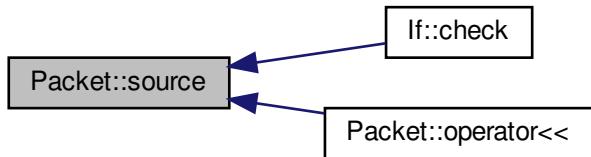
Where the packet came from.

Returns

The source MAVLink address of the packet.

Implemented in [packet_v2::Packet](#), and [packet_v1::Packet](#).

Here is the caller graph for this function:

**15.26.4.10 version()**

```
virtual Version Packet::version ( ) const [pure virtual]
```

Return packet version.

Returns

Two byte [Packet](#) version with major version in MSB and minor version in LSB.

Implemented in [packet_v2::Packet](#), and [packet_v1::Packet](#).

Here is the caller graph for this function:



15.26.5 Friends And Related Function Documentation

15.26.5.1 operator"!=()

```
bool operator!= (
    const Packet & lhs,
    const Packet & rhs ) [related]
```

Inequality comparison.

Compares the raw packet data.

Parameters

<i>lhs</i>	The left hand side packet.
<i>rhs</i>	The right hand side packet.

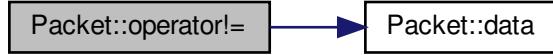
Return values

<i>true</i>	if <i>lhs</i> and <i>rhs</i> do not have the same packet data.
<i>false</i>	if <i>lhs</i> and <i>rhs</i> have the same packet data.

Definition at line 107 of file [Packet.cpp](#).

References [data\(\)](#).

Here is the call graph for this function:



15.26.5.2 operator<<()

```
std::ostream & operator<< (
    std::ostream & os,
    const Packet & packet ) [related]
```

Print the packet to the given output stream.

The format is "<Message Name> (#<Message ID>) from <Source Address> to <Destination Address> (v<Packet Version>)". However, "to <Destination Address>" is optional and will not be printed if the destination is the broadcast address 0.0.

Some examples are:

- HEARTBEAT (#1) from 16.8 (v1.0)
- PING (#4) from 128.4 to 16.8 (v2.0)
- DATA_TRANSMISSION_HANDSHAKE (#130) from 16.8 (v2.0)
- ENCAPSULATED_DATA (#131) from 128.4 (v2.0)

Parameters

<i>os</i>	The output stream to print to.
<i>packet</i>	The MAVLink packet to print.

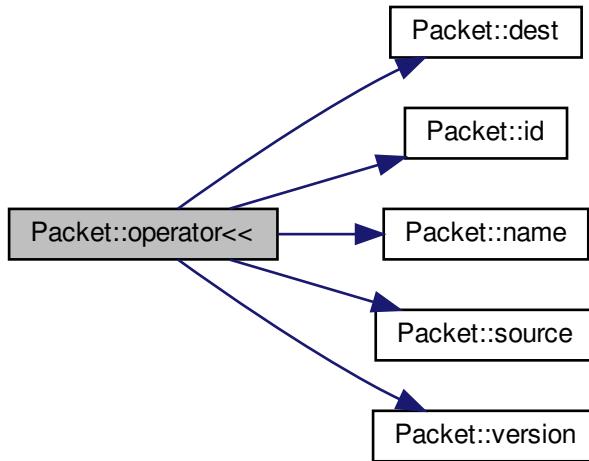
Returns

The output stream.

Definition at line 131 of file [Packet.cpp](#).

References [dest\(\)](#), [id\(\)](#), [name\(\)](#), [source\(\)](#), and [version\(\)](#).

Here is the call graph for this function:



15.26.5.3 operator==()

```
bool operator== (
    const Packet & lhs,
    const Packet & rhs ) [related]
```

Equality comparison.

Compares the raw packet data.

Parameters

<i>lhs</i>	The left hand side packet.
<i>rhs</i>	The right hand side packet.

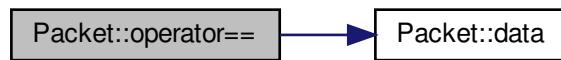
Return values

<i>true</i>	if <i>lhs</i> and <i>rhs</i> have the same packet data.
<i>false</i>	if <i>lhs</i> and <i>rhs</i> do not have the same packet data.

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

References [data\(\)](#).

Here is the call graph for this function:



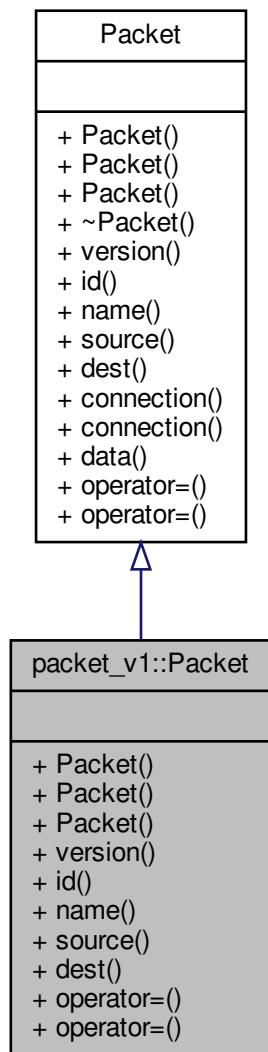
The documentation for this class was generated from the following files:

- [Packet.hpp](#)
- [Packet.cpp](#)

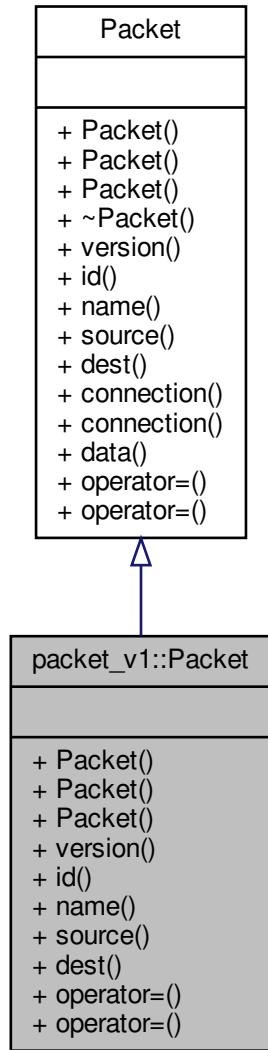
15.27 packet_v1::Packet Class Reference

```
#include <PacketVersion1.hpp>
```

Inheritance diagram for packet_v1::Packet:



Collaboration diagram for packet_v1::Packet:



Public Types

- enum `Version { V1 = 0x0100, V2 = 0x0200 }`

Public Member Functions

- `Packet (const Packet &other)=default`
- `Packet (Packet &&other)=default`

- `Packet (std::vector< uint8_t > data)`
- `virtual ::Packet::Version version () const`
- `virtual unsigned long id () const`
- `virtual std::string name () const`
- `virtual MAVAddress source () const`
- `virtual std::optional< MAVAddress > dest () const`
- `Packet & operator=(const Packet &other)=default`
- `Packet & operator=(Packet &&other)=default`
- `void connection (std::weak_ptr< Connection > connection)`
- `const std::shared_ptr< Connection > connection () const`
- `const std::vector< uint8_t > & data () const`

Related Functions

(Note that these are not member functions.)

- `bool header_complete (const std::vector< uint8_t > &data)`
- `bool packet_complete (const std::vector< uint8_t > &data)`
- `const struct mavlink::v1_header * header (const std::vector< uint8_t > &data)`
- `bool operator==(const Packet &lhs, const Packet &rhs)`
- `bool operator!=(const Packet &lhs, const Packet &rhs)`
- `std::ostream & operator<< (std::ostream &os, const Packet &packet)`

15.27.1 Detailed Description

A MAVLink packet with the version 1 wire protocol.

Definition at line 56 of file [PacketVersion1.hpp](#).

15.27.2 Member Enumeration Documentation

15.27.2.1 Version

```
enum Packet::Version [inherited]
```

Enumerator

V1	MAVLink Version 1.0.
V2	MAVLink Version 2.0.

Definition at line 47 of file [Packet.hpp](#).

15.27.3 Constructor & Destructor Documentation

15.27.3.1 `Packet()` [1/3]

```
packet_v1::Packet::Packet (
    const Packet & other ) [default]
```

Copy constructor.

Parameters

<i>other</i>	Packet to copy from.
--------------	--------------------------------------

15.27.3.2 `Packet()` [2/3]

```
packet_v1::Packet::Packet (
    Packet && other ) [default]
```

Move constructor.

Parameters

<i>other</i>	Packet to move from.
--------------	--------------------------------------

15.27.3.3 `Packet()` [3/3]

```
Packet::Packet (
    std::vector< uint8_t > data )
```

Construct a packet.

Parameters

<i>data</i>	Raw packet data.
-------------	------------------

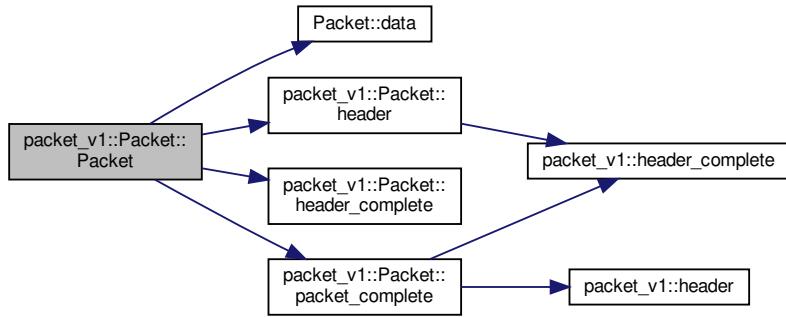
Exceptions

<code>std::invalid_argument</code>	if packet data does not start with the magic byte (0xFE).
<code>std::length_error</code>	if packet data is either too short or too long.

Definition at line 41 of file [PacketVersion1.cpp](#).

References `packet_v1::CHECKSUM_LENGTH`, `Packet::data()`, `header()`, `header_complete()`, `packet_v1::HEADER_LENGTH`, `mavlink::v1_header::len`, `mavlink::v1_header::msgid`, `packet_complete()`, and `packet_v1::START_BYTE`.

Here is the call graph for this function:



15.27.4 Member Function Documentation

15.27.4.1 connection() [1/2]

```
void Packet::connection (
    std::weak_ptr< Connection > connection ) [inherited]
```

Set the source connection of the packet.

Parameters

<code>connection</code>	The source connection.
-------------------------	------------------------

See also

[connection\(\)](#)

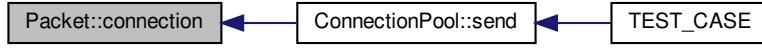
Definition at line 63 of file [Packet.cpp](#).

References [Packet::connection\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.27.4.2 connection() [2/2]

```
const std::shared_ptr< Connection > Packet::connection () const [inherited]
```

Get the source connection of the packet.

Returns

The source connection if set and it still exists, otherwise nullptr.

See also

[connection\(std::weak_ptr<Connection>\)](#)

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

Here is the caller graph for this function:



15.27.4.3 data()

```
const std::vector< uint8_t > & Packet::data ( ) const [inherited]
```

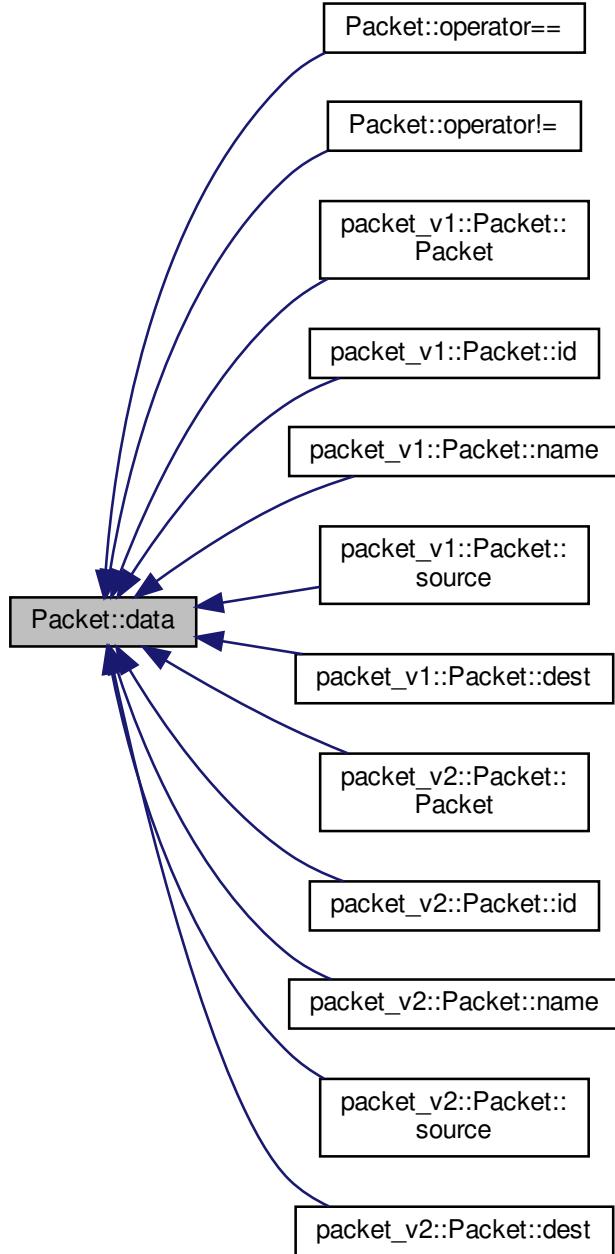
Return the packet data.

Returns

The packet data as a vector of bytes.

Definition at line 52 of file [Packet.cpp](#).

Here is the caller graph for this function:



15.27.4.4 dest()

```
std::optional< MAVAddress > Packet::dest ( ) const [virtual]
```

Return destination address.

Where the packet is sent to. This is optional because not all packets have a destination. If a system is specified, but not a component, a component ID of 0 will be used (the broadcast ID).

Returns

The destination MAVLink address of the packet if not a broadcast packet. If the packet does not have a destination address then {} will be returned.

Thanks

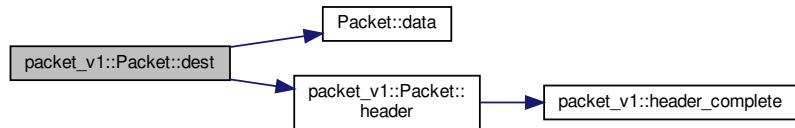
: The [mavlink-router](#) project for an example of how to extract the destination address.

Implements [Packet](#).

Definition at line 146 of file [PacketVersion1.cpp](#).

References [Packet::data\(\)](#), and [header\(\)](#).

Here is the call graph for this function:



15.27.4.5 id()

```
unsigned long Packet::id ( ) const [virtual]
```

Return MAVLink message ID.

Returns

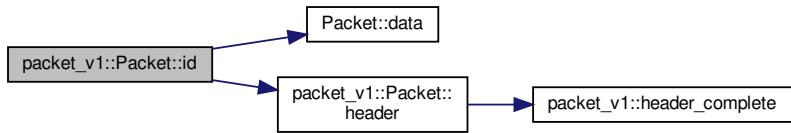
The numeric message ID of the MAVLink packet (0 to 255).

Implements [Packet](#).

Definition at line 108 of file [PacketVersion1.cpp](#).

References [Packet::data\(\)](#), [header\(\)](#), and [mavlink::v1_header::msgid](#).

Here is the call graph for this function:



15.27.4.6 name()

```
std::string Packet::name ( ) const [virtual]
```

Return MAVLink message name.

Returns

The message name of the MAVLink packet.

Exceptions

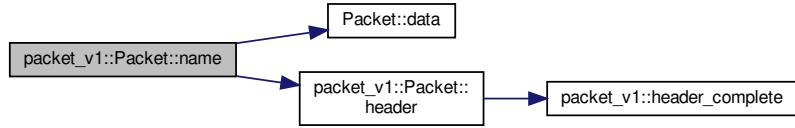
<code>std::runtime_error</code>	if the packet data has an invalid ID.
---------------------------------	---------------------------------------

Implements [Packet](#).

Definition at line 118 of file [PacketVersion1.cpp](#).

References [Packet::data\(\)](#), and [header\(\)](#).

Here is the call graph for this function:



15.27.4.7 operator=() [1/2]

```
Packet& packet_v1::Packet::operator= (
    const Packet & other ) [default]
```

Assignment operator.

Parameters

<i>other</i>	Packet to copy from.
--------------	----------------------

15.27.4.8 operator=() [2/2]

```
Packet& packet_v1::Packet::operator= (
    Packet && other ) [default]
```

Assignment operator (by move semantics).

Parameters

<i>other</i>	Packet to move from.
--------------	----------------------

15.27.4.9 source()

```
MAVAddress Packet::source ( ) const [virtual]
```

Return source address.

Where the packet came from.

Returns

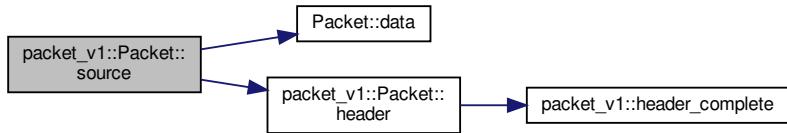
The source MAVLink address of the packet.

Implements [Packet](#).

Definition at line 135 of file [PacketVersion1.cpp](#).

References [Packet::data\(\)](#), and [header\(\)](#).

Here is the call graph for this function:

**15.27.4.10 version()**

```
Packet::Version Packet::version ( ) const [virtual]
```

Return packet version.

Returns

Two byte [Packet](#) version with major version in MSB and minor version in LSB.
0x0100 (v1.0) - [Packet::V1](#)

Implements [Packet](#).

Definition at line 102 of file [PacketVersion1.cpp](#).

References [packet_v1::VERSION](#).

15.27.5 Friends And Related Function Documentation**15.27.5.1 header()**

```
const struct mavlink::v1_header * header (
    const std::vector< uint8_t > & data ) [related]
```

Cast data as a v1.0 packet header structure pointer.

Parameters

<code>data</code>	The packet data.
-------------------	------------------

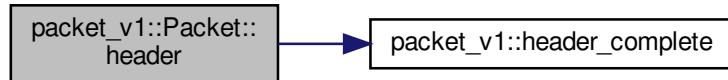
Returns

A pointer to the given data, cast to a v1.0 header structure. If an incomplete header is given a nullptr will be returned.

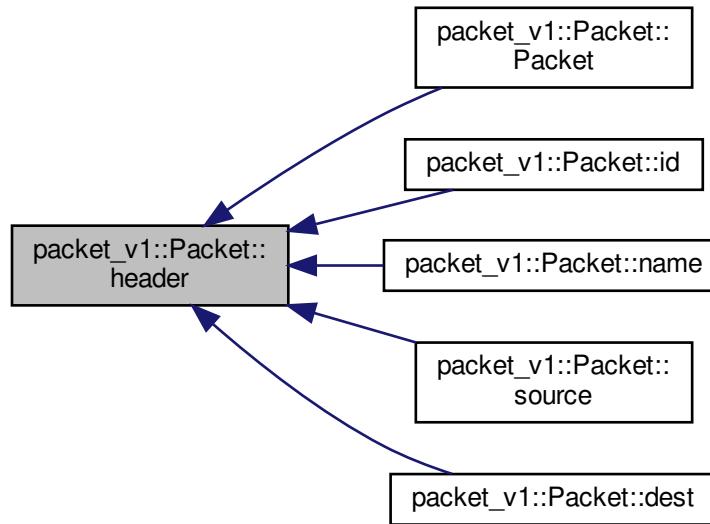
Definition at line 238 of file [PacketVersion1.cpp](#).

References [packet_v1::header_complete\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.27.5.2 header_complete()

```
bool header_complete (
    const std::vector< uint8_t > & data ) [related]
```

Determine if the given data contains a complete v1.0 header.

Parameters

<i>data</i>	The packet data.
-------------	------------------

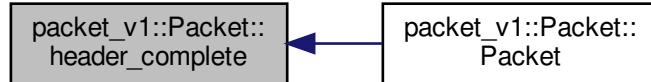
Return values

<i>true</i>	if <i>data</i> contains a complete header (starting with the magic byte).
<i>false</i>	if <i>data</i> does not contain a complete v1.0 header.

Definition at line 200 of file [PacketVersion1.cpp](#).

References [packet_v1::HEADER_LENGTH](#), and [packet_v1::START_BYTE](#).

Here is the caller graph for this function:



15.27.5.3 operator"!=()

```
bool operator!= (
    const Packet & lhs,
    const Packet & rhs ) [related]
```

Inequality comparison.

Compares the raw packet data.

Parameters

<i>lhs</i>	The left hand side packet.
<i>rhs</i>	The right hand side packet.

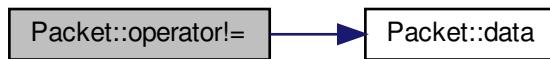
Return values

<i>true</i>	if lhs and rhs do not have the same packet data.
<i>false</i>	if lhs and rhs have the same packet data.

Definition at line 107 of file [Packet.cpp](#).

References [Packet::data\(\)](#).

Here is the call graph for this function:

**15.27.5.4 operator<<()**

```
std::ostream & operator<< (
    std::ostream & os,
    const Packet & packet ) [related]
```

Print the packet to the given output stream.

The format is "<Message Name> (#<Message ID>) from <Source Address> to <Destination Address> (v<Packet Version>)". However, "to <Destination Address>" is optional and will not be printed if the destination is the broadcast address 0.0.

Some examples are:

- HEARTBEAT (#1) from 16.8 (v1.0)
- PING (#4) from 128.4 to 16.8 (v2.0)
- DATA_TRANSMISSION_HANDSHAKE (#130) from 16.8 (v2.0)
- ENCAPSULATED_DATA (#131) from 128.4 (v2.0)

Parameters

<i>os</i>	The output stream to print to.
<i>packet</i>	The MAVLink packet to print.

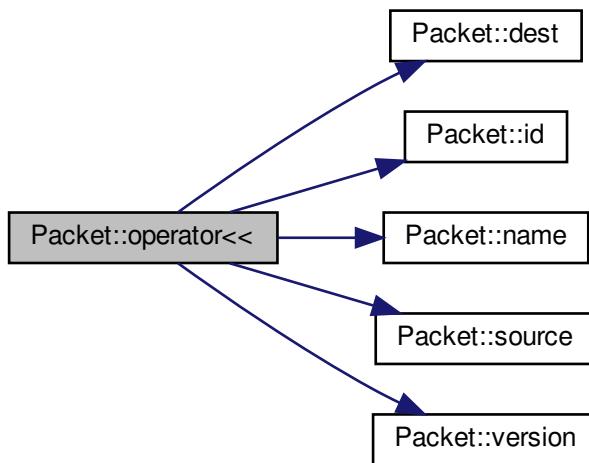
Returns

The output stream.

Definition at line 131 of file [Packet.cpp](#).

References [Packet::dest\(\)](#), [Packet::id\(\)](#), [Packet::name\(\)](#), [Packet::source\(\)](#), and [Packet::version\(\)](#).

Here is the call graph for this function:

**15.27.5.5 operator==()**

```
bool operator== (
    const Packet & lhs,
    const Packet & rhs ) [related]
```

Equality comparison.

Compares the raw packet data.

Parameters

<i>lhs</i>	The left hand side packet.
<i>rhs</i>	The right hand side packet.

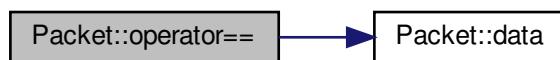
Return values

<i>true</i>	if lhs and rhs have the same packet data.
<i>false</i>	if lhs and rhs do not have the same packet data.

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

References [Packet::data\(\)](#).

Here is the call graph for this function:

**15.27.5.6 packet_complete()**

```
bool packet_complete (
    const std::vector< uint8_t > & data ) [related]
```

Determine if the given data contains a complete v1.0 packet.

Parameters

<i>data</i>	The packet data.
-------------	------------------

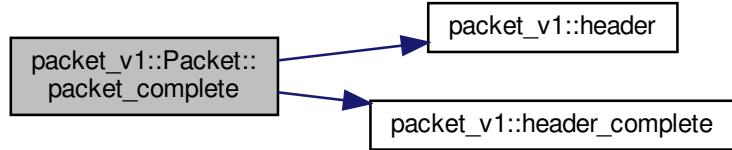
Return values

<i>true</i>	if data contains a complete packet (starting with the magic byte).
<i>false</i>	if data does not contain a complete v1.0 packet, or if there is extra bytes in data beyond the packet.

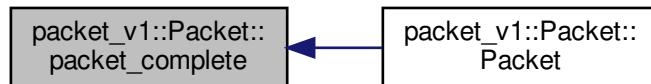
Definition at line 218 of file [PacketVersion1.cpp](#).

References [packet_v1::CHECKSUM_LENGTH](#), [packet_v1::header\(\)](#), [packet_v1::header_complete\(\)](#), [packet_v1::HEADER_LENGTH](#), and [mavlink::v1_header::len](#).

Here is the call graph for this function:



Here is the caller graph for this function:



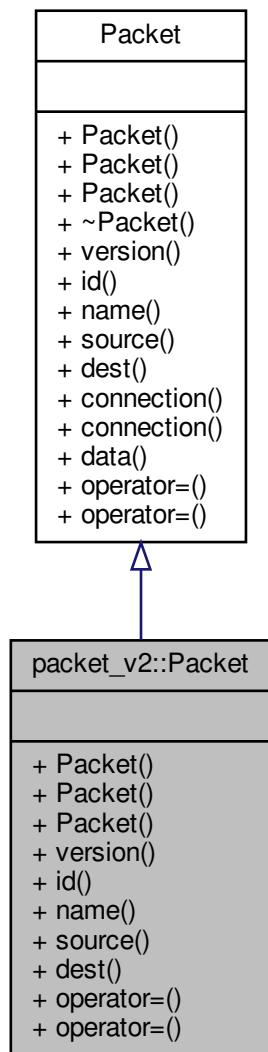
The documentation for this class was generated from the following files:

- [PacketVersion1.hpp](#)
- [PacketVersion1.cpp](#)

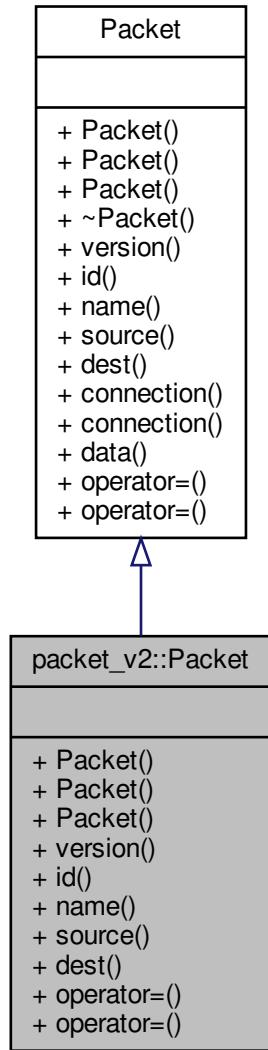
15.28 packet_v2::Packet Class Reference

```
#include <PacketVersion2.hpp>
```

Inheritance diagram for packet_v2::Packet:



Collaboration diagram for packet_v2::Packet:



Public Types

- enum `Version` { `V1` = 0x0100, `V2` = 0x0200 }

Public Member Functions

- `Packet (const Packet &other)=default`
- `Packet (Packet &&other)=default`

- `Packet (std::vector< uint8_t > data)`
- `virtual ::Packet::Version version () const`
- `virtual unsigned long id () const`
- `virtual std::string name () const`
- `virtual MAVAddress source () const`
- `virtual std::optional< MAVAddress > dest () const`
- `Packet & operator= (const Packet &other)=default`
- `Packet & operator= (Packet &&other)=default`
- `void connection (std::weak_ptr< Connection > connection)`
- `const std::shared_ptr< Connection > connection () const`
- `const std::vector< uint8_t > & data () const`

Related Functions

(Note that these are not member functions.)

- `bool is_signed (const std::vector< uint8_t > &data)`
- `bool header_complete (const std::vector< uint8_t > &data)`
- `bool packet_complete (const std::vector< uint8_t > &data)`
- `const struct mavlink::v2_header * header (const std::vector< uint8_t > &data)`
- `bool operator== (const Packet &lhs, const Packet &rhs)`
- `bool operator!= (const Packet &lhs, const Packet &rhs)`
- `std::ostream & operator<< (std::ostream &os, const Packet &packet)`

15.28.1 Detailed Description

A MAVLink packet with the version 2 wire protocol.

Definition at line 61 of file [PacketVersion2.hpp](#).

15.28.2 Member Enumeration Documentation

15.28.2.1 Version

```
enum Packet::Version [inherited]
```

Enumerator

V1	MAVLink Version 1.0.
V2	MAVLink Version 2.0.

Definition at line 47 of file [Packet.hpp](#).

15.28.3 Constructor & Destructor Documentation

15.28.3.1 `Packet()` [1/3]

```
packet_v2::Packet::Packet (
    const Packet & other ) [default]
```

Copy constructor.

Parameters

<i>other</i>	Packet to copy from.
--------------	--------------------------------------

15.28.3.2 `Packet()` [2/3]

```
packet_v2::Packet::Packet (
    Packet && other ) [default]
```

Move constructor.

Parameters

<i>other</i>	Packet to move from.
--------------	--------------------------------------

15.28.3.3 `Packet()` [3/3]

```
Packet::Packet (
    std::vector< uint8_t > data )
```

Construct a packet.

Parameters

<i>data</i>	Raw packet data.
-------------	------------------

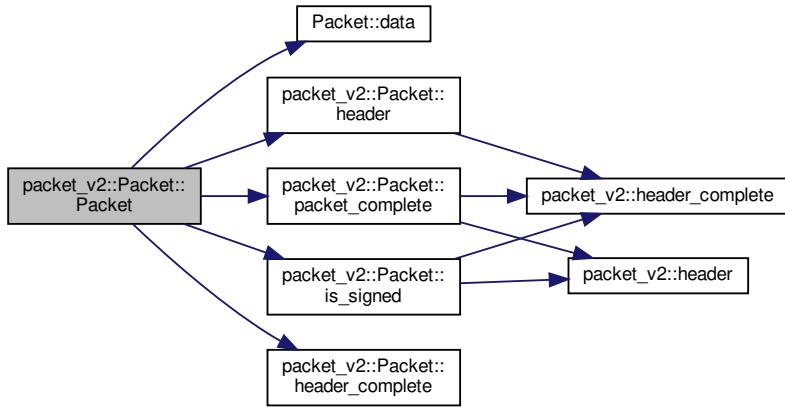
Exceptions

<code>std::invalid_argument</code>	if packet data does not start with the magic byte (0xFD).
<code>std::length_error</code>	if packet data is either too short or too long.

Definition at line 41 of file [PacketVersion2.cpp](#).

References `packet_v2::CHECKSUM_LENGTH`, `Packet::data()`, `header()`, `header_complete()`, `packet_v2::HEADER_LENGTH`, `is_signed()`, `mavlink::v2_header::len`, `mavlink::v2_header::msgid`, `packet_complete()`, `packet_v2::SIGNATURE_LENGTH`, and `packet_v2::START_BYTE`.

Here is the call graph for this function:



15.28.4 Member Function Documentation

15.28.4.1 connection() [1/2]

```
void Packet::connection (
    std::weak_ptr< Connection > connection ) [inherited]
```

Set the source connection of the packet.

Parameters

<code>connection</code>	The source connection.
-------------------------	------------------------

See also[connection\(\)](#)

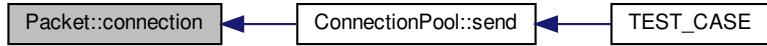
Definition at line 63 of file [Packet.cpp](#).

References [Packet::connection\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.28.4.2 connection() [2/2]

```
const std::shared_ptr< Connection > Packet::connection( ) const [inherited]
```

Get the source connection of the packet.

Returns

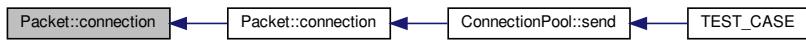
The source connection if set and it still exists, otherwise nullptr.

See also

[connection\(std::weak_ptr<Connection>\)](#)

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

Here is the caller graph for this function:



15.28.4.3 data()

```
const std::vector< uint8_t > & Packet::data( ) const [inherited]
```

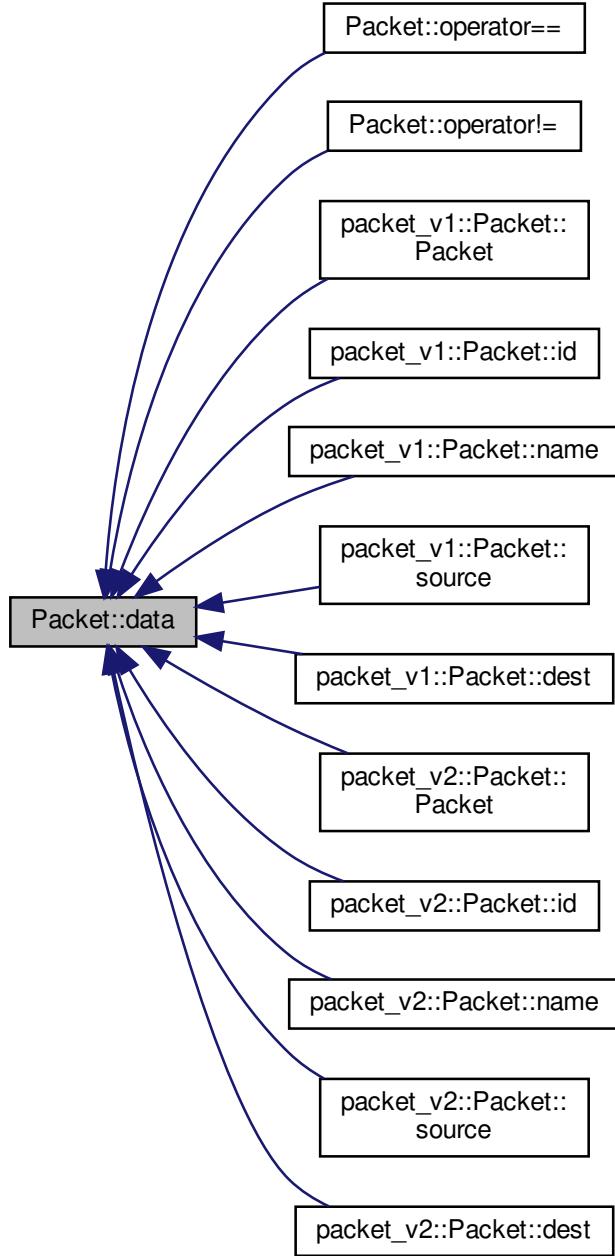
Return the packet data.

Returns

The packet data as a vector of bytes.

Definition at line [52](#) of file [Packet.cpp](#).

Here is the caller graph for this function:



15.28.4.4 dest()

```
std::optional< MAVAddress > Packet::dest ( ) const [virtual]
```

Return destination address.

Where the packet is sent to. This is optional because not all packets have a destination. If a system is specified, but not a component, a component ID of 0 will be used (the broadcast ID).

Returns

The destination MAVLink address of the packet if not a broadcast packet. If the packet does not have a destination address then {} will be returned.

Thanks

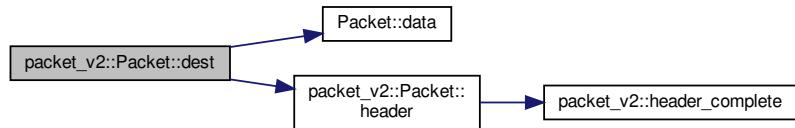
: The [mavlink-router](#) project for an example of how to extract the destination address.

Implements [Packet](#).

Definition at line 155 of file [PacketVersion2.cpp](#).

References [Packet::data\(\)](#), and [header\(\)](#).

Here is the call graph for this function:



15.28.4.5 id()

```
unsigned long Packet::id ( ) const [virtual]
```

Return MAVLink message ID.

Returns

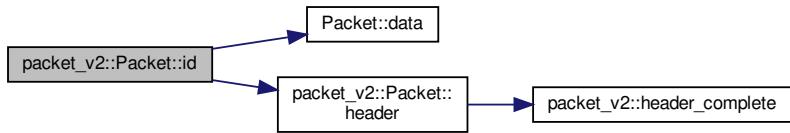
The numeric message ID of the MAVLink packet (0 to 255).

Implements [Packet](#).

Definition at line 117 of file [PacketVersion2.cpp](#).

References [Packet::data\(\)](#), [header\(\)](#), and [mavlink::v2_header::msgid](#).

Here is the call graph for this function:



15.28.4.6 name()

```
std::string Packet::name ( ) const [virtual]
```

Return MAVLink message name.

Returns

The message name of the MAVLink packet.

Exceptions

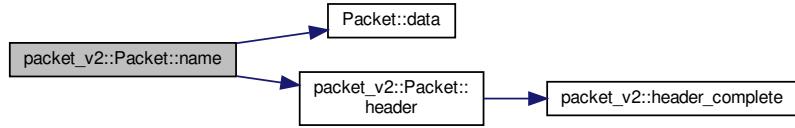
<code>std::runtime_error</code>	If the packet data has an invalid ID.
---------------------------------	---------------------------------------

Implements [Packet](#).

Definition at line 127 of file [PacketVersion2.cpp](#).

References [Packet::data\(\)](#), and [header\(\)](#).

Here is the call graph for this function:



15.28.4.7 operator=() [1/2]

```
Packet& packet_v2::Packet::operator= (
    const Packet & other ) [default]
```

Assignment operator.

Parameters

<i>other</i>	Packet to copy from.
--------------	----------------------

15.28.4.8 operator=() [2/2]

```
Packet& packet_v2::Packet::operator= (
    Packet && other ) [default]
```

Assignment operator (by move semantics).

Parameters

<i>other</i>	Packet to move from.
--------------	----------------------

15.28.4.9 source()

```
MAVAddress Packet::source ( ) const [virtual]
```

Return source address.

Where the packet came from.

Returns

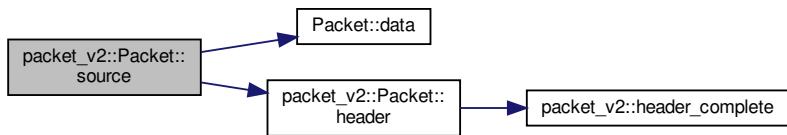
The source MAVLink address of the packet.

Implements [Packet](#).

Definition at line 144 of file [PacketVersion2.cpp](#).

References [Packet::data\(\)](#), and [header\(\)](#).

Here is the call graph for this function:

**15.28.4.10 version()**

```
Packet::Version Packet::version ( ) const [virtual]
```

Return packet version.

Returns

Two byte [Packet](#) version with major version in MSB and minor version in LSB.
0x0200 (v2.0) - [Packet::V2](#)

Implements [Packet](#).

Definition at line 111 of file [PacketVersion2.cpp](#).

15.28.5 Friends And Related Function Documentation**15.28.5.1 header()**

```
const struct mavlink::v2_header * header (
    const std::vector< uint8_t > & data ) [related]
```

Cast data as a v2.0 packet header structure pointer.

Parameters

<code>data</code>	The packet data.
-------------------	------------------

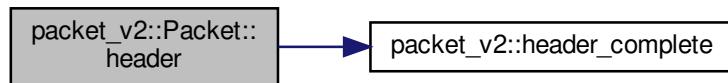
Returns

A pointer to the given data, cast to a v2.0 header structure. If an incomplete header is given a nullptr will be returned.

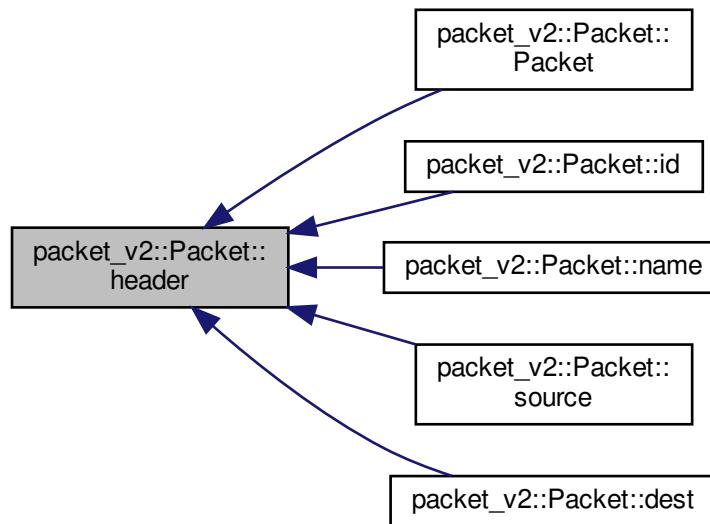
Definition at line 289 of file [PacketVersion2.cpp](#).

References [packet_v2::header_complete\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.28.5.2 header_complete()

```
bool header_complete (
    const std::vector< uint8_t > & data ) [related]
```

Determine if the given data contains a complete v2.0 header.

Parameters

<i>data</i>	The packet data.
-------------	------------------

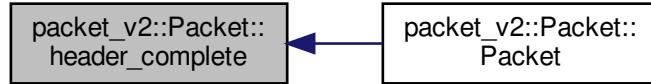
Return values

<i>true</i>	if <i>data</i> contains a complete header (starting with the magic byte).
<i>false</i>	if <i>data</i> does not contain a complete v2.0 header.

Definition at line 247 of file [PacketVersion2.cpp](#).

References [packet_v2::HEADER_LENGTH](#), and [packet_v2::START_BYTE](#).

Here is the caller graph for this function:



15.28.5.3 is_signed()

```
bool is_signed (
    const std::vector< uint8_t > & data ) [related]
```

Determine if a MAVLink v2.0 packet is signed or not.

Parameters

<i>data</i>	The packet data.
-------------	------------------

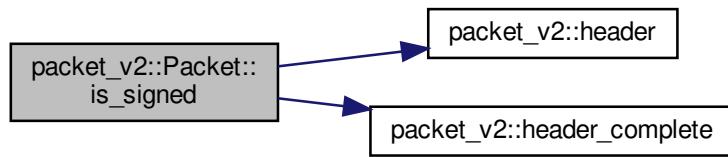
Exceptions

<code>std::invalid_argument</code>	Header is not complete or is invalid.
------------------------------------	---------------------------------------

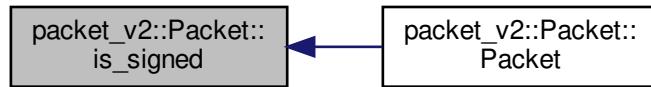
Definition at line 227 of file [PacketVersion2.cpp](#).

References [packet_v2::header\(\)](#), and [packet_v2::header_complete\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.28.5.4 operator"!=()

```
bool operator!= (const Packet & lhs, const Packet & rhs ) [related]
```

Inequality comparison.

Compares the raw packet data.

Parameters

<i>lhs</i>	The left hand side packet.
<i>rhs</i>	The right hand side packet.

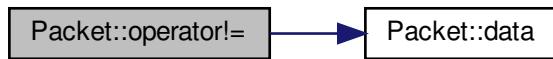
Return values

<i>true</i>	if <i>lhs</i> and <i>rhs</i> do not have the same packet data.
<i>false</i>	if <i>lhs</i> and <i>rhs</i> have the same packet data.

Definition at line 107 of file [Packet.cpp](#).

References [Packet::data\(\)](#).

Here is the call graph for this function:

**15.28.5.5 operator<<()**

```
std::ostream & operator<< (
    std::ostream & os,
    const Packet & packet ) [related]
```

Print the packet to the given output stream.

The format is "<Message Name> (#<Message ID>) from <Source Address> to <Destination Address> (v<Packet Version>)". However, "to <Destination Address>" is optional and will not be printed if the destination is the broadcast address 0.0.

Some examples are:

- HEARTBEAT (#1) from 16.8 (v1.0)
- PING (#4) from 128.4 to 16.8 (v2.0)
- DATA_TRANSMISSION_HANDSHAKE (#130) from 16.8 (v2.0)
- ENCAPSULATED_DATA (#131) from 128.4 (v2.0)

Parameters

<i>os</i>	The output stream to print to.
<i>packet</i>	The MAVLink packet to print.

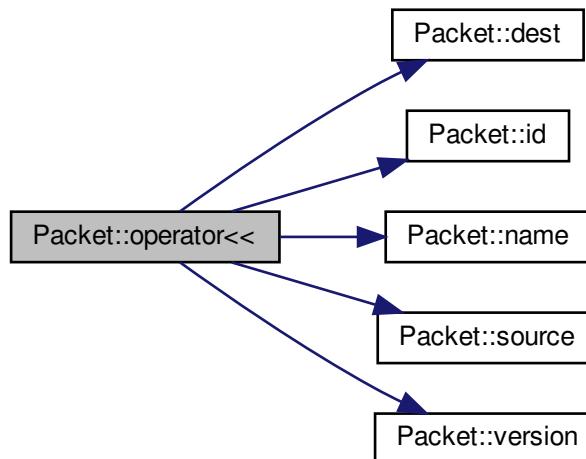
Returns

The output stream.

Definition at line 131 of file [Packet.cpp](#).

References [Packet::dest\(\)](#), [Packet::id\(\)](#), [Packet::name\(\)](#), [Packet::source\(\)](#), and [Packet::version\(\)](#).

Here is the call graph for this function:

**15.28.5.6 operator==()**

```
bool operator== (
    const Packet & lhs,
    const Packet & rhs ) [related]
```

Equality comparison.

Compares the raw packet data.

Parameters

<i>lhs</i>	The left hand side packet.
<i>rhs</i>	The right hand side packet.

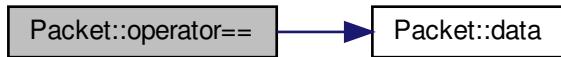
Return values

<i>true</i>	if <i>lhs</i> and <i>rhs</i> have the same packet data.
<i>false</i>	if <i>lhs</i> and <i>rhs</i> do not have the same packet data.

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

References [Packet::data\(\)](#).

Here is the call graph for this function:

**15.28.5.7 packet_complete()**

```
bool packet_complete (
    const std::vector< uint8_t > & data ) [related]
```

Determine if the given data contains a complete v2.0 packet.

Parameters

<i>data</i>	The packet data.
-------------	------------------

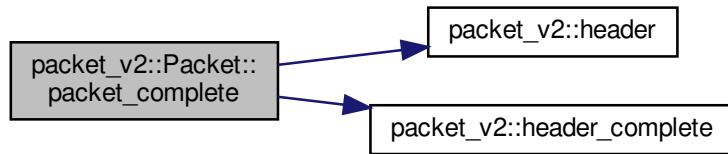
Return values

<i>true</i>	if <i>data</i> contains a complete packet (starting with the magic byte).
<i>false</i>	if <i>data</i> does not contain a complete v1.0 packet, or if there is extra bytes in <i>data</i> beyond the packet.

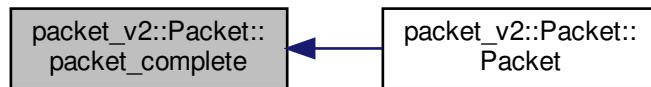
Definition at line 263 of file [PacketVersion2.cpp](#).

References [packet_v2::CHECKSUM_LENGTH](#), [packet_v2::header\(\)](#), [packet_v2::header_complete\(\)](#), [packet_v2::HEADER_LENGTH](#), [mavlink::v2_header::len](#), and [packet_v2::SIGNATURE_LENGTH](#).

Here is the call graph for this function:



Here is the caller graph for this function:



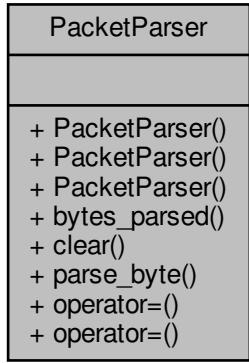
The documentation for this class was generated from the following files:

- [PacketVersion2.hpp](#)
- [PacketVersion2.cpp](#)

15.29 PacketParser Class Reference

```
#include <PacketParser.hpp>
```

Collaboration diagram for PacketParser:



Public Member Functions

- `PacketParser ()`
- `PacketParser (const PacketParser &other)=delete`
- `PacketParser (PacketParser &&other)=delete`
- `size_t bytes_parsed () const`
- `void clear ()`
- `std::unique_ptr<Packet> parse_byte (uint8_t byte)`
- `PacketParser & operator= (const PacketParser &other)=delete`
- `PacketParser & operator= (PacketParser &&other)=delete`

15.29.1 Detailed Description

A MAVLink packet parser.

Parses wire protocol bytes into a MAVLink [Packet](#).

Definition at line 33 of file [PacketParser.hpp](#).

15.29.2 Constructor & Destructor Documentation

15.29.2.1 PacketParser() [1/3]

```
PacketParser::PacketParser ( )
```

Construct a [PacketParser](#).

Definition at line 32 of file [PacketParser.cpp](#).

References [clear\(\)](#).

Here is the call graph for this function:



15.29.2.2 PacketParser() [2/3]

```
PacketParser::PacketParser (
    const PacketParser & other ) [delete]
```

15.29.2.3 PacketParser() [3/3]

```
PacketParser::PacketParser (
    PacketParser && other ) [delete]
```

15.29.3 Member Function Documentation

15.29.3.1 bytes_parsed()

```
size_t PacketParser::bytes_parsed ( ) const
```

Return the number of bytes parsed on the current packet.

Returns

The number of bytes parsed on the current packet, 0 if no packet is currently being parsed.

Definition at line 44 of file [PacketParser.cpp](#).

15.29.3.2 clear()

```
void PacketParser::clear ( )
```

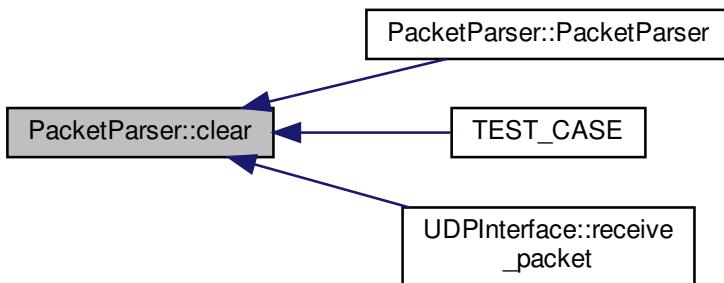
Reset packet parser so it can parse another packet.

If called while parsing a packet, that packet will be lost.

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

References [Packet::V2](#).

Here is the caller graph for this function:



15.29.3.3 operator=() [1/2]

```
PacketParser& PacketParser::operator= (
    const PacketParser & other ) [delete]
```

15.29.3.4 operator=() [2/2]

```
PacketParser& PacketParser::operator= (
    PacketParser && other ) [delete]
```

15.29.3.5 parse_byte()

```
std::unique_ptr< Packet > PacketParser::parse_byte (
    uint8_t byte )
```

Parse a MAVLink wire protocol byte, v1.0 or v2.0.

When a packet is completed it will be returned and the parser reset so it can be used to continue parsing.

Parameters

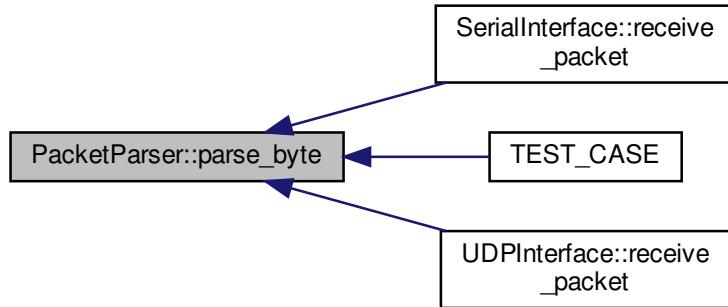
<i>byte</i>	A byte from the MAVLink wire protocol.
-------------	--

Returns

A complete v1.0 or v2.0 packet. If the parser has not yet parsed a complete packet, nullptr is returned.

Definition at line 73 of file [PacketParser.cpp](#).

Here is the caller graph for this function:



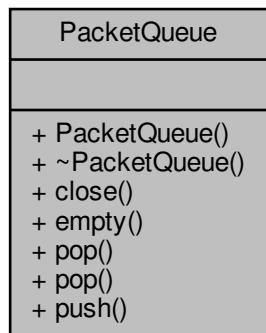
The documentation for this class was generated from the following files:

- [PacketParser.hpp](#)
- [PacketParser.cpp](#)

15.30 PacketQueue Class Reference

```
#include <PacketQueue.hpp>
```

Collaboration diagram for `PacketQueue`:



Public Member Functions

- [PacketQueue \(std::optional< std::function< void\(void\)>> callback={}\)](#)
- TEST_VIRTUAL ~[PacketQueue \(\)](#)=default
- TEST_VIRTUAL void [close \(\)](#)
- TEST_VIRTUAL bool [empty \(\)](#)
- TEST_VIRTUAL std::shared_ptr< const [Packet](#) > [pop \(\)](#)
- TEST_VIRTUAL std::shared_ptr< const [Packet](#) > [pop \(const std::chrono::nanoseconds &timeout\)](#)
- TEST_VIRTUAL void [push \(std::shared_ptr< const \[Packet\]\(#\) > packet, int priority=0\)](#)

15.30.1 Detailed Description

A threadsafe priority queue for MAVLink packets.

This priority queue will order packets based on priority but also maintains insertion order among packets of the same priority.

This is used to implement the packet priority of the firewall and to provide a queueing mechanism for packets when consumers are slower than the producers.

See also

[QueuedPacket](#)

Definition at line 45 of file [PacketQueue.hpp](#).

15.30.2 Constructor & Destructor Documentation

15.30.2.1 PacketQueue()

```
PacketQueue::PacketQueue (
    std::optional< std::function< void(void)>> callback = {} )
```

Construct a packet queue.

Parameters

<code>callback</code>	A function to call whenever a new packet is added to the queue. This allows the queue to signal when it has become non empty. The default is no callback {}.
-----------------------	--

Definition at line 58 of file [PacketQueue.cpp](#).

15.30.2.2 ~PacketQueue()

```
TEST_VIRTUAL PacketQueue::~PacketQueue ( ) [default]
```

15.30.3 Member Function Documentation

15.30.3.1 close()

```
void PacketQueue::close ( )
```

Close the queue.

This will release any blocking calls to [pop](#).

Remarks

Threadssafe (locking).

Definition at line [70](#) of file [PacketQueue.cpp](#).

15.30.3.2 empty()

```
bool PacketQueue::empty ( )
```

Determine if the packet queue is empty or not.

retval true There are no packets in the queue. retval false There is at least one packet in the queue.

Remarks

Threadssafe (locking).

Definition at line [87](#) of file [PacketQueue.cpp](#).

15.30.3.3 pop() [1/2]

```
std::shared_ptr< const Packet > PacketQueue::pop ( )
```

Remove and return the packet at the front of the queue.

This version will block on an empty queue and will not return until the queue becomes non empty or is closed with [close](#).

Returns

The packet that was at the front of the queue, or nullptr if the queue was closed.

Remarks

Threadssafe (locking).

See also

[pop\(const std::chrono::nanoseconds &\)](#)

Definition at line 105 of file [PacketQueue.cpp](#).

15.30.3.4 pop() [2/2]

```
std::shared_ptr< const Packet > PacketQueue::pop (
    const std::chrono::nanoseconds & timeout )
```

Remove and return the packet at the front of the queue.

This version will block on an empty queue and will not return until the queue becomes non empty, is closed with [close](#), or the `timeout` has expired.

Parameters

<code>timeout</code>	How long to block waiting for an empty queue. Set to 0s for non blocking.
----------------------	---

Returns

The packet that was at the front of the queue, or nullptr if the queue was closed or the timeout expired.

Remarks

Threadssafe (locking).

See also[pop\(\)](#)

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

15.30.3.5 push()

```
void PacketQueue::push (
    std::shared_ptr< const Packet > packet,
    int priority = 0 )
```

Add a new packet to the queue, with a priority.

A higher `priority` will result in the `packet` being pushed to the front of the queue. When priorities are equal the order in which the packets were added to the queue is maintained.

Parameters

<code>packet</code>	The packet to add to the queue. It must not be <code>nullptr</code> .
<code>priority</code>	The priority to use when adding it to the queue. The default is 0.

Exceptions

`std::invalid_argument` if the packet pointer is null.

Remarks

Threadsafe (locking).

Definition at line 164 of file [PacketQueue.cpp](#).

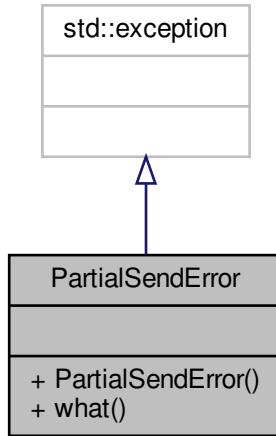
The documentation for this class was generated from the following files:

- [PacketQueue.hpp](#)
- [PacketQueue.cpp](#)

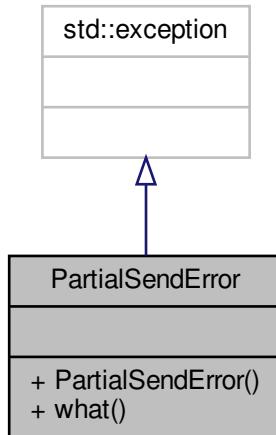
15.31 PartialSendError Class Reference

```
#include <PartialSendError.hpp>
```

Inheritance diagram for PartialSendError:



Collaboration diagram for PartialSendError:



Public Member Functions

- `PartialSendError` (`unsigned long bytes_sent, unsigned long total_bytes`)
- `const char * what () const noexcept`

15.31.1 Detailed Description

Exception type emitted when an interface fails to send a complete packet.

Definition at line 28 of file [PartialSendError.hpp](#).

15.31.2 Constructor & Destructor Documentation

15.31.2.1 PartialSendError()

```
PartialSendError::PartialSendError (
    unsigned long bytes_sent,
    unsigned long total_bytes )
```

Construct a [PartialSendError](#).

Parameters

<i>bytes_sent</i>	Number of bytes that were sent.
<i>total_bytes</i>	Total number of bytes in the packet.

Definition at line 29 of file [PartialSendError.cpp](#).

15.31.3 Member Function Documentation

15.31.3.1 what()

```
const char * PartialSendError::what ( ) const [noexcept]
```

Return error message string.

Returns

Error message string containing sent to total bytes ratio.

Definition at line 41 of file [PartialSendError.cpp](#).

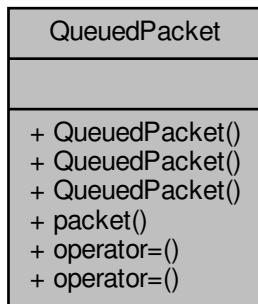
The documentation for this class was generated from the following files:

- [PartialSendError.hpp](#)
- [PartialSendError.cpp](#)

15.32 QueuedPacket Class Reference

```
#include <QueuedPacket.hpp>
```

Collaboration diagram for QueuedPacket:



Public Member Functions

- `QueuedPacket (const QueuedPacket &other)=default`
- `QueuedPacket (QueuedPacket &&other)=default`
- `QueuedPacket (std::shared_ptr< const Packet > packet, int priority, unsigned long long ticket_number)`
- `std::shared_ptr< const Packet > packet () const`
- `QueuedPacket & operator= (const QueuedPacket &other)=default`
- `QueuedPacket & operator= (QueuedPacket &&other)=default`

Friends

- `bool operator== (const QueuedPacket &lhs, const QueuedPacket &rhs)`
- `bool operator!= (const QueuedPacket &lhs, const QueuedPacket &rhs)`
- `bool operator< (const QueuedPacket &lhs, const QueuedPacket &rhs)`
- `bool operator> (const QueuedPacket &lhs, const QueuedPacket &rhs)`
- `bool operator<= (const QueuedPacket &lhs, const QueuedPacket &rhs)`
- `bool operator>= (const QueuedPacket &lhs, const QueuedPacket &rhs)`
- `std::ostream & operator<< (std::ostream &os, const QueuedPacket &queued_packet)`

15.32.1 Detailed Description

A packet in the queue to be sent out.

Forms a node in the [PacketQueue](#) class.

This is the data structure used in the priority queues used by the [Connection](#) class. It stores a MAVLink packet as well as a priority and ticket number used to maintain packet order in the priority queue when packets have the same priority.

See also

[PacketQueue](#)

Definition at line 39 of file [QueuedPacket.hpp](#).

15.32.2 Constructor & Destructor Documentation

15.32.2.1 QueuedPacket() [1/3]

```
QueuedPacket::QueuedPacket (
    const QueuedPacket & other ) [default]
```

Copy constructor.

Parameters

<i>other</i>	QueuedPacket to copy from.
--------------	--

15.32.2.2 QueuedPacket() [2/3]

```
QueuedPacket::QueuedPacket (
    QueuedPacket && other ) [default]
```

Move constructor.

Parameters

<i>other</i>	QueuedPacket to move from.
--------------	--

15.32.2.3 QueuedPacket() [3/3]

```
QueuedPacket::QueuedPacket (
    std::shared_ptr< const Packet > packet,
    int priority,
    unsigned long long ticket_number )
```

Construct a queued packet.

Parameters

<i>packet</i>	The packet to store in the queue.
<i>priority</i>	The priority to send the packet with, higher numbers result in a higher priority.
<i>ticket_number</i>	A number that should always be incremented for each queued packet created per packet queue, even if this increment causes an unsigned integer wraparound.

Exceptions

<i>std::invalid_argument</i>	if the given packet pointer is nullptr.
------------------------------	---

Definition at line 36 of file [QueuedPacket.cpp](#).

15.32.3 Member Function Documentation

15.32.3.1 operator=() [1/2]

```
QueuedPacket& QueuedPacket::operator= (
    const QueuedPacket & other ) [default]
```

Assignment operator.

Parameters

<i>other</i>	QueuedPacket to copy from.
--------------	----------------------------

15.32.3.2 operator=() [2/2]

```
QueuedPacket& QueuedPacket::operator= (
    QueuedPacket && other ) [default]
```

Assignment operator (by move semantics).

Parameters

<i>other</i>	QueuedPacket to move from.
--------------	----------------------------

15.32.3.3 packet()

```
std::shared_ptr< const Packet > QueuedPacket::packet ( ) const
```

Return the contained packet.

Returns

The contained MAVLink packet.

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

15.32.4 Friends And Related Function Documentation**15.32.4.1 operator"!=**

```
bool operator!= (
    const QueuedPacket & lhs,
    const QueuedPacket & rhs ) [friend]
```

Inequality comparison.

Note

It should never be the case that two queued packets have the same ticket number and thus two queued packets should never be equal.

Parameters

<i>lhs</i>	The left hand side queued packet.
<i>rhs</i>	The right hand side queued packet.

Return values

<i>true</i>	if <i>lhs</i> and <i>rhs</i> do not have the same priority and ticket number.
-------------	---

Return values

<i>false</i>	if <i>lhs</i> and <i>rhs</i> have the same priority and ticket number.
--------------	--

Definition at line 90 of file [QueuedPacket.cpp](#).

15.32.4.2 operator<

```
bool operator< (
    const QueuedPacket & lhs,
    const QueuedPacket & rhs ) [friend]
```

Less than comparison.

The priority is considered first, followed by the ticket number in reverse order (lower ticket number is greater).

Note

The ticket number is considered to be a wrapping integer and thus numbers that are within `std::numeric_limits<unsigned long long>::max() / 2` of each other are considered in the same range. In this way 0 is greater than `std::numeric_limits<unsigned long long>::max()`. Because of this it is important that anything relying on ordering must not contain a range of ticket numbers equal to or greater than `std::numeric_limits<unsigned long long>::max() / 2`.

Parameters

<i>lhs</i>	The left hand side queued packet.
<i>rhs</i>	The right hand side queued packet.

Return values

<i>true</i>	if <i>lhs</i> is less than <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is not less than <i>rhs</i> .

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

15.32.4.3 operator<<

```
std::ostream & operator<< (
    std::ostream & os,
    const QueuedPacket & queued_packet ) [friend]
```

Print the packet to the given output stream.

Some examples are:

- HEARTBEAT (#1) from 16.8 (v1.0) with priority -3
- PING (#4) from 128.4 to 16.8 (v2.0) with priority 0
- DATA_TRANSMISSION_HANDSHAKE (#130) from 16.8 (v2.0) with priority 3
- ENCAPSULATED_DATA (#131) from 128.4 (v2.0) with priority 1

Parameters

<i>os</i>	The output stream to print to.
<i>queued_packet</i>	The queued packet to print.

Returns

The output stream.

Definition at line 192 of file [QueuedPacket.cpp](#).

15.32.4.4 operator<=

```
bool operator<= (
    const QueuedPacket & lhs,
    const QueuedPacket & rhs ) [friend]
```

Less than or equal comparison.

Parameters

<i>lhs</i>	The left hand side queued packet.
<i>rhs</i>	The right hand side queued packet.

Return values

<i>true</i>	if <i>lhs</i> is less than or equal to <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is greater than <i>rhs</i> .

Definition at line 159 of file [QueuedPacket.cpp](#).

15.32.4.5 operator==

```
bool operator== (
    const QueuedPacket & lhs,
    const QueuedPacket & rhs ) [friend]
```

Equality comparison.

Note

It should never be the case that two queued packets have the same ticket number and thus two queued packets should never be equal.

Parameters

<i>lhs</i>	The left hand side queued packet.
<i>rhs</i>	The right hand side queued packet.

Return values

<i>true</i>	if <i>lhs</i> and <i>rhs</i> have the same priority and ticket number.
<i>false</i>	if <i>lhs</i> and <i>rhs</i> do not have the same priority and ticket number.

Definition at line 71 of file [QueuedPacket.cpp](#).

15.32.4.6 operator>

```
bool operator> (
    const QueuedPacket & lhs,
    const QueuedPacket & rhs ) [friend]
```

Greater than comparison.

The priority is considered first, followed by the ticket number in reverse order (lower ticket number is greater).

Note

The ticket number is considered to be a wrapping integer and thus numbers that are within `std::numeric_limits<unsigned long long>::max() / 2` of each other are considered in the same range. In this way 0 is greater than `std::numeric_limits<unsigned long long>::max()`. Because of this it is important that anything relying on ordering must not contain a range of ticket numbers equal to or greater than `std::numeric_limits<unsigned long long>::max() / 2`.

Parameters

<i>lhs</i>	The left hand side queued packet.
<i>rhs</i>	The right hand side queued packet.

Return values

<i>true</i>	if <i>lhs</i> is less than <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is not less than <i>rhs</i> .

Definition at line 143 of file [QueuedPacket.cpp](#).

15.32.4.7 operator>=

```
bool operator>= (
    const QueuedPacket & lhs,
    const QueuedPacket & rhs ) [friend]
```

Greater than or equal comparison.

Parameters

<i>lhs</i>	The left hand side queued packet.
<i>rhs</i>	The right hand side queued packet.

Return values

<i>true</i>	if <i>lhs</i> is greater than or equal to <i>rhs</i> .
<i>false</i>	if <i>lhs</i> is less than <i>rhs</i> .

Definition at line 173 of file [QueuedPacket.cpp](#).

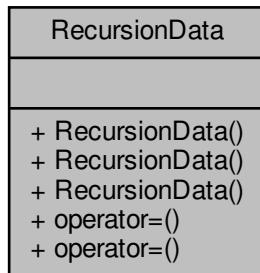
The documentation for this class was generated from the following files:

- [QueuedPacket.hpp](#)
- [QueuedPacket.cpp](#)

15.33 RecursionData Class Reference

```
#include <RecursionData.hpp>
```

Collaboration diagram for RecursionData:



Public Member Functions

- `RecursionData ()=default`
- `RecursionData (const RecursionData &other)`
- `RecursionData (RecursionData &&other)`
- `RecursionData & operator= (const RecursionData &other)`
- `RecursionData & operator= (RecursionData &&other)`

Friends

- class `RecursionGuard`

15.33.1 Detailed Description

A data structure used by `RecursionGuard` to detect unwanted recursion.

Note

While `RecursionData` supports copy and move semantics both with constructors and assignment operator, a recursion data structure should never change instance. One way to deal with this would have been to delete these operators but this would force users to manually implement copy and move semantics for their classes. Therefore, `RecursionData` will always make a new data structure on copy, move or assignment, allowing default copy and move constructors/assignment operators to be created for classes containing a `RecursionData` instance.

Definition at line 39 of file `RecursionData.hpp`.

15.33.2 Constructor & Destructor Documentation

15.33.2.1 RecursionData() [1/3]

```
RecursionData::RecursionData ( ) [default]
```

15.33.2.2 RecursionData() [2/3]

```
RecursionData::RecursionData (
    const RecursionData & other ) [inline]
```

Definition at line 45 of file [RecursionData.hpp](#).

15.33.2.3 RecursionData() [3/3]

```
RecursionData::RecursionData (
    RecursionData && other ) [inline]
```

Definition at line 49 of file [RecursionData.hpp](#).

15.33.3 Member Function Documentation

15.33.3.1 operator=() [1/2]

```
RecursionData& RecursionData::operator= (
    const RecursionData & other ) [inline]
```

Definition at line 53 of file [RecursionData.hpp](#).

15.33.3.2 operator=() [2/2]

```
RecursionData& RecursionData::operator= (
    RecursionData && other ) [inline]
```

Definition at line 59 of file [RecursionData.hpp](#).

15.33.4 Friends And Related Function Documentation

15.33.4.1 RecursionGuard

```
friend class RecursionGuard [friend]
```

Definition at line [41](#) of file [RecursionData.hpp](#).

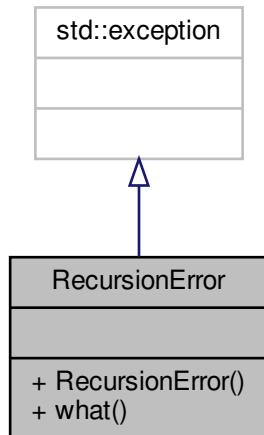
The documentation for this class was generated from the following file:

- [RecursionData.hpp](#)

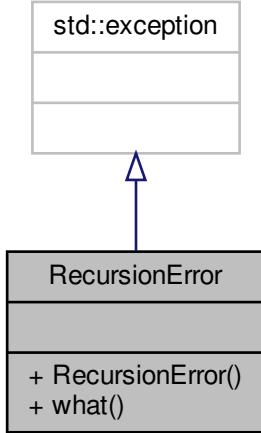
15.34 RecursionError Class Reference

```
#include <RecursionError.hpp>
```

Inheritance diagram for RecursionError:



Collaboration diagram for RecursionError:



Public Member Functions

- [RecursionError](#) (std::string message)
- const char * [what \(\)](#) const noexcept

15.34.1 Detailed Description

Exception type emitted by a recursion guard.

Definition at line [28](#) of file [RecursionError.hpp](#).

15.34.2 Constructor & Destructor Documentation

15.34.2.1 RecursionError()

```
RecursionError::RecursionError (
    std::string message )
```

Construct a [RecursionError](#) given a message.

Parameters

<i>message</i>	The error message.
----------------	--------------------

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

15.34.3 Member Function Documentation

15.34.3.1 what()

```
const char * RecursionError::what ( ) const [noexcept]
```

Return error message string.

Returns

Error message string.

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

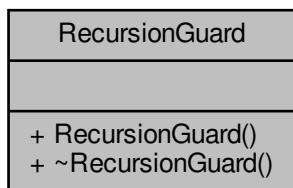
The documentation for this class was generated from the following files:

- [RecursionError.hpp](#)
- [RecursionError.cpp](#)

15.35 RecursionGuard Class Reference

```
#include <RecursionGuard.hpp>
```

Collaboration diagram for RecursionGuard:



Public Member Functions

- [RecursionGuard \(RecursionData &data\)](#)
- [~RecursionGuard \(\)](#)

15.35.1 Detailed Description

Guard against recursion.

A recursion guard is an RAII (Resource Acquisition Is Initialization) data structure used to raise an error upon recursion. The constructor marks a [RecursionData](#) structure, acquiring ownership of the containing function (within the given thread). Recursion guards treat calls from different threads separately, therefore, it will not guard against re-entrancy.

An example of how to use this is:

```
#include "RecursionGuard.hpp"

int a_function(int value)
{
    // shared data between calls
    static RecursionData rdata;
    // take ownership of the call
    RecursionGuard rguard(rdata);
    return b_function(value);
    // the recursion guard is released upon destruction of rguard
}
```

If `b_function`, or any function it calls, ever calls `a_function` then this will throw [RecursionError](#). However, if multiple threads call `a_function` (possibly at the same time) but `b_function` does not call `a_function` then no error will be thrown.

See also

[RecursionData](#)
[RecursionError](#)

Definition at line 60 of file [RecursionGuard.hpp](#).

15.35.2 Constructor & Destructor Documentation

15.35.2.1 RecursionGuard()

```
RecursionGuard::RecursionGuard (
    RecursionData & data )
```

Construct a [RecursionGuard](#).

This marks the given [RecursionData](#) structure, ensuring it cannot be used to construct another guard without raising a [RecursionError](#).

Parameters

<code>data</code>	The object used to prevent recursion.
-------------------	---------------------------------------

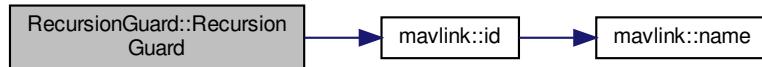
Exceptions

<code>RecursionError</code>	if the given <code>data</code> structure has already been marked by a <code>RecursionGuard</code> instance that is still in scope.
-----------------------------	--

Definition at line 35 of file [RecursionGuard.cpp](#).

References [mavlink::id\(\)](#).

Here is the call graph for this function:



15.35.2.2 ~RecursionGuard()

`RecursionGuard::~RecursionGuard()`

Release the lock on the contain [RecursionData](#).

This allows the function to be called again without error.

Definition at line 52 of file [RecursionGuard.cpp](#).

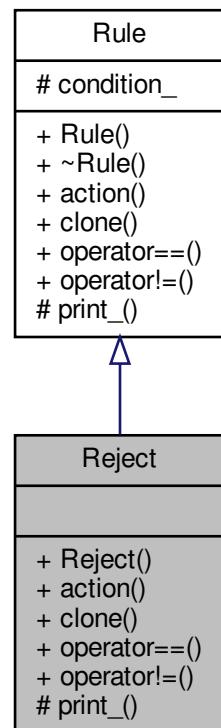
The documentation for this class was generated from the following files:

- [RecursionGuard.hpp](#)
- [RecursionGuard.cpp](#)

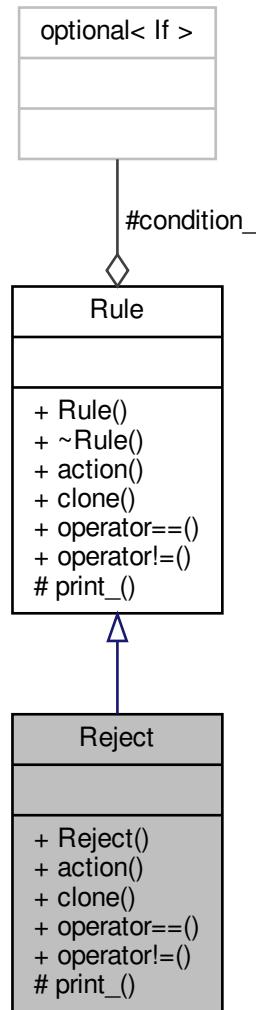
15.36 Reject Class Reference

```
#include <Reject.hpp>
```

Inheritance diagram for Reject:



Collaboration diagram for Reject:



Public Member Functions

- `Reject (std::optional< If > condition={})`
- virtual `Action action (const Packet &packet, const MAVAddress &address) const`
- virtual `std::unique_ptr< Rule > clone () const`
- virtual `bool operator== (const Rule &other) const`
- virtual `bool operator!= (const Rule &other) const`

Protected Member Functions

- virtual `std::ostream & print_ (std::ostream &os) const`

Protected Attributes

- std::optional< [If](#) > `condition_`

15.36.1 Detailed Description

[Rule](#) to reject a packet.

Definition at line [35](#) of file [Reject.hpp](#).

15.36.2 Constructor & Destructor Documentation

15.36.2.1 Reject()

```
Reject::Reject (
    std::optional< If > condition = {} )
```

Construct a reject rule.

A reject rule is used to reject packet/address combinations that match the condition of the rule.

Parameters

<code>condition</code>	The condition used to determine the rule matches a particular packet/address combination given to the action method. The default is {} which indicates the rule matches any packet/address combination.
------------------------	---

See also

[action](#)

Definition at line [42](#) of file [Reject.cpp](#).

15.36.3 Member Function Documentation

15.36.3.1 action()

```
Action Reject::action (
    const Packet & packet,
    const MAVAddress & address ) const [virtual]
```

Decide what to do with a [Packet](#).

Determine what action to take with the given packet sent to the given address. The possible actions are documented in the [Action](#) class. The continue object is always returned if the condition was set and does not match the packet/address combination.

Parameters

<i>packet</i>	The packet to determine whether to allow or not.
<i>address</i>	The address the packet will be sent out on if the action dictates it.

Returns

The action to take with the packet. If this is the accept object, it may also contain a priority for the packet.

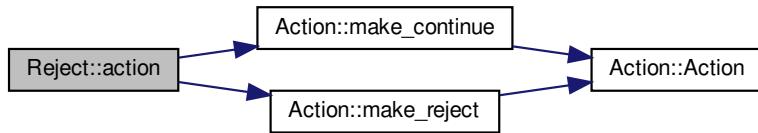
If the condition has not been set or it matches the given packet/address combination then it will return the reject [Action](#) object, otherwise it will return the continue [Action](#) object.

Implements [Rule](#).

Definition at line 72 of file [Reject.cpp](#).

References [Rule::condition_](#), [Action::make_continue\(\)](#), and [Action::make_reject\(\)](#).

Here is the call graph for this function:



15.36.3.2 clone()

```
std::unique_ptr< Rule > Reject::clone( ) const [virtual]
```

Return a copy of the [Rule](#) polymorphically.

This allows [Rule](#)'s to be copied without knowing the derived type.

Returns

A pointer to a new object with base type [Rule](#) which is an exact copy of this one.

Implements [Rule](#).

Definition at line 84 of file [Reject.cpp](#).

References [Rule::condition_](#).

15.36.3.3 operator!=()

```
bool Reject::operator!= (
    const Rule & other ) const [virtual]
```

Inequality comparison.

Compares the type of the [Rule](#) and the condition ([If](#)) if set.

Parameters

<i>other</i>	The other rule to compare this to.
--------------	------------------------------------

Return values

<i>true</i>	if this rule is not the same as <i>other</i> .
<i>false</i>	if this rule is the same as <i>other</i> .

Implements [Rule](#).

Definition at line 97 of file [Reject.cpp](#).

References [Rule::condition_](#).

15.36.3.4 operator==()

```
bool Reject::operator== (
    const Rule & other ) const [virtual]
```

Equality comparison.

Compares the type of the [Rule](#) and the condition ([If](#)) if set.

Parameters

<i>other</i>	The other rule to compare this to.
--------------	------------------------------------

Return values

<i>true</i>	if this rule is the same as <i>other</i> .
<i>false</i>	if this rule is not the same as <i>other</i> .

Implements [Rule](#).

Definition at line 90 of file [Reject.cpp](#).

References [Rule::condition_](#).

15.36.3.5 print_()

```
std::ostream & Reject::print_ (
    std::ostream & os ) const [protected], [virtual]
```

Print the rule to the given output stream.

Parameters

<code>os</code>	The output stream to print to.
-----------------	--------------------------------

Returns

The output stream.

Prints "reject" or "reject <If Statement>" if the rule's condition was set.

Implements [Rule](#).

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

References [Rule::condition_](#).

15.36.4 Member Data Documentation

15.36.4.1 condition_

```
std::optional<If> Rule::condition_ [protected], [inherited]
```

Definition at line 91 of file [Rule.hpp](#).

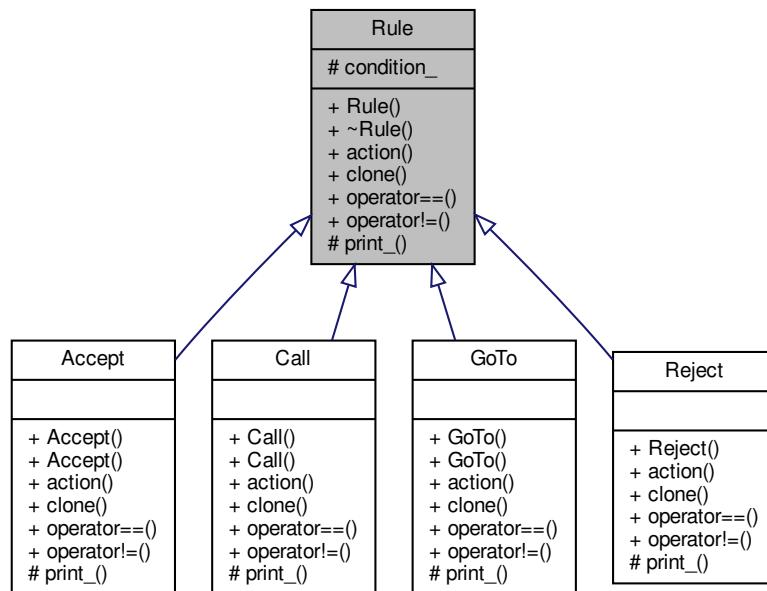
The documentation for this class was generated from the following files:

- [Reject.hpp](#)
- [Reject.cpp](#)

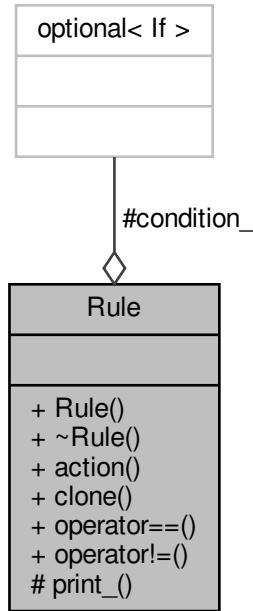
15.37 Rule Class Reference

```
#include <Rule.hpp>
```

Inheritance diagram for Rule:



Collaboration diagram for Rule:



Public Member Functions

- `Rule (std::optional< If > condition={})`
- `virtual ~Rule ()`
- `virtual Action action (const Packet &packet, const MAVAddress &address) const =0`
- `virtual std::unique_ptr< Rule > clone () const =0`
- `virtual bool operator==(const Rule &other) const =0`
- `virtual bool operator!=(const Rule &other) const =0`

Protected Member Functions

- `virtual std::ostream & print_(std::ostream &os) const =0`

Protected Attributes

- `std::optional< If > condition_`

Friends

- `std::ostream & operator<< (std::ostream &os, const Rule &action)`

15.37.1 Detailed Description

Base class of all rules, used in filter [Chain](#)'s.

[Rule](#)'s are used to determine an [Action](#) to take with a packet based on its type, source address, and destination address. They are very much like the rules found in a typical software defined firewalls.

Definition at line [38](#) of file [Rule.hpp](#).

15.37.2 Constructor & Destructor Documentation

15.37.2.1 Rule()

```
Rule::Rule (
    std::optional< If > condition = {} )
```

Base constructor for [Rule](#) classes.

Parameters

<i>condition</i>	The condition used to determine if the rule matches a particular packet/address combination given to the action method. The default is {} which indicates the rule matches any packet/address combination.
------------------	--

See also

[action](#)

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

15.37.2.2 ~Rule()

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

Definition at line [42](#) of file [Rule.cpp](#).

15.37.3 Member Function Documentation

15.37.3.1 `action()`

```
virtual Action Rule::action (
    const Packet & packet,
    const MAVAddress & address ) const [pure virtual]
```

Decide what to do with a [Packet](#).

Determine what action to take with the given packet sent to the given address. The possible actions are documented in the [Action](#) class. The continue object is always returned if the condition was set and does not match the packet/address combination.

Parameters

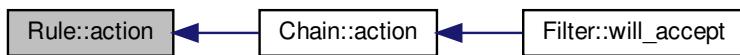
<i>packet</i>	The packet to determine whether to allow or not.
<i>address</i>	The address the <i>packet</i> will be sent out on if the action dictates it.

Returns

The action to take with the packet. If this is the accept object, it may also contain a priority for the packet.

Implemented in [GoTo](#), [Call](#), [Accept](#), and [Reject](#).

Here is the caller graph for this function:

15.37.3.2 `clone()`

```
virtual std::unique_ptr<Rule> Rule::clone ( ) const [pure virtual]
```

Return a copy of the [Rule](#) polymorphically.

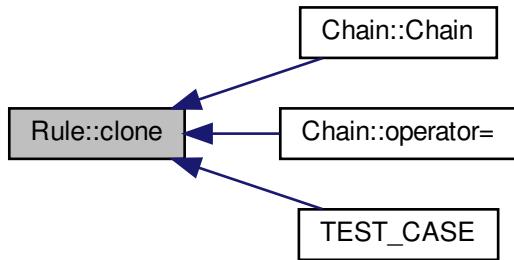
This allows [Rule](#)'s to be copied without knowing the derived type.

Returns

A pointer to a new object with base type [Rule](#) which is an exact copy of this one.

Implemented in [GoTo](#), [Call](#), [Accept](#), and [Reject](#).

Here is the caller graph for this function:

**15.37.3.3 operator"!=()**

```
virtual bool Rule::operator!= (
    const Rule & other ) const [pure virtual]
```

Inequality comparison.

Compares the type of the [Rule](#) and the condition ([If](#)) if set.

Parameters

<i>other</i>	The other rule to compare this to.
--------------	------------------------------------

Return values

<i>true</i>	if this rule is not the same as <i>other</i> .
<i>false</i>	if this rule is the same as <i>other</i> .

Implemented in [GoTo](#), [Call](#), [Accept](#), and [Reject](#).

15.37.3.4 operator==()

```
virtual bool Rule::operator== (
    const Rule & other ) const [pure virtual]
```

Equality comparison.

Compares the type of the [Rule](#) and the condition ([If](#)) if set.

Parameters

<i>other</i>	The other rule to compare this to.
--------------	------------------------------------

Returns

<i>true</i>	if this rule is the same as <i>other</i> .
<i>false</i>	if this rule is not the same as <i>other</i> .

Implemented in [GoTo](#), [Call](#), [Accept](#), and [Reject](#).

15.37.3.5 print_()

```
virtual std::ostream& Rule::print_ (
    std::ostream & os ) const [protected], [pure virtual]
```

Print the rule to the given output stream.

Parameters

<i>os</i>	The output stream to print to.
-----------	--------------------------------

Returns

The output stream.

Implemented in [GoTo](#), [Call](#), [Accept](#), and [Reject](#).

15.37.4 Friends And Related Function Documentation

15.37.4.1 operator<<

```
std::ostream & operator<< (
    std::ostream & os,
    const Rule & rule ) [friend]
```

Print the given rule to the given output stream.

Note

This is a polymorphic print, it will work on any child of [Rule](#) even if the pointer/reference is to the base class [Rule](#).

Some examples are:

- accept
- accept with priority 3
- reject if ENCAPSULATED_DATA
- call gcs_in if from 192.168
- goto autopilot_out

Parameters

<i>os</i>	The output stream to print to.
<i>rule</i>	The rule (or any child of Rule) to print.

Returns

The output stream.

Definition at line 65 of file [Rule.cpp](#).

15.37.5 Member Data Documentation

15.37.5.1 condition_

```
std::optional<If> Rule::condition_ [protected]
```

Definition at line 91 of file [Rule.hpp](#).

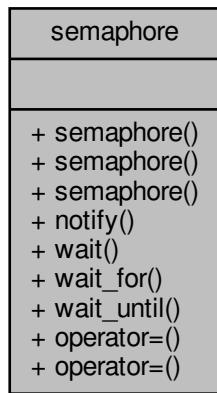
The documentation for this class was generated from the following files:

- [Rule.hpp](#)
- [Rule.cpp](#)

15.38 semaphore Class Reference

```
#include <semaphore.hpp>
```

Collaboration diagram for semaphore:



Public Member Functions

- `semaphore (const semaphore &other)=delete`
- `semaphore (semaphore &&other)=delete`
- `semaphore (size_t initial_value=0)`
- `void notify ()`
- `void wait ()`
- template<class Rep , class Period >
 `bool wait_for (const std::chrono::duration< Rep, Period > &rel_time)`
- template<class Clock , class Duration >
 `bool wait_until (const std::chrono::time_point< Clock, Duration > &timeout_time)`
- `semaphore & operator= (const semaphore &other)=delete`
- `semaphore & operator= (semaphore &&other)=delete`

15.38.1 Detailed Description

A weak semaphore implementation.

Note

This uses a different naming scheme than the rest of this project because it is intended to have the same feel as the standard library.

This semaphore implementation is based on <https://gist.github.com/sguzman/9594227>

Definition at line 36 of file `semaphore.hpp`.

15.38.2 Constructor & Destructor Documentation

15.38.2.1 semaphore() [1/3]

```
semaphore::semaphore (
    const semaphore & other ) [delete]
```

15.38.2.2 semaphore() [2/3]

```
semaphore::semaphore (
    semaphore && other ) [delete]
```

15.38.2.3 semaphore() [3/3]

```
semaphore::semaphore (
    size_t initial_value = 0 )
```

Construct a semaphore with the given initial value.

Parameters

<i>initial_value</i>	The initial value of the semaphore. Defaults to 0.
----------------------	--

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

15.38.3 Member Function Documentation

15.38.3.1 notify()

```
void semaphore::notify ( )
```

Signal the semaphore.

Increments the semaphore.

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

15.38.3.2 operator=() [1/2]

```
semaphore& semaphore::operator= (
    const semaphore & other ) [delete]
```

15.38.3.3 operator=() [2/2]

```
semaphore& semaphore::operator= (
    semaphore && other ) [delete]
```

15.38.3.4 wait()

```
void semaphore::wait ( )
```

Wait on the semaphore.

Decrements the semaphore.

Definition at line 52 of file [semaphore.cpp](#).

15.38.3.5 wait_for()

```
template<class Rep , class Period >
bool semaphore::wait_for (
    const std::chrono::duration< Rep, Period > & rel_time )
```

Wait on the semaphore, or a given duration timeout.

Parameters

<i>rel_time</i>	The duration of the timeout.
-----------------	------------------------------

Return values

<i>true</i>	The semaphore has been successfully decremented.
<i>false</i>	The semaphore timed out.

Definition at line 66 of file [semaphore.hpp](#).

15.38.3.6 wait_until()

```
template<class Clock , class Duration >
bool semaphore::wait_until (
    const std::chrono::time_point< Clock, Duration > & timeout_time )
```

Wait on the semaphore, or a given timepoint timeout.

Parameters

<i>timeout_time</i>	The timepoint for the timeout.
---------------------	--------------------------------

Return values

<i>true</i>	The semaphore has been successfully decremented.
<i>false</i>	The semaphore timed out.

Definition at line 90 of file [semaphore.hpp](#).

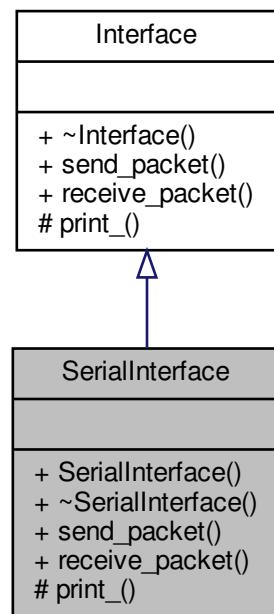
The documentation for this class was generated from the following files:

- [semaphore.hpp](#)
- [semaphore.cpp](#)

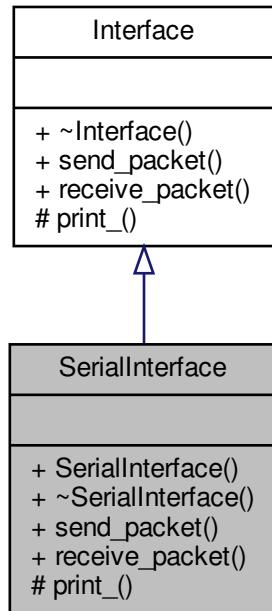
15.39 SerialInterface Class Reference

```
#include <SerialInterface.hpp>
```

Inheritance diagram for SerialInterface:



Collaboration diagram for SerialInterface:



Public Member Functions

- `SerialInterface (std::unique_ptr< SerialPort > port, std::shared_ptr< ConnectionPool > connection_pool, std::unique_ptr< Connection > connection)`
- `~SerialInterface ()=default`
- `void send_packet (const std::chrono::nanoseconds &timeout) final`
- `void receive_packet (const std::chrono::nanoseconds &timeout) final`

Protected Member Functions

- `std::ostream & print_ (std::ostream &os) const final`

15.39.1 Detailed Description

A serial port interface.

An interface (for sending and receiving packets) implementing the serial port protocol.

Definition at line 37 of file [SerialInterface.hpp](#).

15.39.2 Constructor & Destructor Documentation

15.39.2.1 SerialInterface()

```
SerialInterface::SerialInterface (
    std::unique_ptr< SerialPort > port,
    std::shared_ptr< ConnectionPool > connection_pool,
    std::unique_ptr< Connection > connection )
```

Construct a serial port interface using a given device.

Parameters

<i>port</i>	The serial port device to communicate over.
<i>connection_pool</i>	The connection pool to use for sending packets the interface has received and to register the <i>connection</i> with.
<i>connection</i>	The connection to get packets to send packets from. This will be registered with the given <i>ConnectionPool</i> .

Exceptions

<i>std::invalid_argument</i>	if the <i>serial_port</i> device pointer is null.
<i>std::invalid_argument</i>	if the <i>connection_pool</i> pointer is null.
<i>std::invalid_argument</i>	if the <i>connection</i> pointer is null.

Definition at line 39 of file [SerialInterface.cpp](#).

15.39.2.2 ~SerialInterface()

```
SerialInterface::~SerialInterface () [default]
```

15.39.3 Member Function Documentation

15.39.3.1 print_()

```
std::ostream & SerialInterface::print_ (
    std::ostream & os ) const [final], [protected], [virtual]
```

Print the interface to the given output stream.

Parameters

<i>os</i>	The output stream to print to.
-----------	--------------------------------

Returns

The output stream.

Example:

```
serial {
    device /dev/ttyUSB0;
    baudrate 115200;
    flow_control yes;
}
```

Parameters

<i>os</i>	The output stream to print to.
-----------	--------------------------------

Implements [Interface](#).

Definition at line [121](#) of file [SerialInterface.cpp](#).

15.39.3.2 receive_packet()

```
void SerialInterface::receive_packet (
    const std::chrono::nanoseconds & timeout ) [final], [virtual]
```

Receive a packet on the interface.

Parameters

<i>timeout</i>	The maximum amount of time to wait for incoming data.
----------------	---

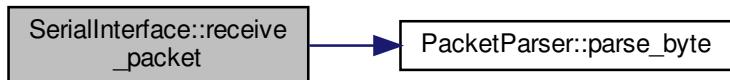
Reads the data in the serial port's receive buffer or waits for up to *timeout* until data arrives if no data is present in the serial port buffer.

Implements [Interface](#).

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

References [PacketParser::parse_byte\(\)](#).

Here is the call graph for this function:



15.39.3.3 send_packet()

```
void SerialInterface::send_packet (
    const std::chrono::nanoseconds & timeout ) [final], [virtual]
```

Send a packet from one of the interface's connections.

Note

Which connection to take this packet from is not defined but it must not starve any one of the [Interface](#)'s connections.

Parameters

<i>timeout</i>	The maximum amount of time to wait for a packet to be available for sending.
----------------	--

Writes up to one packet from the contained connection to the serial port.

Implements [Interface](#).

Definition at line [70](#) of file [SerialInterface.cpp](#).

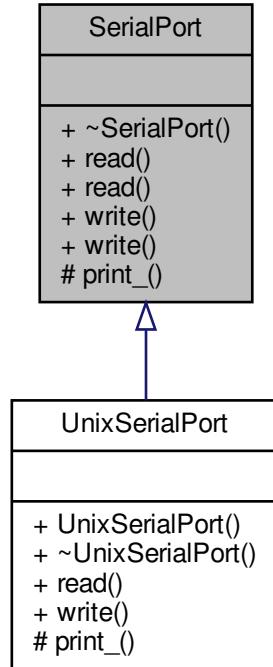
The documentation for this class was generated from the following files:

- [SerialInterface.hpp](#)
- [SerialInterface.cpp](#)

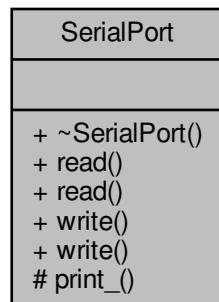
15.40 SerialPort Class Reference

```
#include <SerialPort.hpp>
```

Inheritance diagram for SerialPort:



Collaboration diagram for SerialPort:



Public Types

- enum `Parity` {
 `NONE`, `ODD`, `EVEN`, `MARK`,
 `SPACE` }
- enum `Feature` { `DEFAULT` = 0, `FLOW_CONTROL` = 1 << 0 }

Public Member Functions

- virtual `~SerialPort()`
- virtual `std::vector< uint8_t > read` (`const std::chrono::nanoseconds &timeout=std::chrono::nanoseconds::zero()`)
- virtual `void read (std::back_insert_iterator< std::vector< uint8_t > > it, const std::chrono::nanoseconds &timeout=std::chrono::nanoseconds::zero())`
- virtual `void write (const std::vector< uint8_t > &data)`
- virtual `void write (std::vector< uint8_t >::const_iterator first, std::vector< uint8_t >::const_iterator last)`

Protected Member Functions

- virtual `std::ostream & print_ (std::ostream &os) const`

Friends

- `std::ostream & operator<< (std::ostream &os, const SerialPort &serial_port)`

15.40.1 Detailed Description

The base class of all serial port classes.

This provides an abstraction of serial ports across operating systems.

Warning

This class should be treated as pure virtual and should never be instantiated.

Either `read(const std::chrono::nanoseconds &)` or `read(std::back_insert_iterator<std::vector<uint8_t> >, const std::chrono::nanoseconds &)` must be overridden in child classes to avoid infinite recursion.

Either `write(const std::vector<uint8_t > &data)` or `write(std::vector<uint8_t >::const_iterator, std::vector<uint8_t >::const_iterator)` must be overridden in child classes to avoid infinite recursion.

Definition at line 44 of file `SerialPort.hpp`.

15.40.2 Member Enumeration Documentation

15.40.2.1 Feature

```
enum SerialPort::Feature
```

Feature bitflags.

Enumerator

DEFAULT	No special features.
FLOW_CONTROL	Enable flow control.

Definition at line 59 of file [SerialPort.hpp](#).

15.40.2.2 Parity

```
enum SerialPort::Parity
```

Parity options.

Enumerator

NONE	No parity.
ODD	Odd parity, must have odd number of set bits.
EVEN	Even parity, must have even number of set bits.
MARK	Fill parity bit with 1.
SPACE	Fill parity bit with 0.

Definition at line 49 of file [SerialPort.hpp](#).

15.40.3 Constructor & Destructor Documentation**15.40.3.1 ~SerialPort()**

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

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

15.40.4 Member Function Documentation**15.40.4.1 print_()**

```
std::ostream & SerialPort::print_ (
    std::ostream & os ) const [protected], [virtual]
```

Print the serial port to the given output stream.

Parameters

<code>os</code>	The output stream to print to.
-----------------	--------------------------------

Returns

The output stream.

Reimplemented in [UnixSerialPort](#).

Definition at line 104 of file [SerialPort.cpp](#).

15.40.4.2 `read()` [1/2]

```
std::vector< uint8_t > SerialPort::read (
    const std::chrono::nanoseconds & timeout = std::chrono::nanoseconds::zero() ) [virtual]
```

Read data from the serial port.

Note

The `timeout` is not guaranteed to be up to nanosecond precision, the actual precision is up to the operating system's implementation but is guaranteed to have at least millisecond precision.

Parameters

<code>timeout</code>	How long to wait for data to arrive on the serial port if there is not already data to read. The default is to not wait.
----------------------	--

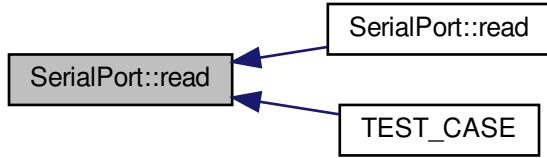
Returns

The data read from the serial port.

Reimplemented in [UnixSerialPort](#).

Definition at line 47 of file [SerialPort.cpp](#).

Here is the caller graph for this function:



15.40.4.3 `read()` [2/2]

```
void SerialPort::read (
    std::back_insert_iterator< std::vector< uint8_t >> it,
    const std::chrono::nanoseconds & timeout = std::chrono::nanoseconds::zero() ) [virtual]
```

Read data from the serial port.

Note

The `timeout` is not guaranteed to be up to nanosecond precision, the actual precision is up to the operating system's implementation but is guaranteed to have at least millisecond precision.

Parameters

<code>it</code>	A back insert iterator to read bytes into.
<code>timeout</code>	How long to wait for data to arrive on the serial port if there is not already data to read. The default is to not wait.

Definition at line 65 of file [SerialPort.cpp](#).

References [read\(\)](#).

Here is the call graph for this function:



15.40.4.4 write() [1/2]

```
void SerialPort::write (
    const std::vector< uint8_t > & data )  [virtual]
```

Write data to the serial port (blocking write).

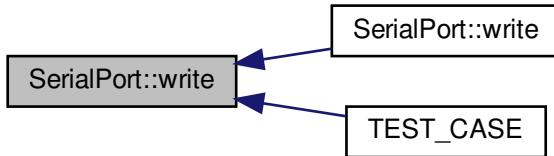
Parameters

<i>data</i>	The bytes to send.
-------------	--------------------

Reimplemented in [UnixSerialPort](#).

Definition at line [78](#) of file [SerialPort.cpp](#).

Here is the caller graph for this function:



15.40.4.5 `write()` [2/2]

```
void SerialPort::write (
    std::vector< uint8_t >::const_iterator first,
    std::vector< uint8_t >::const_iterator last ) [virtual]
```

Write data to the serial port (blocking write).

Parameters

<i>first</i>	Iterator to first byte in range to send.
<i>last</i>	Iterator to one past the last byte to send.

Definition at line 89 of file [SerialPort.cpp](#).

References [write\(\)](#).

Here is the call graph for this function:



15.40.5 Friends And Related Function Documentation

15.40.5.1 `operator<<`

```
std::ostream & operator<< (
    std::ostream & os,
    const SerialPort & serial_port ) [friend]
```

Print the given serial port to the given output stream.

Note

This is a polymorphic print, it will work on any child of [SerialPort](#) even if the pointer/reference is to the base class [SerialPort](#).

An example:

```
serial {  
    device /dev/ttyUSB0;  
    baudrate 115200;  
    flow_control yes;  
}
```

The base [SerialPort](#) class will print:

```
unknown serial port
```

Parameters

<i>os</i>	The output stream to print to.
<i>serial_port</i>	The serial port (or any child of SerialPort) to print.

Returns

The output stream.

Definition at line 137 of file [SerialPort.cpp](#).

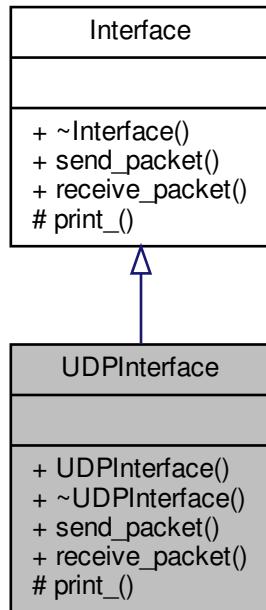
The documentation for this class was generated from the following files:

- [SerialPort.hpp](#)
- [SerialPort.cpp](#)

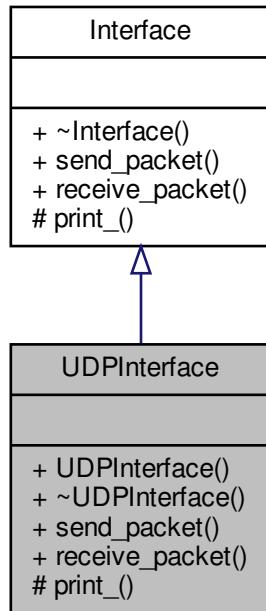
15.41 UDPInterface Class Reference

```
#include <UDPIface.hpp>
```

Inheritance diagram for UDPIface:



Collaboration diagram for UDPInterface:



Public Member Functions

- `UDPInterface (std::unique_ptr< UDPSocket > socket, std::shared_ptr< ConnectionPool > connection_pool, std::unique_ptr< ConnectionFactory>>> connection_factory)`
- `~UDPInterface ()=default`
- `void send_packet (const std::chrono::nanoseconds &timeout) final`
- `void receive_packet (const std::chrono::nanoseconds &timeout) final`

Protected Member Functions

- `std::ostream & print_ (std::ostream &os) const final`

15.41.1 Detailed Description

A UDP (User Datagram Protocol) interface.

An interface (for sending and receiving packets) implementing the user datagram protocol.

Definition at line 40 of file [UDPInterface.hpp](#).

15.41.2 Constructor & Destructor Documentation

15.41.2.1 UDPInterface()

```
UDPInterface::UDPInterface (
    std::unique_ptr< UDPSocket > socket,
    std::shared_ptr< ConnectionPool > connection_pool,
    std::unique_ptr< ConnectionFactory>>> connection_factory )
```

Construct a UDP interface using a given socket.

Parameters

<i>socket</i>	The UDP socket to communicate over.
<i>connection_pool</i>	The connection pool to use for sending packets and to register new connections with.
<i>connection_factory</i>	The connection factory to use for constructing new connections when an outside connection is made.

Exceptions

<i>std::invalid_argument</i>	if the serial port device pointer is null.
<i>std::invalid_argument</i>	if the <i>connection_pool</i> pointer is null.
<i>std::invalid_argument</i>	if the <i>connection_factory</i> pointer is null.

Definition at line 71 of file [UDPIface.cpp](#).

15.41.2.2 ~UDPIface()

```
UDPIface::~UDPIface ( ) [default]
```

15.41.3 Member Function Documentation

15.41.3.1 print_()

```
std::ostream & UDPIface::print_ (
    std::ostream & os ) const [final], [protected], [virtual]
```

Print the interface to the given output stream.

Parameters

<i>os</i>	The output stream to print to.
-----------	--------------------------------

Returns

The output stream.

Example:

```
udp {
    port 14500;
    address 127.0.0.1;
}
```

Implements [Interface](#).

Definition at line 180 of file [UDPIface.cpp](#).

15.41.3.2 receive_packet()

```
void UDPIface::receive_packet (
    const std::chrono::nanoseconds & timeout ) [final], [virtual]
```

Receive a packet on the interface.

Parameters

<i>timeout</i>	The maximum amount of time to wait for incoming data.
----------------	---

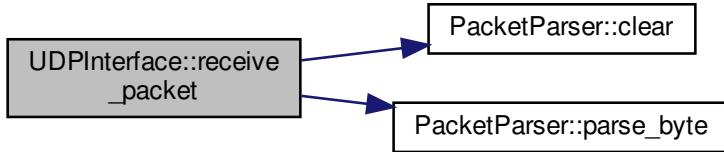
Receives up to one UDP packet worth of data and parses it into MAVLink packets before passing these packets onto the connection pool. Will wait up to *timeout* for a UDP packet to be received.

Implements [Interface](#).

Definition at line 138 of file [UDPIface.cpp](#).

References [PacketParser::clear\(\)](#), and [PacketParser::parse_byte\(\)](#).

Here is the call graph for this function:



15.41.3.3 send_packet()

```
void UDPInterface::send_packet (
    const std::chrono::nanoseconds & timeout ) [final], [virtual]
```

Send a packet from one of the interface's connections.

Note

Which connection to take this packet from is not defined but it must not starve any one of the [Interface](#)'s connections.

Parameters

<i>timeout</i>	The maximum amount of time to wait for a packet to be available for sending.
----------------	--

Sends up to one packet from each connection, belonging to the interface, over the UDP socket.

Implements [Interface](#).

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

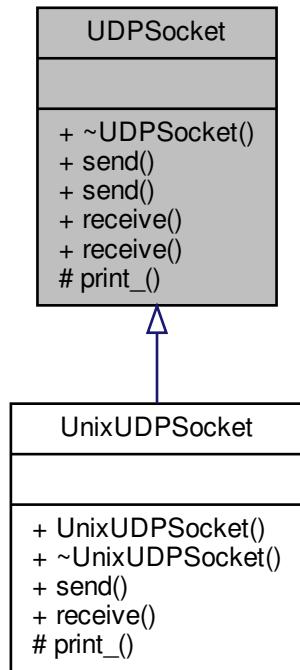
The documentation for this class was generated from the following files:

- [UDPIface.hpp](#)
- [UDPIface.cpp](#)

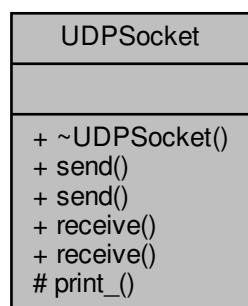
15.42 UDPSocket Class Reference

```
#include <UDPSocket.hpp>
```

Inheritance diagram for UDPSocket:



Collaboration diagram for UDPSocket:



Public Member Functions

- virtual ~UDPSocket ()
- virtual void send (const std::vector< uint8_t > &data, const IPAddress &address)
- virtual void send (std::vector< uint8_t >::const_iterator first, std::vector< uint8_t >::const_iterator last, const IPAddress &address)
- virtual std::pair< std::vector< uint8_t >, IPAddress > receive (const std::chrono::nanoseconds &timeout=std::chrono::nanoseconds::zero())
- virtual IPAddress receive (std::back_insert_iterator< std::vector< uint8_t > > it, const std::chrono::nanoseconds &timeout=std::chrono::nanoseconds::zero())

Protected Member Functions

- virtual std::ostream & print_ (std::ostream &os) const

Friends

- std::ostream & operator<< (std::ostream &os, const UDPSocket &udp_socket)

15.42.1 Detailed Description

A UDP socket, listening on a port/address combination.

Warning

This class should be treated as pure virtual and should never be instantiated.

Either `send(const std::vector<uint8_t> &data, const IPAddress &)` or `send(std::vector<uint8_t>::const_iterator, std::vector<uint8_t>::const_iterator, const IPAddress &)` must be overridden in child classes to avoid infinite recursion.

Either `receive(const std::chrono::nanoseconds &)` or `receive(std::back_insert_iterator<std::vector<uint8_t> >, const std::chrono::nanoseconds &)` must be overridden in child classes to avoid infinite recursion.

Definition at line 46 of file [UDPSocket.hpp](#).

15.42.2 Constructor & Destructor Documentation

15.42.2.1 ~UDPSocket()

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

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

15.42.3 Member Function Documentation

15.42.3.1 print_()

```
std::ostream & UDPSocket::print_ (
    std::ostream & os ) const [protected], [virtual]
```

Print the UDP socket to the given output stream.

Parameters

<code>os</code>	The output stream to print to.
-----------------	--------------------------------

Returns

The output stream.

Reimplemented in [UnixUDPSocket](#).

Definition at line 112 of file [UDPSocket.cpp](#).

15.42.3.2 receive() [1/2]

```
std::pair< std::vector< uint8_t >, IPAddress > UDPSocket::receive (
    const std::chrono::nanoseconds & timeout = std::chrono::nanoseconds::zero() ) [virtual]
```

Receive data on the socket.

Note

The `timeout` is not guaranteed to be up to nanosecond precision, the actual precision is up to the operating system's implementation but is guaranteed to have at least millisecond precision.

Parameters

<code>timeout</code>	How long to wait for data to arrive on the socket. The default is to not wait.
----------------------	--

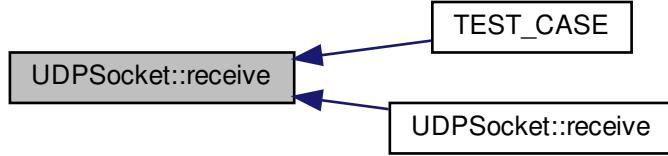
Returns

The data read from the socket and the IP address it was sent from.

Reimplemented in [UnixUDPSocket](#).

Definition at line 77 of file [UDPSocket.cpp](#).

Here is the caller graph for this function:



15.42.3.3 receive() [2/2]

```
IPAddress UDPSocket::receive (
    std::back_insert_iterator< std::vector< uint8_t >> it,
    const std::chrono::nanoseconds & timeout = std::chrono::nanoseconds::zero() ) [virtual]
```

Receive data on the socket.

Note

The `timeout` is not guaranteed to be up to nanosecond precision, the actual precision is up to the operating system's implementation but is guaranteed to have at least millisecond precision.

Parameters

<code>it</code>	A back insert iterator to read bytes into.
<code>timeout</code>	How long to wait for data to arrive on the socket. The default is to not wait.

Returns

The IP address the data was sent from, this is where a reply should be sent to.

Definition at line 97 of file [UDPSocket.cpp](#).

References [receive\(\)](#).

Here is the call graph for this function:



15.42.3.4 send() [1/2]

```
void UDPSocket::send (
    const std::vector< uint8_t > & data,
    const IPAddress & address ) [virtual]
```

Send data to the given address using the socket.

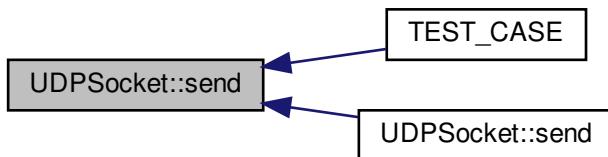
Parameters

<i>data</i>	The bytes to send.
<i>address</i>	The IP address (with port number) to send the bytes to, using UDP.

Reimplemented in [UnixUDPSocket](#).

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

Here is the caller graph for this function:



15.42.3.5 `send()` [2/2]

```
void UDPSocket::send (
    std::vector< uint8_t >::const_iterator first,
    std::vector< uint8_t >::const_iterator last,
    const IPAddress & address ) [virtual]
```

Send data to the given address using the socket.

Parameters

<i>first</i>	Iterator to first byte in range to send.
<i>last</i>	Iterator to one past the last byte to send.
<i>address</i>	The IP address (with port number) to send the bytes to, using UDP.

Definition at line 56 of file [UDPSocket.cpp](#).

References [send\(\)](#).

Here is the call graph for this function:



15.42.4 Friends And Related Function Documentation

15.42.4.1 `operator<<`

```
std::ostream & operator<< (
    std::ostream & os,
    const UDPSocket & udp_socket ) [friend]
```

Print the given UDP socket to the given output stream.

An example:

```
udp {
    port 14555;
    address 127.0.0.1;
    max_bitrate 262144;
}
```

The base [UDPSocket](#) class will print:

```
unknown UDP socket
```

Parameters

<i>os</i>	The output stream to print to.
<i>udp_socket</i>	The UDP socket (or any child of UDPSocket) to print.

Returns

The output stream.

Definition at line 140 of file [UDPSocket.cpp](#).

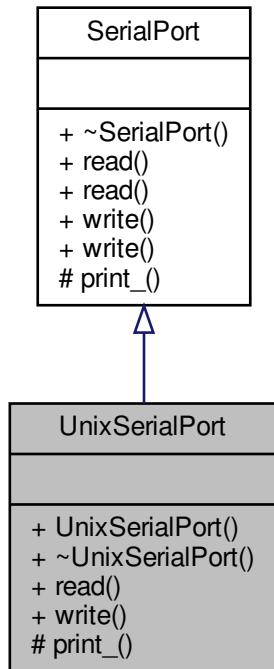
The documentation for this class was generated from the following files:

- [UDPSocket.hpp](#)
- [UDPSocket.cpp](#)

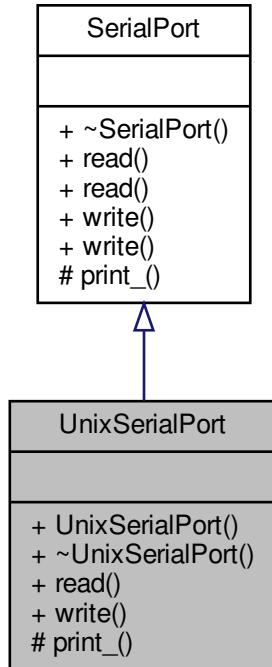
15.43 UnixSerialPort Class Reference

```
#include <UnixSerialPort.hpp>
```

Inheritance diagram for UnixSerialPort:



Collaboration diagram for UnixSerialPort:



Public Types

- enum `Parity` {
 `NONE`, `ODD`, `EVEN`, `MARK`,
`SPACE` }
- enum `Feature` { `DEFAULT` = 0, `FLOW_CONTROL` = 1 << 0 }

Public Member Functions

- `UnixSerialPort` (`std::string device, unsigned long baud_rate=9600, SerialPort::Feature features=SerialPort::DEFAULT, std::unique_ptr< UnixSyscalls > syscalls=std::make_unique< UnixSyscalls >()`)
- virtual `~UnixSerialPort ()`
- virtual `std::vector< uint8_t > read` (`const std::chrono::nanoseconds &timeout=std::chrono::nanoseconds::zero()`)
`final`
- virtual `void write` (`const std::vector< uint8_t > &data`) `final`
- virtual `void read` (`std::back_insert_iterator< std::vector< uint8_t > > it, const std::chrono::nanoseconds &timeout=std::chrono::nanoseconds::zero()`)
- virtual `void write` (`std::vector< uint8_t >::const_iterator first, std::vector< uint8_t >::const_iterator last`)

Protected Member Functions

- std::ostream & [print_](#) (std::ostream &os) const final

15.43.1 Detailed Description

A unix serial port.

Definition at line 34 of file [UnixSerialPort.hpp](#).

15.43.2 Member Enumeration Documentation

15.43.2.1 Feature

```
enum SerialPort::Feature [inherited]
```

Feature bitflags.

Enumerator

DEFAULT	No special features.
FLOW_CONTROL	Enable flow control.

Definition at line 59 of file [SerialPort.hpp](#).

15.43.2.2 Parity

```
enum SerialPort::Parity [inherited]
```

Parity options.

Enumerator

NONE	No parity.
ODD	Odd parity, must have odd number of set bits.
EVEN	Even parity, must have even number of set bits.
MARK	Fill parity bit with 1.
SPACE	Fill parity bit with 0.

Definition at line 49 of file [SerialPort.hpp](#).

15.43.3 Constructor & Destructor Documentation

15.43.3.1 UnixSerialPort()

```
UnixSerialPort::UnixSerialPort (
    std::string device,
    unsigned long baud_rate = 9600,
    SerialPort::Feature features = SerialPort::DEFAULT,
    std::unique_ptr< UnixSyscalls > syscalls = std::make_unique<UnixSyscalls>() )
```

Construct a serial port.

Parameters

<i>device</i>	The string representing the serial port. For example "/dev/ttyUSB0".
<i>baud_rate</i>	The baud rate in bits per second, the default value is 9600 bps.
<i>features</i>	A bitflag of the features to enable, default is to not enable any features. See SerialPort::Feature for flags.
<i>syscalls</i>	The object to use for unix system calls. It is default constructed to the production implementation. This argument is only used for testing.

Exceptions

<i>std::invalid_argument</i>	if the baud rate is not supported.
<i>std::system_error</i>	if a system call produces an error.

Definition at line 47 of file [UnixSerialPort.cpp](#).

15.43.3.2 ~UnixSerialPort()

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

The serial port destructor.

Closes the underlying file descriptor of the serial port device.

Definition at line 64 of file [UnixSerialPort.cpp](#).

15.43.4 Member Function Documentation

15.43.4.1 print_()

```
std::ostream & UnixSerialPort::print_ (
    std::ostream & os ) const [final], [protected], [virtual]
```

Print the serial port to the given output stream.

Parameters

<code>os</code>	The output stream to print to.
-----------------	--------------------------------

Returns

The output stream.

Example:

```
serial {
    device /dev/ttyUSB0;
    baudrate 115200;
    flow_control yes;
}
```

Parameters

<code>os</code>	The output stream to print to.
-----------------	--------------------------------

Reimplemented from [SerialPort](#).

Definition at line 360 of file [UnixSerialPort.cpp](#).

References [SerialPort::FLOW_CONTROL](#).

15.43.4.2 read() [1/2]

```
std::vector< uint8_t > UnixSerialPort::read (
    const std::chrono::nanoseconds & timeout = std::chrono::nanoseconds::zero() ) [final],
[virtual]
```

Read data from the serial port.

Note

The `timeout` is not guaranteed to be up to nanosecond precision, the actual precision is up to the operating system's implementation but is guaranteed to have at least millisecond precision.

Parameters

<i>timeout</i>	How long to wait for data to arrive on the serial port if there is not already data to read. The default is to not wait.
----------------	--

Returns

The data read from the serial port.

Note

The timeout precision of this implementation is 1 millisecond.

Exceptions

<i>std::system_error</i>	if a system call produces an error.
--------------------------	-------------------------------------

Reimplemented from [SerialPort](#).

Definition at line [77](#) of file [UnixSerialPort.cpp](#).

15.43.4.3 read() [2/2]

```
void SerialPort::read (
    std::back_insert_iterator< std::vector< uint8_t >> it,
    const std::chrono::nanoseconds & timeout = std::chrono::nanoseconds::zero() ) [virtual],
[inherited]
```

Read data from the serial port.

Note

The *timeout* is not guaranteed to be up to nanosecond precision, the actual precision is up to the operating system's implementation but is guaranteed to have at least millisecond precision.

Parameters

<i>it</i>	A back insert iterator to read bytes into.
<i>timeout</i>	How long to wait for data to arrive on the serial port if there is not already data to read. The default is to not wait.

Definition at line [65](#) of file [SerialPort.cpp](#).

References [SerialPort::read\(\)](#).

Here is the call graph for this function:



15.43.4.4 write() [1/2]

```
void UnixSerialPort::write (
    const std::vector< uint8_t > & data ) [final], [virtual]
```

Write data to the serial port (blocking write).

Parameters

<i>data</i>	The bytes to send.
-------------	--------------------

Exceptions

<i>std::system_error</i>	if a system call produces an error.
<i>PartialSendError</i>	if it fails to write all the data it is given.

Reimplemented from [SerialPort](#).

Definition at line 118 of file [UnixSerialPort.cpp](#).

15.43.4.5 write() [2/2]

```
void SerialPort::write (
    std::vector< uint8_t >::const_iterator first,
    std::vector< uint8_t >::const_iterator last ) [virtual], [inherited]
```

Write data to the serial port (blocking write).

Parameters

<i>first</i>	Iterator to first byte in range to send.
<i>last</i>	Iterator to one past the last byte to send.

Definition at line 89 of file [SerialPort.cpp](#).

References [SerialPort::write\(\)](#).

Here is the call graph for this function:



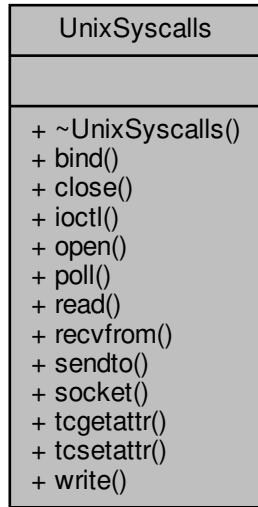
The documentation for this class was generated from the following files:

- [UnixSerialPort.hpp](#)
- [UnixSerialPort.cpp](#)

15.44 UnixSyscalls Class Reference

```
#include <UnixSyscalls.hpp>
```

Collaboration diagram for UnixSyscalls:



Public Member Functions

- TEST_VIRTUAL `~UnixSyscalls ()=default`
- TEST_VIRTUAL int `bind` (int sockfd, const struct sockaddr *addr, socklen_t addrlen)
- TEST_VIRTUAL int `close` (int fd)
- TEST_VIRTUAL int `ioctl` (int fd, unsigned long request, void *argp)
- TEST_VIRTUAL int `open` (const char *pathname, int flags)
- TEST_VIRTUAL int `poll` (struct pollfd *fds, nfds_t nfds, int timeout)
- TEST_VIRTUAL ssize_t `read` (int fd, void *buf, size_t count)
- TEST_VIRTUAL ssize_t `recvfrom` (int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen)
- TEST_VIRTUAL ssize_t `sendto` (int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen)
- TEST_VIRTUAL int `socket` (int domain, int type, int protocol)
- TEST_VIRTUAL int `tcgetattr` (int fd, struct termios *termios_p)
- TEST_VIRTUAL int `tcsetattr` (int fd, int optional_actions, const struct termios *termios_p)
- TEST_VIRTUAL ssize_t `write` (int fd, const void *buf, size_t count)

15.44.1 Detailed Description

A thin wrapper around Unix system calls.

The purpose of this is to allow system calls to be mocked during testing.

See the following man pages for documentation:

- `man 2 bind`
- `man 2 close`
- `man 2 socket`
- `man 2 ioctl`
- `man 7 ip`
- `man 2 open`
- `man 2 poll`
- `man 2 read`
- `man 2 recvfrom`
- `man 2 sendto`
- `man 2 termios`
- `man 2 write`

Definition at line 54 of file [UnixSyscalls.hpp](#).

15.44.2 Constructor & Destructor Documentation

15.44.2.1 `~UnixSyscalls()`

```
TEST_VIRTUAL UnixSyscalls::~UnixSyscalls ( ) [default]
```

15.44.3 Member Function Documentation

15.44.3.1 `bind()`

```
int UnixSyscalls::bind (
    int sockfd,
    const struct sockaddr * addr,
    socklen_t addrlen )
```

Bind a name to a socket.

See `man 2 bind` for documentation.

Parameters

<i>sockfd</i>	Socket file descriptor.
<i>addr</i>	Address to assign to socket.
<i>addrlen</i>	Size of address structure in bytes.

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

15.44.3.2 close()

```
int UnixSyscalls::close (
    int fd )
```

Close a file descriptor.

See [man 2 close](#) for documentation.

Parameters

<i>fd</i>	The file descriptor to close.
-----------	-------------------------------

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

15.44.3.3 ioctl()

```
int UnixSyscalls::ioctl (
    int fd,
    unsigned long request,
    void * argp )
```

Control device.

See [man 2 ioctl](#) for documentation.

Parameters

<i>fd</i>	The file descriptor to control.
<i>request</i>	Request code, defined in <sys/ioctl.h>
<i>argp</i>	Pointer to input/output, dependent on request code.

Definition at line 69 of file [UnixSyscalls.cpp](#).

15.44.3.4 open()

```
int UnixSyscalls::open (
    const char * pathname,
    int flags )
```

Open and possibly create a file.

See [man 2 open](#) for documentation.

Definition at line 80 of file [UnixSyscalls.cpp](#).

15.44.3.5 poll()

```
int UnixSyscalls::poll (
    struct pollfd * fds,
    nfds_t nfds,
    int timeout )
```

Wait for some event on a file descriptor.

See [man 2 poll](#) for documentation.

Parameters

<i>fds</i>	File descriptor event structures.
<i>nfds</i>	Number of file descriptor event structures.
<i>timeout</i>	The timeout in milliseconds.

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

15.44.3.6 read()

```
ssize_t UnixSyscalls::read (
    int fd,
    void * buf,
    size_t count )
```

Read from a file descriptor.

See [man 2 read](#) for documentation.

Definition at line 106 of file [UnixSyscalls.cpp](#).

15.44.3.7 recvfrom()

```
ssize_t UnixSyscalls::recvfrom (
    int sockfd,
    void * buf,
    size_t len,
    int flags,
    struct sockaddr * src_addr,
    socklen_t * addrlen )
```

Receive a message from a socket.

See [man 2 recvfrom](#) for documentation.

Parameters

<i>sockfd</i>	Socket file descriptor to receive data on.
<i>buf</i>	The buffer to write the data into.
<i>len</i>	The length of the buffer.
<i>flags</i>	Option flags.
<i>src_addr</i>	Source address buffer.
<i>addrlen</i>	Before call, length of source address buffer. After call, actual length of address data.

Returns

The number of bytes written to or -1 if an error occurred.

Definition at line 126 of file [UnixSyscalls.cpp](#).

15.44.3.8 sendto()

```
ssize_t UnixSyscalls::sendto (
    int sockfd,
    const void * buf,
    size_t len,
    int flags,
    const struct sockaddr * dest_addr,
    socklen_t addrlen )
```

Send a message on a socket.

See [man 2 sendto](#) for documentation.

Definition at line 139 of file [UnixSyscalls.cpp](#).

15.44.3.9 socket()

```
int UnixSyscalls::socket (
    int domain,
    int type,
    int protocol )
```

Create an endpoint for communication.

See [man 2 socket](#) for documentation.

Parameters

<i>domain</i>	Protocol family to use for communication.
<i>type</i>	Socket type.
<i>protocol</i>	Protocol to use for socket.

Definition at line 156 of file [UnixSyscalls.cpp](#).

15.44.3.10 tcgetattr()

```
int UnixSyscalls::tcgetattr (
    int fd,
    struct termios * termios_p )
```

Get serial port parameters associated with the given file descriptor.

See [man 2 termios](#) for documentation.

Definition at line 167 of file [UnixSyscalls.cpp](#).

15.44.3.11 tcsetattr()

```
int UnixSyscalls::tcsetattr (
    int fd,
    int optional_actions,
    const struct termios * termios_p )
```

Set serial port parameters for given file descriptor.

See [man 2 termios](#) for documentation.

Definition at line 178 of file [UnixSyscalls.cpp](#).

15.44.3.12 write()

```
ssize_t UnixSyscalls::write (
    int fd,
    const void * buf,
    size_t count )
```

Write to a file descriptor.

See [man 2 write](#) for documentation.

Definition at line 190 of file [UnixSyscalls.cpp](#).

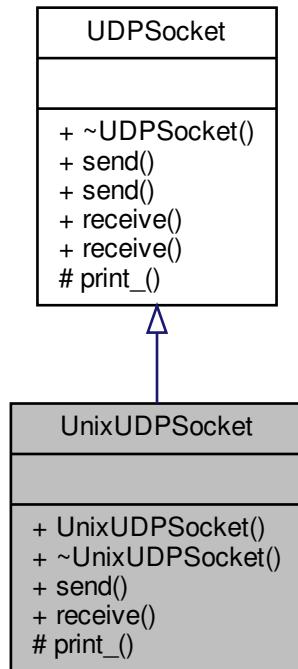
The documentation for this class was generated from the following files:

- [UnixSyscalls.hpp](#)
- [UnixSyscalls.cpp](#)

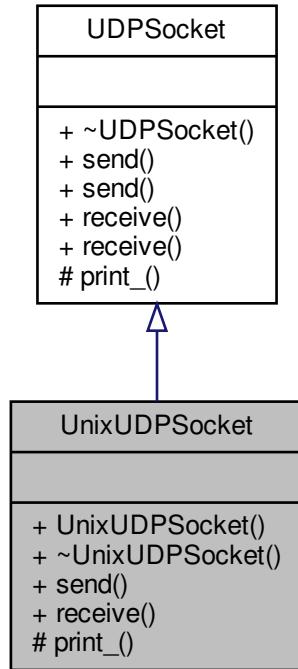
15.45 UnixUDPSocket Class Reference

```
#include <UnixUDPSocket.hpp>
```

Inheritance diagram for UnixUDPSocket:



Collaboration diagram for UnixUDPSocket:



Public Member Functions

- `UnixUDPSocket (unsigned int port, std::optional< IPAddress > address={}, unsigned long max_bitrate=0, std::unique_ptr< UnixSyscalls > syscalls=std::make_unique< UnixSyscalls >())`
- `virtual ~UnixUDPSocket ()`
- `virtual void send (const std::vector< uint8_t > &data, const IPAddress &address) final`
- `virtual std::pair< std::vector< uint8_t >, IPAddress > receive (const std::chrono::nanoseconds &timeout=std::chrono::nanoseconds::zero()) final`
- `virtual void send (std::vector< uint8_t >::const_iterator first, std::vector< uint8_t >::const_iterator last, const IPAddress &address)`
- `virtual IPAddress receive (std::back_insert_iterator< std::vector< uint8_t > > it, const std::chrono::nanoseconds &timeout=std::chrono::nanoseconds::zero())`

Protected Member Functions

- `std::ostream & print_ (std::ostream &os) const final`

15.45.1 Detailed Description

A unix UDP socket, listening on a port/address combination.

Definition at line 36 of file [UnixUDPSocket.hpp](#).

15.45.2 Constructor & Destructor Documentation

15.45.2.1 UnixUDPSocket()

```
UnixUDPSocket::UnixUDPSocket (
    unsigned int port,
    std::optional< IPAddress > address = {},
    unsigned long max_bitrate = 0,
    std::unique_ptr< UnixSyscalls > syscalls = std::make_unique<UnixSyscalls>() )
```

Construct a UDP socket.

Parameters

<i>port</i>	The port number to listen on.
<i>address</i>	The address to listen on (the port portion of the address is ignored). The default is to listen on any address.
<i>max_bitrate</i>	The maximum number of bits per second to transmit on the UDP interface. The default is 0, which indicates no limit.
<i>syscalls</i>	The object to use for unix system calls. It is default constructed to the production implementation. This argument is only used for testing.

Exceptions

<code>std::system_error</code>	if a system call produces an error.
--------------------------------	-------------------------------------

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

15.45.2.2 ~UnixUDPSocket()

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

The socket destructor.

Closes the underlying file descriptor of the UDP socket.

Definition at line 70 of file [UnixUDPSocket.cpp](#).

15.45.3 Member Function Documentation

15.45.3.1 print_()

```
std::ostream & UnixUDPSocket::print_ (
    std::ostream & os ) const [final], [protected], [virtual]
```

Print the UDP socket to the given output stream.

Parameters

<code>os</code>	The output stream to print to.
-----------------	--------------------------------

Returns

The output stream.

An example:

```
udp {
    port 14555;
    address 127.0.0.1;
    max_bitrate 262144;
}
```

Parameters

<code>os</code>	The output stream to print to.
-----------------	--------------------------------

Reimplemented from [UDPSocket](#).

Definition at line [257](#) of file [UnixUDPSocket.cpp](#).

15.45.3.2 receive() [1/2]

```
std::pair< std::vector< uint8_t >, IPAddress > UnixUDPSocket::receive (
    const std::chrono::nanoseconds & timeout = std::chrono::nanoseconds::zero() ) [final],
[virtual]
```

Receive data on the socket.

Note

The `timeout` is not guaranteed to be up to nanosecond precision, the actual precision is up to the operating system's implementation but is guaranteed to have at least millisecond precision.

Parameters

<i>timeout</i>	How long to wait for data to arrive on the socket. The default is to not wait.
----------------	--

Returns

The data read from the socket and the IP address it was sent from.

Note

The timeout precision of this implementation is 1 millisecond.

Exceptions

<i>std::system_error</i>	if a system call produces an error.
--------------------------	-------------------------------------

Reimplemented from [UDPSocket](#).

Definition at line 122 of file [UnixUDPSocket.cpp](#).

15.45.3.3 receive() [2/2]

```
IPAddress UDPSocket::receive (
    std::back_insert_iterator< std::vector< uint8_t >> it,
    const std::chrono::nanoseconds & timeout = std::chrono::nanoseconds::zero() ) [virtual],
[inherited]
```

Receive data on the socket.

Note

The *timeout* is not guaranteed to be up to nanosecond precision, the actual precision is up to the operating system's implementation but is guaranteed to have at least millisecond precision.

Parameters

<i>it</i>	A back insert iterator to read bytes into.
<i>timeout</i>	How long to wait for data to arrive on the socket. The default is to not wait.

Returns

The IP address the data was sent from, this is where a reply should be sent to.

Definition at line 97 of file [UDPSocket.cpp](#).

References [UDPSocket::receive\(\)](#).

Here is the call graph for this function:



15.45.3.4 send() [1/2]

```
void UnixUDPSocket::send (
    const std::vector< uint8_t > & data,
    const IPAddress & address ) [final], [virtual]
```

Send data to the given address using the socket.

Parameters

<i>data</i>	The bytes to send.
<i>address</i>	The IP address (with port number) to send the bytes to, using UDP.

Exceptions

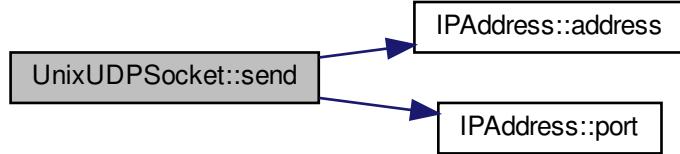
[PartialSendError](#) if it fails to write all the data it is given.

Reimplemented from [UDPSocket](#).

Definition at line 81 of file [UnixUDPSocket.cpp](#).

References [IPAddress::address\(\)](#), and [IPAddress::port\(\)](#).

Here is the call graph for this function:



15.45.3.5 `send()` [2/2]

```
void UDPSocket::send (
    std::vector< uint8_t >::const_iterator first,
    std::vector< uint8_t >::const_iterator last,
    const IPAddress & address ) [virtual], [inherited]
```

Send data to the given address using the socket.

Parameters

<i>first</i>	Iterator to first byte in range to send.
<i>last</i>	Iterator to one past the last byte to send.
<i>address</i>	The IP address (with port number) to send the bytes to, using UDP.

Definition at line 56 of file [UDPSocket.cpp](#).

References [UDPSocket::send\(\)](#).

Here is the call graph for this function:



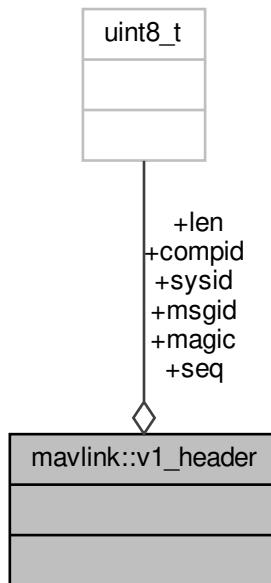
The documentation for this class was generated from the following files:

- [UnixUDPSocket.hpp](#)
- [UnixUDPSocket.cpp](#)

15.46 mavlink::v1_header Struct Reference

```
#include <mavlink.hpp>
```

Collaboration diagram for mavlink::v1_header:



Public Attributes

- `uint8_t magic`
Protocol magic marker (0xFE).
- `uint8_t len`
Length of payload.
- `uint8_t seq`
Sequence of packet.
- `uint8_t sysid`
ID of message sender system/aircraft.
- `uint8_t compid`
ID of the message sender component.
- `uint8_t msgid`
ID of message in payload.

15.46.1 Detailed Description

MAVLink packet v1.0 header.

Definition at line [56](#) of file [mavlink.hpp](#).

15.46.2 Member Data Documentation

15.46.2.1 compid

```
uint8_t mavlink::v1_header::compid
```

ID of the message sneder component.

Definition at line [62](#) of file [mavlink.hpp](#).

15.46.2.2 len

```
uint8_t mavlink::v1_header::len
```

Length of payload.

Definition at line [59](#) of file [mavlink.hpp](#).

15.46.2.3 magic

```
uint8_t mavlink::v1_header::magic
```

Protocol magic marker (0xFE).

Definition at line [58](#) of file [mavlink.hpp](#).

15.46.2.4 msgid

```
uint8_t mavlink::v1_header::msgid
```

ID of message in payload.

Definition at line [63](#) of file [mavlink.hpp](#).

15.46.2.5 seq

```
uint8_t mavlink::v1_header::seq
```

Sequence of packet.

Definition at line 60 of file [mavlink.hpp](#).

15.46.2.6 sysid

```
uint8_t mavlink::v1_header::sysid
```

ID of message sender system/aircraft.

Definition at line 61 of file [mavlink.hpp](#).

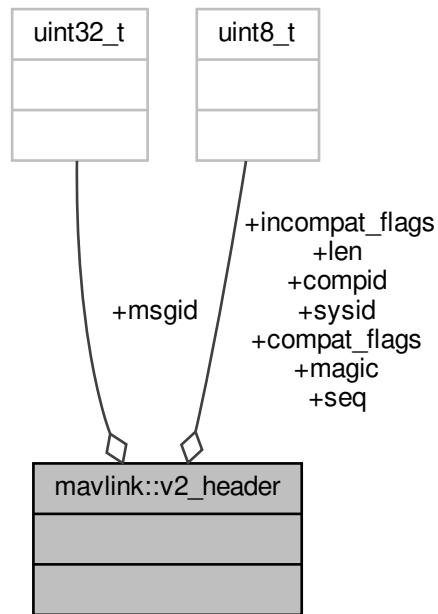
The documentation for this struct was generated from the following file:

- [mavlink.hpp](#)

15.47 mavlink::v2_header Struct Reference

```
#include <mavlink.hpp>
```

Collaboration diagram for mavlink::v2_header:



Public Attributes

- `uint8_t magic`
Protocol magic marker (0xFD).
- `uint8_t len`
Length of payload.
- `uint8_t incompat_flags`
Flags that must be understood.
- `uint8_t compat_flags`
Flags that can be ignored if not known.
- `uint8_t seq`
Sequence of packet.
- `uint8_t sysid`
ID of message sender system/aircraft.
- `uint8_t compid`
ID of the message sender component.
- `uint32_t msgid`: 24
ID of message in payload (3 bytes).

15.47.1 Detailed Description

MAVLink packet v2.0 header.

Definition at line 71 of file [mavlink.hpp](#).

15.47.2 Member Data Documentation

15.47.2.1 compat_flags

```
uint8_t mavlink::v2_header::compat_flags
```

Flags that can be ignored if not known.

Definition at line 76 of file [mavlink.hpp](#).

15.47.2.2 compid

```
uint8_t mavlink::v2_header::compid
```

ID of the message sender component.

Definition at line 79 of file [mavlink.hpp](#).

15.47.2.3 incompat_flags

```
uint8_t mavlink::v2_header::incompat_flags
```

Flags that must be understood.

Definition at line 75 of file [mavlink.hpp](#).

15.47.2.4 len

```
uint8_t mavlink::v2_header::len
```

Length of payload.

Definition at line 74 of file [mavlink.hpp](#).

15.47.2.5 magic

```
uint8_t mavlink::v2_header::magic
```

Protocol magic marker (0xFD).

Definition at line 73 of file [mavlink.hpp](#).

15.47.2.6 msgid

```
uint32_t mavlink::v2_header::msgid
```

ID of message in payload (3 bytes).

Definition at line 80 of file [mavlink.hpp](#).

15.47.2.7 seq

```
uint8_t mavlink::v2_header::seq
```

Sequence of packet.

Definition at line 77 of file [mavlink.hpp](#).

15.47.2.8 sysid

```
uint8_t mavlink::v2_header::sysid
```

ID of message sender system/aircraft.

Definition at line 78 of file [mavlink.hpp](#).

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

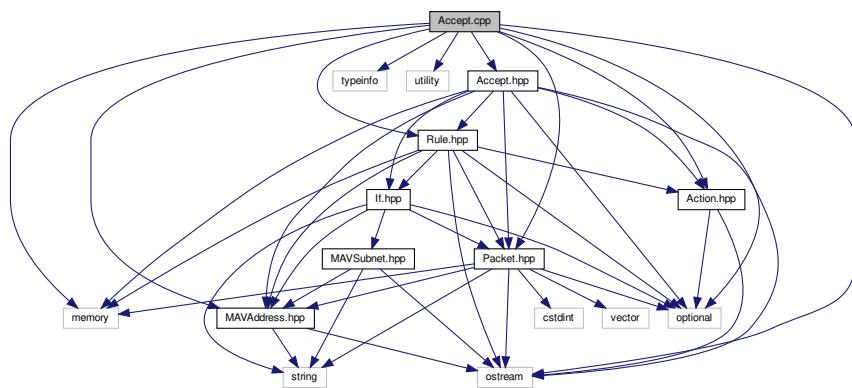
- [mavlink.hpp](#)

Chapter 16

File Documentation

16.1 Accept.cpp File Reference

```
#include <memory>
#include <optional>
#include <iostream>
#include <typeinfo>
#include <utility>
#include "Accept.hpp"
#include "Action.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "Rule.hpp"
Include dependency graph for Accept.cpp:
```



16.2 Accept.cpp

```
00001 // MAVLink router and firewall.
```

```

00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <memory>
00019 #include <optional>
00020 #include <ostream>
00021 #include <typeinfo>
00022 #include <utility>
00023
00024 #include "Accept.hpp"
00025 #include "Action.hpp"
00026 #include "MAVAddress.hpp"
00027 #include "Packet.hpp"
00028 #include "Rule.hpp"
00029
00030
00031 /** Construct an accept rule, without a priority.
00032 *
00033 * An accept rule is used to accept packet/address combinations that match the
00034 * condition of the rule.
00035 *
00036 * \param condition The condition used to determine the rule matches a
00037 * particular packet/address combination given to the \ref action method.
00038 * The default is {} which indicates the rule matches any packet/address
00039 * combination.
00040 * \sa action
00041 */
00042 Accept::Accept(std::optional<If> condition)
00043     : Rule(std::move(condition))
00044 {
00045 }
00046
00047
00048 /** Construct an accept rule, with a priority.
00049 *
00050 * An accept rule is used to accept packet/address combinations that match the
00051 * condition of the rule.
00052 *
00053 * \param condition The condition used to determine the rule matches a
00054 * particular packet/address combination given to the \ref action method.
00055 * The default is {} which indicates the rule matches any packet/address
00056 * combination.
00057 * \param priority The priority to accept packets with. A higher number is
00058 * more important and will be routed first.
00059 * \sa action
00060 */
00061 Accept::Accept(int priority, std::optional<If> condition)
00062     : Rule(std::move(condition)), priority_(priority)
00063 {
00064 }
00065
00066
00067 /** \copydoc Rule::print_(std::ostream &os) const
00068 *
00069 * Prints '"reject <Priority> <If Statement>"' where the priority and if
00070 * statement are only printed if the priority or condition is set,
00071 * respectively.
00072 *
00073 * \param os The output stream to print to.
00074 */
00075 std::ostream &Accept::print_(std::ostream &os) const
00076 {
00077     os << "accept";
00078
00079     if (priority_)
00080     {
00081         os << " with priority " << priority_.value();
00082     }

```

```

00083
00084     if (condition_)
00085     {
00086         os << " " << condition_.value();
00087     }
00088
00089     return os;
00090 }
00091
00092
00093 /** \copydoc Rule::action(const Packet&,const MAVAddress&) const
00094 *
00095 *  If the condition has not been set or it matches the given packet/address
00096 *  combination then it will return the accept \ref Action object (with optional
00097 *  priority), otherwise it will return the continue \ref Action object.
00098 */
00099 Action Accept::action(
00100     const Packet &packet, const MAVAddress &address) const
00101 {
00102     if (!condition_ || condition_->check(packet, address))
00103     {
00104         return Action::make_accept(priority_);
00105     }
00106
00107     return Action::make_continue();
00108 }
00109
00110
00111 std::unique_ptr<Rule> Accept::clone() const
00112 {
00113     if (priority_)
00114     {
00115         return std::make_unique<Accept>(priority_.value(), condition_);
00116     }
00117
00118     return std::make_unique<Accept>(condition_);
00119 }
00120
00121
00122 /** \copydoc Rule::operator==(const Rule&) const
00123 *
00124 *  Compares the priority (if set) associated with the rule as well.
00125 */
00126 bool Accept::operator==(const Rule &other) const
00127 {
00128     return typeid(*this) == typeid(other) &&
00129         priority_ == static_cast<const Accept &>(other).priority_ &&
00130         condition_ == static_cast<const Accept &>(other).
00131         condition_;
00132
00133
00134 /** \copydoc Rule::operator!=(const Rule&) const
00135 *
00136 *  Compares the priority (if set) associated with the rule as well.
00137 */
00138 bool Accept::operator!=(const Rule &other) const
00139 {
00140     return typeid(*this) != typeid(other) ||
00141         priority_ != static_cast<const Accept &>(other).priority_ ||
00142         condition_ != static_cast<const Accept &>(other).
00143         condition_;

```

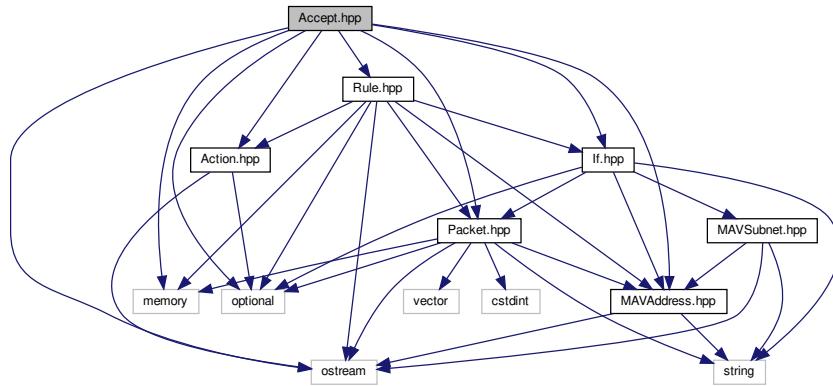
16.3 Accept.hpp File Reference

```

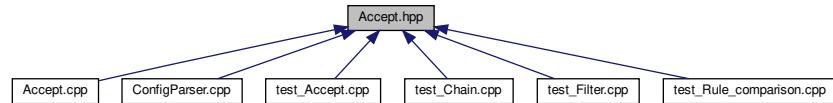
#include <memory>
#include <optional>
#include <iostream>
#include "Action.hpp"
#include "If.hpp"
#include "MAVAddress.hpp"

```

```
#include "Packet.hpp"
#include "Rule.hpp"
Include dependency graph for Accept.hpp:
```



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



Classes

- class [Accept](#)

16.4 Accept.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef ACCEPT_HPP_
```

```

00019 #define ACCEPT_HPP_
00020
00021
00022 #include <memory>
00023 #include <optional>
00024 #include <iostream>
00025
00026 #include "Action.hpp"
00027 #include "If.hpp"
00028 #include "MAVAddress.hpp"
00029 #include "Packet.hpp"
00030 #include "Rule.hpp"
00031
00032
00033 /** Rule to accept a packet, optionally with a priority.
00034 */
00035 class Accept : public Rule
00036 {
00037     public:
00038         Accept(std::optional<If> condition = {});
00039         Accept(int priority, std::optional<If> condition = {});
00040         virtual Action action(
00041             const Packet &packet, const MAVAddress &address) const;
00042         virtual std::unique_ptr<Rule> clone() const;
00043         virtual bool operator==(const Rule &other) const;
00044         virtual bool operator!=(const Rule &other) const;
00045
00046     protected:
00047         virtual std::ostream &print_(std::ostream &os) const;
00048
00049     private:
00050         std::optional<int> priority_;
00051 };
00052
00053
00054 #endif // ACCEPT_HPP_

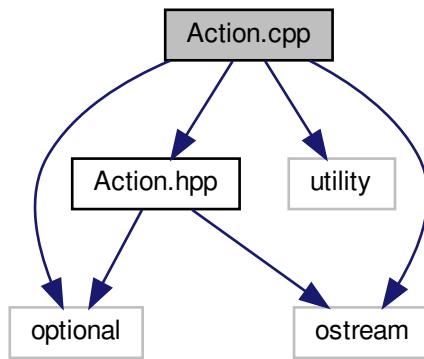
```

16.5 Action.cpp File Reference

```

#include <optional>
#include <iostream>
#include <utility>
#include "Action.hpp"
Include dependency graph for Action.cpp:

```



16.6 Action.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <optional>
00019 #include <iostream>
00020 #include <utility>
00021
00022 #include "Action.hpp"
00023
00024
00025 /** Construct an Action.
00026 *
00027 * \note Private constructor, use \ref make_accept, \ref make_reject, \ref
00028 * make_continue, or \ref make_default.
00029 *
00030 * \param action The action this result represents.
00031 * \param priority The priority, only used with Action::ACCEPT.
00032 * \sa Action::Option
00033 */
00034 Action::Action(
00035     Action::Option action, std::optional<int> priority)
00036     : action_(action), priority_(std::move(priority))
00037 {
00038 }
00039
00040
00041 /** Return the action that has been chosen.
00042 *
00043 * \returns The Action::Option enum associated with this action.
00044 */
00045 Action::Option Action::action() const
00046 {
00047     return action_;
00048 }
00049
00050
00051 /** Set the priority of the action.
00052 *
00053 * This only has an effect if the \ref action is Action::ACCEPT and the
00054 * priority has never been set before.
00055 *
00056 * The default priority is 0. A higher priority will result in the
00057 * packet being prioritized over other packets while a lower (negative)
00058 * priority will de-prioritize the packet.
00059 *
00060 * \param priority The priority to apply to the accept action.
00061 */
00062 void Action::priority(int priority)
00063 {
00064     if (action_ == Action::ACCEPT)
00065     {
00066         if (!priority_)
00067         {
00068             priority_ = priority;
00069         }
00070     }
00071 }
00072
00073
00074 /** Return the priority if it exists.
00075 *
00076 * \returns The priority of the action. This will always be 0 if the \ref
00077 * action is not Action::ACCEPT. It will also be 0 (the default priority)

```

```
00078     *      if the priority has never been set.
00079     */
00080 int Action::priority() const
00081 {
00082     if (priority_)
00083     {
00084         return priority_.value();
00085     }
00086
00087     return 0;
00088 }
00089
00090
00091 /** Make a new action result with the Action::ACCEPT action.
00092 *
00093 * An accept action indicates that the packet/address combination this action
00094 * is the response to should be accepted without any further processing.
00095 *
00096 * \param priority The priority to accept the packet with. The default is to
00097 *      not apply a priority.
00098 * \returns The new 'accept' action.
00099 */
00100 Action Action::make_accept(std::optional<int> priority)
00101 {
00102     return Action(Action::ACCEPT, priority);
00103 }
00104
00105
00106 /** Make a new action result with the Action::REJECT action.
00107 *
00108 * A reject action indicates that the packet/address combination this action is
00109 * the response to should be rejected without any further processing.
00110 *
00111 * \returns The new 'reject' action.
00112 */
00113 Action Action::make_reject()
00114 {
00115     return Action(Action::REJECT);
00116 }
00117
00118
00119 /** Make a new action result with the Action::CONTINUE action.
00120 *
00121 * A continue action indicates that filtering of the packet/address combination
00122 * this action is the response should continue with the next \ref Rule.
00123 *
00124 * \returns The new 'continue' action.
00125 */
00126 Action Action::make_continue()
00127 {
00128     return Action(Action::CONTINUE);
00129 }
00130
00131
00132 /** Make a new action result with the Action::DEFAULT action.
00133 *
00134 * A default action indicates that the default action (defined in \ref Filter)
00135 * should be taken for the packet/address combination this action is the
00136 * response to.
00137 *
00138 * \returns The new 'default' action.
00139 */
00140 Action Action::make_default()
00141 {
00142     return Action(Action::DEFAULT);
00143 }
00144
00145
00146 /** Equality comparison.
00147 *
00148 * \relates Action
00149 * \param lhs The left hand side action.
00150 * \param rhs The right hand side action.
00151 * \retval true if \p lhs is the same as rhs.
00152 * \retval false if \p lhs is not the same as rhs.
00153 */
00154 bool operator==(const Action &lhs, const Action &rhs)
00155 {
00156     return (lhs.action() == rhs.action()) && (lhs.priority() == rhs.
00157     priority());
00157 }
```

```

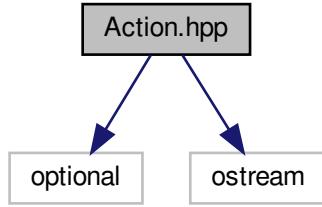
00158
00159
00160 /** Inequality comparison.
00161  * 
00162  * \relates Action
00163  * \param lhs The left hand side action.
00164  * \param rhs The right hand side action.
00165  * \retval true if \p lhs is not the same as rhs.
00166  * \retval false if \p lhs is the same as rhs.
00167  */
00168 bool operator!=(const Action &lhs, const Action &rhs)
00169 {
00170     return (lhs.action() != rhs.action()) || (lhs.priority() != rhs.
00171     priority());
00172
00173
00174 /** Print the action to the given output stream.
00175  * 
00176  * Some examples are:
00177  *     - 'accept'
00178  *     - 'accept with priority 3'
00179  *     - 'reject'
00180  *     - 'continue'
00181  *     - 'default'
00182  * 
00183  * \relates Action
00184  * \param os The output stream to print to.
00185  * \param action The action result to print.
00186  * \returns The output stream.
00187  */
00188 std::ostream &operator<<(std::ostream &os, const Action &action)
00189 {
00190     switch (action.action())
00191     {
00192         case Action::ACCEPT:
00193             os << "accept";
00194
00195             if (action.priority() != 0)
00196             {
00197                 os << " with priority " << action.priority();
00198             }
00199
00200             break;
00201
00202         case Action::REJECT:
00203             os << "reject";
00204             break;
00205
00206         case Action::CONTINUE:
00207             os << "continue";
00208             break;
00209
00210         case Action::DEFAULT:
00211             os << "default";
00212             break;
00213     }
00214
00215     return os;
00216 }

```

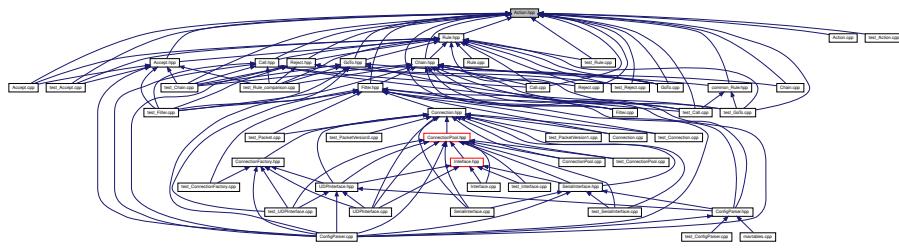
16.7 Action.hpp File Reference

```
#include <optional>
#include <iostream>
```

Include dependency graph for Action.hpp:



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



Classes

- class Action

16.7.1 Function Documentation

16.7.1.1 operator"!=()

```
    bool operator!= ( const Action & lhs, const Action & rhs ) [related]
```

16.7.1.2 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const Action & action ) [related]
```

16.7.1.3 operator==()

```
bool operator== (
    const Action & lhs,
    const Action & rhs ) [related]
```

16.8 Action.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef ACTION_HPP_
00019 #define ACTION_HPP_
00020
00021
00022 #include <optional>
00023 #include <iostream>
00024
00025
00026 /** An action that is to be taken with a packet.
00027 *
00028 * This is used as a return value to determine what to do with a packet.
00029 */
00030 class Action
00031 {
00032     public:
00033         /** Possible actions.
00034         */
00035     enum Option
00036     {
00037         ACCEPT,    //!< The packet has been accepted, possibly with priority.
00038         REJECT,   //!< The packet has been rejected.
00039         CONTINUE,  //!< Continue evaluating rules.
00040         DEFAULT   //!< Use the default rule.
00041     };
00042     // Methods
00043     /** Copy constructor.
00044     *
00045     * \param other Action to copy from.
00046     */
00047     Action(const Action &other) = default;
00048     /** Move constructor.
00049     *
00050     * \param other Action to move from.
```

```

00051      */
00052      Action(Action &&other) = default;
00053      Action::Option action() const;
00054      void priority(int priority);
00055      int priority() const;
00056      /** Assignment operator.
00057      *
00058      * \param other Action to copy from.
00059      */
00060      Action &operator=(const Action &other) = default;
00061      /** Assignment operator (by move semantics).
00062      *
00063      * \param other Action to move from.
00064      */
00065      Action &operator=(Action &&other) = default;
00066
00067      static Action make_accept(std::optional<int> priority = {});
00068      static Action make_reject();
00069      static Action make_continue();
00070      static Action make_default();
00071
00072  private:
00073      Action::Option action_;
00074      // Note: The reason this is optional is because there is a difference
00075      //       between {} and 0. This is because a priority of {} can still
00076      //       be set to something other than 0 by a rule higher up the chain
00077      //       (see \ref Call or \ref GoTo) while if the priority has been set
00078      //       to 0 it should not be set again.
00079      std::optional<int> priority_;
00080      Action(Action::Option action, std::optional<int>
00081      priority = {});
00081  };
00082
00083
00084  bool operator==(const Action &lhs, const Action &rhs);
00085  bool operator!=(const Action &lhs, const Action &rhs);
00086  std::ostream &operator<<(std::ostream &os, const Action &action);
00087
00088
00089 #endif // ACTION_HPP_

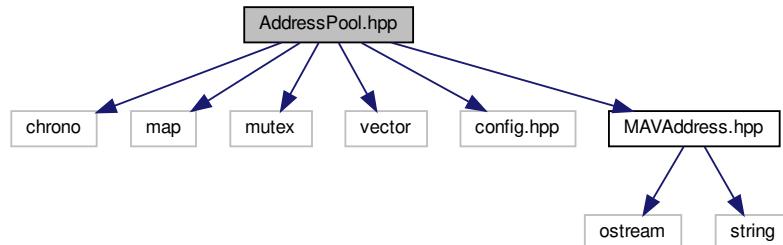
```

16.9 AddressPool.hpp File Reference

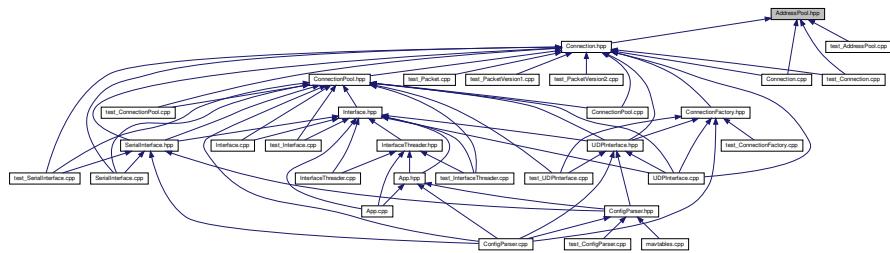
```

#include <chrono>
#include <map>
#include <mutex>
#include <vector>
#include "config.hpp"
#include "MAVAddress.hpp"
Include dependency graph for AddressPool.hpp:

```



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



Classes

- class [AddressPool< TC >](#)

16.10 AddressPool.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef ADDRESSPOOL_HPP_
00019 #define ADDRESSPOOL_HPP_
00020
00021
00022 #include <chrono>
00023 #include <map>
00024 #include <mutex>
00025 #include <vector>
00026
00027 #include "config.hpp"
00028 #include "MAVAddress.hpp"
00029
00030
00031 /** A threadsafe container for addresses that expire after a given time.
00032 */
00033 template <class TC = std::chrono::steady_clock>
00034 class AddressPool
00035 {
00036     public:
00037         AddressPool(std::chrono::milliseconds timeout =
00038                     std::chrono::milliseconds(120000));
00039         // LCOV_EXCL_START
00040         TEST_VIRTUAL ~AddressPool() = default;
00041         // LCOV_EXCL_STOP
00042         TEST_VIRTUAL void add(MAVAddress address);
00043         TEST_VIRTUAL std::vector<MAVAddress> addresses();
00044         TEST_VIRTUAL bool contains(const MAVAddress &address);
00045
00046     private:
00047         std::map<MAVAddress, std::chrono::time_point<TC>> addresses_;

```

```
00048     std::chrono::milliseconds timeout_;
00049     std::mutex mutex_;
00050 };
00051
00052
00053 /** Construct a new address pool.
00054 *
00055 * \param timeout The amount of time (in milliseconds) before a component will
00056 *      be considered offline and removed from the pool, unless its time is
00057 *      updated with \ref add.
00058 */
00059 template <class TC>
00060 AddressPool<TC>::AddressPool(std::chrono::milliseconds timeout)
00061     : timeout_(std::move(timeout))
00062 {
00063 }
00064
00065
00066 /** Add a MAVLink address to the pool.
00067 *
00068 * \note Addresses will be removed after the timeout (set in the
00069 *      constructor) has run out. Re-adding the address (even before this time
00070 *      runs out) will reset the timeout.
00071 *
00072 * \param address The MAVLink address to add or update the timeout for.
00073 * \remarks
00074 *      Threadsafe (locking).
00075 */
00076 template <class TC>
00077 void AddressPool<TC>::add(MAVAddress address)
00078 {
00079     std::lock_guard<std::mutex> lock(mutex_);
00080     addresses_.insert_or_assign(std::move(address), TC::now());
00081 }
00082
00083
00084 /** Get a vector of all the addresses in the pool.
00085 *
00086 * \note A copy is returned instead of using iterators in order to make the
00087 *      call thread safe.
00088 *
00089 * \returns A vector of the addresses in the pool.
00090 * \remarks
00091 *      Threadsafe (locking).
00092 */
00093 template <class TC>
00094 std::vector<MAVAddress> AddressPool<TC>::addresses()
00095 {
00096     std::lock_guard<std::mutex> lock(mutex_);
00097     std::vector<MAVAddress> addresses;
00098     addresses.reserve(addresses_.size());
00099     auto current_time = TC::now();
00100
00101     // Loop over addresses.
00102     for (auto it = addresses_.cbegin(); it != addresses_.cend();)
00103     {
00104         // Remove the address if it has expired.
00105         if ((current_time - it->second) > timeout_)
00106         {
00107             it = addresses_.erase(it);
00108         }
00109         // Store the address.
00110         else
00111         {
00112             addresses.push_back((it++)->first);
00113         }
00114     }
00115
00116     return addresses;
00117 }
00118
00119
00120 /** Determine if the pool contains a given MAVLink address.
00121 *
00122 * \param address The MAVLink address to test for.
00123 * \retval true If the pool contains \p address.
00124 * \retval false If the pool does not contain \p address.
00125 * \remarks
00126 *      Threadsafe (locking).
00127 */
00128 template <class TC>
```

```

00129 bool AddressPool<TC>::contains(const MAVAddress &address)
00130 {
00131     std::lock_guard<std::mutex> lock(mutex_);
00132     auto it = addresses_.find(address);
00133
00134     if (it != addresses_.end())
00135     {
00136         auto current_time = TC::now();
00137
00138         if (current_time - it->second > timeout_)
00139         {
00140             addresses_.erase(it);
00141             return false;
00142         }
00143
00144         return true;
00145     }
00146
00147     return false;
00148 }
00149
00150
00151 #endif // ADDRESSPOOL_HPP_

```

16.11 ansi_codes.sh File Reference

16.12 ansi_codes.sh

```

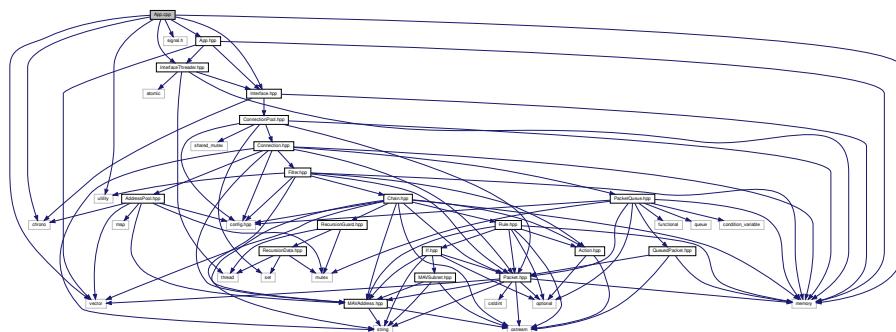
00001#!/bin/bash
00002
00003# Copyright 2017 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00004#
00005# Licensed under the Apache License, Version 2.0 (the "License");
00006# you may not use this file except in compliance with the License.
00007# You may obtain a copy of the License at
00008#
00009# http://www.apache.org/licenses/LICENSE-2.0
00010#
00011# Unless required by applicable law or agreed to in writing, software
00012# distributed under the License is distributed on an "AS IS" BASIS,
00013# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00014# See the License for the specific language governing permissions and
00015# limitations under the License.
00016
00017
00018# BASH library that provides variables for the ANSI escape codes.
00019#
00020# To use this library simply source the file:
00021#
00022#       $ source ansi_codes
00023#
00024# Codes to turn on a feature begin with and underscore while those to disable
00025# the feature end in an underscore.
00026#
00027# Example:
00028#
00029#       $ echo "This is ${_BOLD}bold${BOLD}_."
00030#       $ echo "This is ${_RED}red${RED}_."
00031#       $ echo "This is ${_BOLD}${_RED}bold and red${RED}_>${BOLD}_."
00032
00033
00034# Reset all ANSI codes.
00035ANSI_RESET="\e[0m"
00036
00037# Font properties.
00038_BOLD="\e[1m"
00039_BOLD_="\e[21m"
00040_ITALICS="\e[3m"
00041_ITALICS_="\e[23m"
00042_UNDERLINE="\e[4m"
00043_UNDERLINE_="\e[24m"
00044_INVERSE="\e[7m"

```

```
00045 INVERSE_= "\e[27m"
00046 _STRIKE=" \e[9m"
00047 STRIKE_= "\e[29m"
00048
00049 # Foreground colors.
00050 _BLACK=" \e[30m"
00051 BLACK_= "\e[39m"
00052 _RED=" \e[31m"
00053 RED_= "\e[39m"
00054 _GREEN=" \e[32m"
00055 GREEN_= "\e[39m"
00056 _YELLOW=" \e[33m"
00057 YELLOW_= "\e[39m"
00058 _BLUE=" \e[34m"
00059 BLUE_= "\e[39m"
00060 _MAGENTA=" \e[35m"
00061 MAGENTA_= "\e[39m"
00062 _CYAN=" \e[36m"
00063 CYAN_= "\e[39m"
00064 _WHITE=" \e[37m"
00065 WHITE_= "\e[39m"
00066 FG_RESET=" \e[39m"
00067
00068 # Background colors.
00069 _BG_BLACK=" \e[40m"
00070 BG_BLACK_= "\e[49m"
00071 _BG_RED=" \e[41m"
00072 BG_RED_= "\e[49m"
00073 _BG_GREEN=" \e[42m"
00074 BG_GREEN_= "\e[49m"
00075 _BG_YELLOW=" \e[43m"
00076 BG_YELLOW_= "\e[49m"
00077 _BG_BLUE=" \e[44m"
00078 BG_BLUE_= "\e[49m"
00079 _BG_MAGENTA=" \e[45m"
00080 BG_MAGENTA_= "\e[49m"
00081 _BG_CYAN=" \e[46m"
00082 BG_CYAN_= "\e[49m"
00083 _BG_WHITE=" \e[47m"
00084 BG_WHITE_= "\e[49m"
00085 BG_RESET=" \e[49m"
```

16.13 App.cpp File Reference

```
#include <chrono>
#include <memory>
#include <utility>
#include <vector>
#include <signal.h>
#include "App.hpp"
#include "Interface.hpp"
#include "InterfaceThreader.hpp"
Include dependency graph for App.cpp:
```



16.14 App.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <chrono>
00019 #include <memory>
00020 #include <utility>
00021 #include <vector>
00022
00023 #include <signal.h>
00024
00025 #include "App.hpp"
00026 #include "Interface.hpp"
00027 #include "InterfaceThreader.hpp"
00028
00029
00030 using namespace std::chrono_literals;
00031
00032
00033 /** Construct mavtables application from a vector of interfaces.
00034 *
00035 * Neither the interfaces, nor the application will be started until the \ref
00036 * run method is called.
00037 *
00038 * \param interfaces A vector of interfaces.
00039 */
00040 App::App(std::vector<std::unique_ptr<Interface>> interfaces)
00041 {
00042     // Create threader for each interface.
00043     for (auto &interface : interfaces)
00044     {
00045         threaders_.push_back(
00046             std::make_unique<InterfaceThreader>(
00047                 std::move(interface), 100ms, InterfaceThreader::DELAY_START))
00048     }
00049 }
00050
00051
00052 /** Start the application.
00053 *
00054 * This starts listening on all interfaces.
00055 *
00056 * \throws std::system_error if an error is generated while waiting for Ctrl+C.
00057 * \throws std::runtime_error if run on Microsoft Windows.
00058 */
00059 void App::run()
00060 {
00061     // Start interfaces.
00062     for (auto &interface : threaders_)
00063     {
00064         interface->start();
00065     }
00066
00067     #ifdef UNIX
00068     // Wait for SIGINT (Ctrl+C).
00069     sigset(SIGINT, &sigHandler);
00070     sigemptyset(SIGPOLLIN, &pollSet);
00071     sigadd(SIGPOLLIN, &pollSet, &sigHandler);
00072     sigprocmask(SIG_BLOCK, &pollSet, NULL);
00073     int sig;
00074
00075     if (sigwait(&pollSet, &sig) < 0)
00076     {

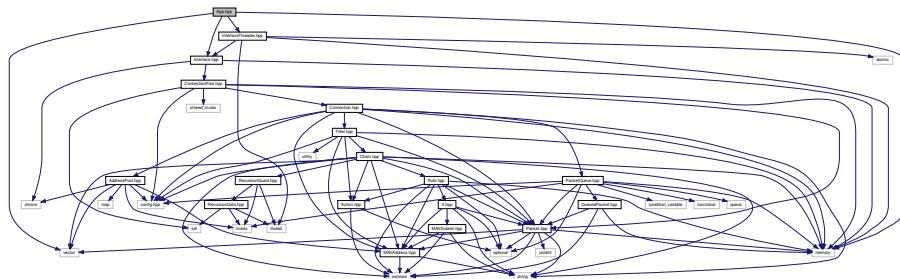
```

```

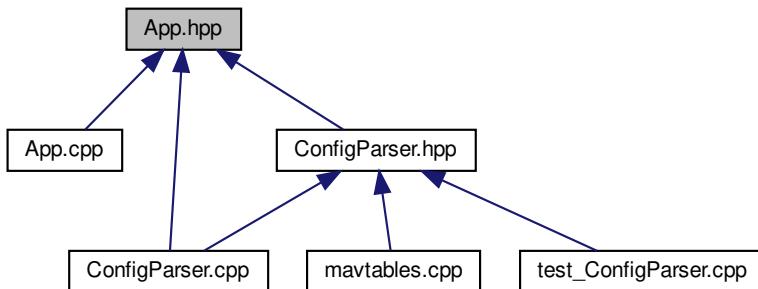
00077     throw std::system_error(std::error_code(errno, std::system_category()));
00078 }
00079
00080 #elif WINDOWS
00081 throw std::runtime_error("Microsoft Windows is not currently supported.")
00082 #endif
00083
00084 // Shutdown interfaces.
00085 for (auto &interface : threaders_)
00086 {
00087     interface->shutdown();
00088 }
00089 }
```

16.15 App.hpp File Reference

```
#include <memory>
#include <vector>
#include "Interface.hpp"
#include "InterfaceThreader.hpp"
Include dependency graph for App.hpp:
```



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



Classes

- class [App](#)

16.16 App.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef APP_HPP_
00019 #define APP_HPP_
00020
00021
00022 #include <memory>
00023 #include <vector>
00024
00025 #include "Interface.hpp"
00026 #include "InterfaceThreader.hpp"
00027
00028
00029 /** The mavtables application class.
00030 */
00031 class App
00032 {
00033     public:
00034         App(std::vector<std::unique_ptr<Interface>> interfaces);
00035         void run();
00036
00037     private:
00038         std::vector<std::unique_ptr<InterfaceThreader>> threaders_;
00039 };
00040
00041
00042 #endif // APP_HPP_

```

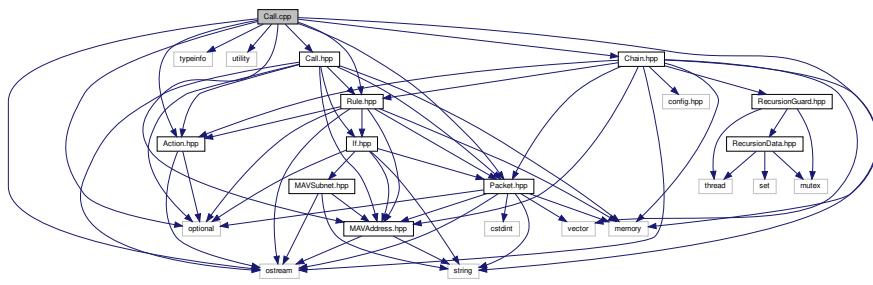
16.17 Call.cpp File Reference

```

#include <memory>
#include <optional>
#include <ostream>
#include <typeinfo>
#include <utility>
#include "Action.hpp"
#include "Call.hpp"
#include "Chain.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "Rule.hpp"

```

Include dependency graph for Call.cpp:



16.18 Call.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <memory>
00019 #include <optional>
00020 #include <ostream>
00021 #include <typeinfo>
00022 #include <utility>
00023
00024 #include "Action.hpp"
00025 #include "Call.hpp"
00026 #include "Chain.hpp"
00027 #include "MAVAddress.hpp"
00028 #include "Packet.hpp"
00029 #include "Rule.hpp"
00030
00031
00032 /** Construct a call rule given a chain to delegate to, without a priority.
00033 *
00034 * A call rule is used to delegate the decision on whether to accept or reject
00035 * a packet/address combination to another filter \ref Chain. If this called
00036 * chain does not make a decision then rule evaluation should continue in the
00037 * calling chain.
00038 *
00039 * \param chain The chain to delegate decisions of whether to accept or reject
00040 * a packet/address combination to. nullptr is not valid.
00041 * \param condition The condition used to determine the rule matches a
00042 * particular packet/address combination given to the \ref action method.
00043 * The default is {} which indicates the rule matches any packet/address
00044 * combination.
00045 * \throws std::invalid_argument if the given \p chain pointer is null.
00046 * \sa action
00047 */
00048 Call::Call(
00049     std::shared_ptr<Chain> chain, std::optional<If> condition)
00050     : Rule(std::move(condition)), chain_(std::move(chain))
00051 {
00052     if (chain_ == nullptr)
00053     {

```

```

00054         throw std::invalid_argument("Given chain pointer is null.");
00055     }
00056 }
00057
00058
00059 /** Construct a call rule given a chain to delegate to, with a priority.
00060 *
00061 * A call rule is used to delegate the decision on whether to accept or reject
00062 * a packet/address combination to another filter \ref Chain.
00063 *
00064 * \param chain The chain to delegate decisions of whether to accept or reject
00065 * a packet/address combination to. nullptr is not valid.
00066 * \param priority The priority to accept packets with. A higher number is
00067 * more important and will be routed first.
00068 * \param condition The condition used to determine the rule matches a
00069 * particular packet/address combination given to the \ref action method.
00070 * The default is {} which indicates the rule matches any packet/address
00071 * combination.
00072 * \throws std::invalid_argument if the given pointer is null.
00073 * \sa action
00074 */
00075 Call::Call(
00076     std::shared_ptr<Chain> chain, int priority,
00077     std::optional<If> condition)
00078 : Rule(std::move(condition)), chain_(std::move(chain)), priority_(priority)
00079 {
00080     if (chain_ == nullptr)
00081     {
00082         throw std::invalid_argument("Given chain pointer is null.");
00083     }
00084 }
00085
00086
00087 /** \copydoc Rule::print_(std::ostream &os) const
00088 *
00089 * Prints "call <Chain Name> <If Statement>" or "call <Chain Name> with
00090 * priority <If Statement> with priority <priority>" if the priority is
00091 * given.
00092 */
00093 std::ostream &Call::print_(std::ostream &os) const
00094 {
00095     os << "call " << chain_->name();
00096
00097     if (priority_)
00098     {
00099         os << " with priority " << priority_.value();
00100     }
00101
00102     if (condition_)
00103     {
00104         os << " " << condition_.value();
00105     }
00106
00107     return os;
00108 }
00109
00110
00111 /** \copydoc Rule::action(const Packet&,const MAVAddress&) const
00112 *
00113 * %If the condition has not been set or it matches the given packet/address
00114 * combination then the choice of \ref Action will be delegated to the
00115 * contained \ref Chain.
00116 *
00117 * %If the result from the chain is an accept \ref Action object and no
00118 * priority has been set on it but this \ref Rule has a priority then the
00119 * priority will be set.
00120 */
00121 Action Call::action(
00122     const Packet &packet, const MAVAddress &address) const
00123 {
00124     if (!condition_ || condition_->check(packet, address))
00125     {
00126         auto result = chain_->action(packet, address);
00127
00128         if (priority_)
00129         {
00130             // Only has an effect if the action is accept and does not already
00131             // have a priority.
00132             result.priority(priority_.value());
00133         }
00134

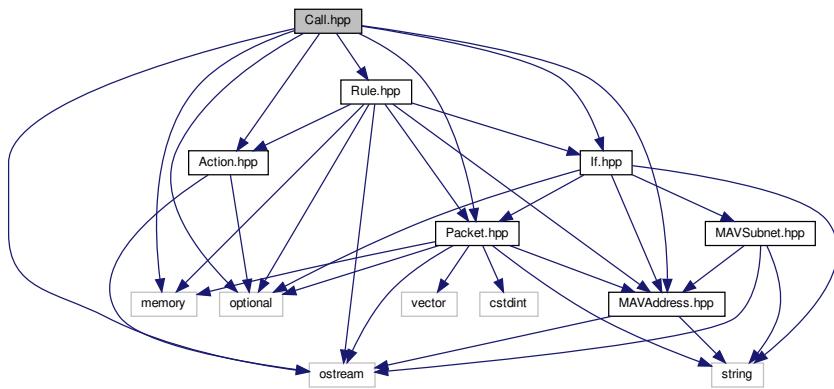
```

```
00135         return result;
00136     }
00137
00138     return Action::make_continue();
00139 }
00140
00141
00142 std::unique_ptr<Rule> Call::clone() const
00143 {
00144     if (priority_)
00145     {
00146         return std::make_unique<Call>(chain_, priority_.value(), condition_);
00147     }
00148
00149     return std::make_unique<Call>(chain_, condition_);
00150 }
00151
00152
00153 /** \copydoc Rule::operator==(const Rule&) const
00154 *
00155 * Compares the chain and priority (if set) associated with the rule as well.
00156 */
00157 bool Call::operator==(const Rule &other) const
00158 {
00159     return typeid(*this) == typeid(other) &&
00160            chain_ == static_cast<const Call &>(other).chain_ &&
00161            priority_ == static_cast<const Call &>(other).priority_ &&
00162            condition_ == static_cast<const Call &>(other).condition_;
00163 }
00164
00165
00166 /** \copydoc Rule::operator!=(const Rule&) const
00167 *
00168 * Compares the chain and priority (if set) associated with the rule as well.
00169 */
00170 bool Call::operator!=(const Rule &other) const
00171 {
00172     return typeid(*this) != typeid(other) ||
00173            chain_ != static_cast<const Call &>(other).chain_ ||
00174            priority_ != static_cast<const Call &>(other).priority_ ||
00175            condition_ != static_cast<const Call &>(other).condition_;
00176 }
```

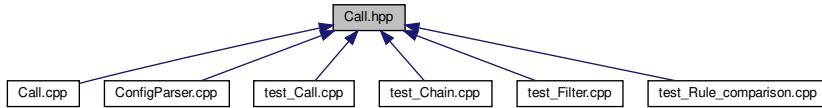
16.19 Call.hpp File Reference

```
#include <memory>
#include <optional>
#include <iostream>
#include "Action.hpp"
#include "If.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "Rule.hpp"
```

Include dependency graph for Call.hpp:



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



Classes

- class [Call](#)

16.20 Call.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef CALL_HPP_
00019 #define CALL_HPP_
00020
  
```

```

00021
00022 #include <memory>
00023 #include <optional>
00024 #include <ostream>
00025
00026 #include "Action.hpp"
00027 #include "If.hpp"
00028 #include "MAVAddress.hpp"
00029 #include "Packet.hpp"
00030 #include "Rule.hpp"
00031
00032
00033 // Forward declaration of the chain class.
00034 class Chain;
00035
00036
00037 /** Delegate decision on a packet to another \ref Chain.
00038 *
00039 * Rule to delegate the decision on what to do with a packet to a filter \ref
00040 * Chain. %If this chain cannot make a determination (continue action
00041 * returned), \ref Rule evaluation should resume after this rule.
00042 */
00043 class Call : public Rule
00044 {
00045     public:
00046         Call(std::shared_ptr<Chain> chain,
00047               std::optional<If> condition = {});
00048         Call(std::shared_ptr<Chain> chain, int priority,
00049               std::optional<If> condition = {});
00050         virtual Action action(
00051             const Packet &packet, const MAVAddress &address) const;
00052         virtual std::unique_ptr<Rule> clone() const;
00053         virtual bool operator==(const Rule &other) const;
00054         virtual bool operator!=(const Rule &other) const;
00055
00056     protected:
00057         virtual std::ostream &print_(std::ostream &os) const;
00058
00059     private:
00060         std::shared_ptr<Chain> chain_;
00061         std::optional<int> priority_;
00062     };
00063
00064
00065 #endif // CALL_HPP_

```

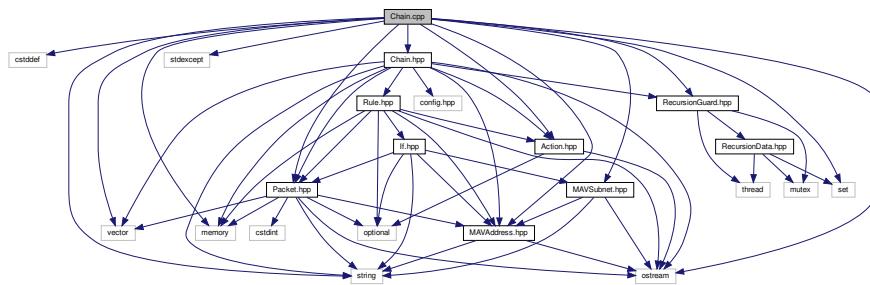
16.21 Chain.cpp File Reference

```

#include <cstddef>
#include <memory>
#include <ostream>
#include <set>
#include <stdexcept>
#include <string>
#include <vector>
#include "Action.hpp"
#include "Chain.hpp"
#include "MAVAddress.hpp"
#include "MAVSubnet.hpp"
#include "Packet.hpp"
#include "RecursionGuard.hpp"

```

Include dependency graph for Chain.cpp:



16.22 Chain.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <cstddef>
00019 #include <memory>
00020 #include <iostream>
00021 #include <set>
00022 #include <stdexcept>
00023 #include <string>
00024 #include <vector>
00025
00026 #include "Action.hpp"
00027 #include "Chain.hpp"
00028 #include "MAVAddress.hpp"
00029 #include "MAVSubnet.hpp"
00030 #include "Packet.hpp"
00031 #include "RecursionGuard.hpp"
00032
00033
00034 /** Copy constructor.
00035 *
00036 * \param other Chain to copy from.
00037 */
00038 Chain::Chain(const Chain &other)
00039 : name_(other.name_)
00040 {
00041     for (auto &rule : other.rules_)
00042     {
00043         rules_.push_back(rule->clone());
00044     }
00045 }
00046
00047
00048 /** Construct a new filter chain.
00049 *
00050 * \note No rule in the chain may contain a \ref GoTo or \ref Call that would
00051 * directly or indirectly result in returning to this chain.
00052 */
  
```

```
00053 * \param name The name of the filter chain. This is only used when printing
00054 *      the chain. The name cannot contain whitespace.
00055 * \param rules A vector of the rules used in the filter chain. This must be
00056 *      moved from since the vector is made up of std::unique_ptr's.
00057 * \throws std::invalid_argument if the \p name contains whitespace.
00058 */
00059 Chain::Chain(
00060     std::string name, std::vector<std::unique_ptr<Rule>> &&rules)
00061     : name_(std::move(name)), rules_(std::move(rules))
00062 {
00063     if (name_.find_first_of("\t\n") != std::string::npos)
00064     {
00065         throw std::invalid_argument("Chain names cannot contain whitespace.");
00066     }
00067 }
00068
00069
00070 /** Decide what to do with a \ref Packet.
00071 *
00072 * Determine what action to take with the given \p packet sent to the
00073 * given \p address. The possible actions are documented in the \ref
00074 * Action class.
00075 *
00076 * The filter chain will loop through all the rules and the first one that
00077 * matches and returns something other than the continue \ref Action will be
00078 * taken as the result.
00079 *
00080 * \note An error will be thrown if any \ref Call or \ref GoTo rule matches
00081 *      that directly or indirectly loops back to this chain.
00082 *
00083 * \param packet The packet to determine whether to allow or not.
00084 * \param address The address the \p packet will be sent out on if the
00085 *      action allows it.
00086 * \returns The action to take with the packet. If this is the accept \ref
00087 *      Action object, it may also contain a priority for the packet.
00088 * \throws RecursionError if a rule loops back to this chain.
00089 */
00090 Action Chain::action(
00091     const Packet &packet, const MAVAddress &address)
00092 {
00093     // Prevent recursion.
00094     RecursionGuard recursion_guard(recursion_data_);
00095
00096     // Loop through the rules.
00097     for (auto const &rule : rules_)
00098     {
00099         auto result = rule->action(packet, address);
00100
00101         // Return rule result if not CONTINUE.
00102         if (result.action() != Action::CONTINUE)
00103         {
00104             return result;
00105         }
00106     }
00107
00108     return Action::make_continue();
00109 }
00110
00111
00112 /** Append a new rule to the filter chain.
00113 *
00114 * \param rule A new filter rule to append to the chain.
00115 */
00116 void Chain::append(std::unique_ptr<Rule> rule)
00117 {
00118     rules_.push_back(std::move(rule));
00119 }
00120
00121
00122 /** Return the name of the chain.
00123 *
00124 * \note This is only used when printing the chain.
00125 *
00126 * \returns The chain's name.
00127 */
00128 const std::string &Chain::name() const
00129 {
00130     return name_;
00131 }
00132
00133
```

```

00134 /** Assignment operator.
00135 *
00136 * \param other Chain to copy from.
00137 */
00138 Chain &Chain::operator=(const Chain &other)
00139 {
00140     name_ = other.name_;
00141     rules_.clear();
00142
00143     for (auto &rule : other.rules_)
00144     {
00145         rules_.push_back(rule->clone());
00146     }
00147
00148     recursion_data_ = other.recursion_data_;
00149     return *this;
00150 }
00151
00152
00153 /** Equality comparison.
00154 *
00155 * Compares the chain name and each \ref Rule in the chain.
00156 *
00157 * \relates Chain
00158 * \param lhs The left hand side filter chain.
00159 * \param rhs The right hand side filter chain.
00160 * \retval true if \p lhs is the same as rhs.
00161 * \retval false if \p lhs is not the same as rhs.
00162 */
00163 bool operator==(const Chain &lhs, const Chain &rhs)
00164 {
00165     // Compare names.
00166     if (lhs.name() != rhs.name())
00167     {
00168         return false;
00169     }
00170
00171     // Compare number of rules.
00172     if (lhs.rules_.size() != rhs.rules_.size())
00173     {
00174         return false;
00175     }
00176
00177     // Compare rules one by one.
00178     for (size_t i = 0; i < lhs.rules_.size(); ++i)
00179     {
00180         if (*(lhs.rules_[i]) != *(rhs.rules_[i]))
00181         {
00182             return false;
00183         }
00184     }
00185
00186     return true;
00187 }
00188
00189
00190 /** Inequality comparison.
00191 *
00192 * Compares the chain name and each \ref Rule in the chain.
00193 *
00194 * \relates Chain
00195 * \param lhs The left hand side action.
00196 * \param rhs The right hand side action.
00197 * \retval true if \p lhs is not the same as rhs.
00198 * \retval false if \p lhs is the same as rhs.
00199 */
00200 bool operator!=(const Chain &lhs, const Chain &rhs)
00201 {
00202     return !(lhs == rhs);
00203 }
00204
00205
00206 /** Print the given filter chain to to the given output stream.
00207 *
00208 * An example is:
00209 * ``
00210 * chain default {
00211 *     reject if HEARTBEAT from 10.10;
00212 *     accept with priority -3 if GPS_STATUS to 172.0/8;
00213 *     accept if GLOBAL_POSITION_INT to 172.0/8;
00214 *     goto ap-in with priority 3 if from 192.168;

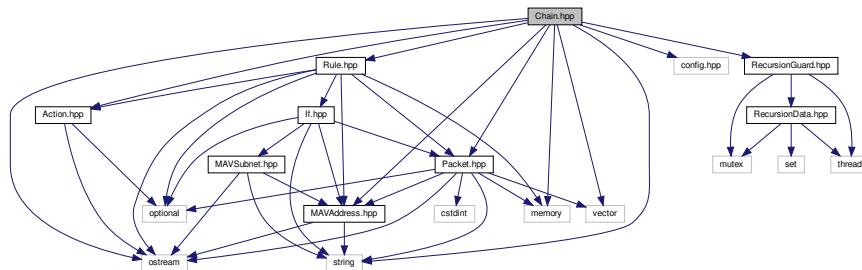
```

```

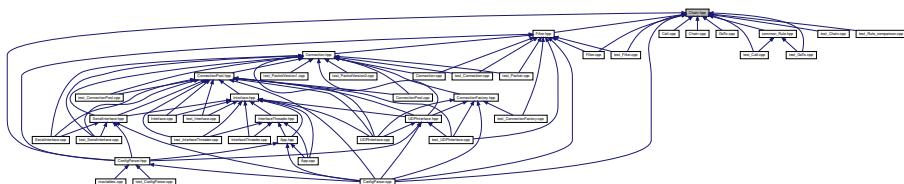
00215 *      call ap-out if to 192.168;
00216 *      reject;
00217 *
00218 */
00219 *
00220 * \relates Chain
00221 * \param os The output stream to print to.
00222 * \param chain The filter chain to print.
00223 * \returns The output stream.
00224 */
00225 std::ostream &operator<<(std::ostream &os, const Chain &chain)
00226 {
00227     os << "chain " << chain.name() << " {" << std::endl;
00228
00229     for (auto const &rule : chain.rules_)
00230     {
00231         os << "    " << *rule << ";" << std::endl;
00232     }
00233
00234     os << "}";
00235     return os;
00236 }
```

16.23 Chain.hpp File Reference

```
#include <memory>
#include <iostream>
#include <string>
#include <vector>
#include "Action.hpp"
#include "config.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "RecursionGuard.hpp"
#include "Rule.hpp"
Include dependency graph for Chain.hpp:
```



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



Classes

- class [Chain](#)

Functions

- bool [operator==](#) (const [Chain](#) &lhs, const [Chain](#) &rhs)
- bool [operator!=](#) (const [Chain](#) &lhs, const [Chain](#) &rhs)
- std::ostream & [operator<<](#) (std::ostream &os, const [Chain](#) &chain)

16.23.1 Function Documentation

16.23.1.1 operator"!="()

```
bool operator!= (
    const Chain & lhs,
    const Chain & rhs )
```

16.23.1.2 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const Chain & chain )
```

16.23.1.3 operator==()

```
bool operator== (
    const Chain & lhs,
    const Chain & rhs )
```

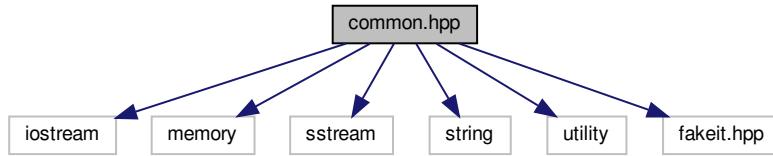
16.24 Chain.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef CHAIN_HPP_
00019 #define CHAIN_HPP_
00020
00021
00022 #include <memory>
00023 #include <iostream>
00024 #include <string>
00025 #include <vector>
00026
00027 #include "Action.hpp"
00028 #include "config.hpp"
00029 #include "MAVAddress.hpp"
00030 #include "Packet.hpp"
00031 #include "RecursionGuard.hpp"
00032 #include "Rule.hpp"
00033
00034
00035 /** A filter chain, containing a list of rules to check packets against.
00036 */
00037 class Chain
00038 {
00039     public:
00040         Chain(const Chain &other);
00041         /** Move constructor.
00042             *
00043             * \param other Chain to move from.
00044             */
00045         Chain(Chain &&other) = default;
00046         Chain(std::string name_,
00047               std::vector<std::unique_ptr<Rule>> &&rules = {});
00048         TEST_VIRTUAL ~Chain() = default;
00049         TEST_VIRTUAL Action action(
00050             const Packet &packet, const MAVAddress &address);
00051         void append(std::unique_ptr<Rule> rule);
00052         const std::string &name() const;
00053         Chain &operator=(const Chain &other);
00054         /** Assignment operator (by move semantics).
00055             *
00056             * \param other Chain to move from.
00057             */
00058         Chain &operator=(Chain &&other) = default;
00059
00060         friend bool operator==(const Chain &lhs, const Chain &rhs);
00061         friend bool operator!=(const Chain &lhs, const Chain &rhs);
00062         friend std::ostream &operator<<(std::ostream &os, const Chain &chain);
00063
00064     private:
00065         std::string name_;
00066         std::vector<std::unique_ptr<Rule>> rules_;
00067         RecursionData recursion_data_;
00068     };
00069
00070
00071 bool operator==(const Chain &lhs, const Chain &rhs);
00072 bool operator!=(const Chain &lhs, const Chain &rhs);
00073 std::ostream &operator<<(std::ostream &os, const Chain &chain);
00074
00075
00076 #endif // CHAIN_HPP_
```

16.25 common.hpp File Reference

```
#include <iostream>
#include <memory>
#include <sstream>
#include <string>
#include <utility>
#include "fakeit.hpp"
```

Include dependency graph for common.hpp:



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



Classes

- class [MockCOut](#)
- class [MockCErr](#)

Functions

- template<typename T >
std::shared_ptr<T> [mock_shared](#) (fakeit::Mock< T > &mock)
- template<typename T >
std::unique_ptr<T> [mock_unique](#) (fakeit::Mock< T > &mock)

16.25.1 Function Documentation

16.25.1.1 [mock_shared\(\)](#)

```
template<typename T >
std::shared_ptr<T> mock_shared (
    fakeit::Mock< T > & mock )
```

Construct a std::shared_ptr from a fakit::Mock object.

Template Parameters

<i>T</i>	The type of object being mocked.
----------	----------------------------------

Parameters

<i>mock</i>	The mock object itself.
-------------	-------------------------

Definition at line 33 of file [common.hpp](#).

Here is the caller graph for this function:



16.25.1.2 mock_unique()

```
template<typename T >
std::unique_ptr<T> mock_unique (
    fakeit::Mock< T > & mock )
```

Construct a std::unique_ptr from a fakit::Mock object.

Template Parameters

<i>T</i>	The type of object being mocked.
----------	----------------------------------

Parameters

<i>mock</i>	The mock object itself.
-------------	-------------------------

Definition at line 46 of file [common.hpp](#).

Here is the caller graph for this function:



16.26 common.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <iostream>
00019 #include <memory>
00020 #include <sstream>
00021 #include <string>
00022 #include <utility>
00023
00024 #include "fakeit.hpp"
00025
00026
00027 /** Construct a std::shared_ptr from a fakit::Mock object.
00028 *
00029 * \tparam T The type of object being mocked.
00030 * \param mock The mock object itself.
00031 */
00032 template<typename T>
00033 std::shared_ptr<T> mock_shared(fakeit::Mock<T> &mock) {
00034     fakeit::Fake(Dtor(mock));
00035     std::shared_ptr<T> ptr(&mock.get());
00036     return std::move(ptr);
00037 }
00038
00039
00040 /** Construct a std::unique_ptr from a fakit::Mock object.
00041 *
00042 * \tparam T The type of object being mocked.
00043 * \param mock The mock object itself.
00044 */
00045 template<typename T>
00046 std::unique_ptr<T> mock_unique(fakeit::Mock<T> &mock) {
00047     fakeit::Fake(Dtor(mock));
00048     std::unique_ptr<T> ptr(&mock.get());
00049     return std::move(ptr);
00050 }
00051
00052
00053 /** RAII class to replace std::cout with a mocked buffer.
00054 */
00055 class MockCOut
00056 {
  
```

```

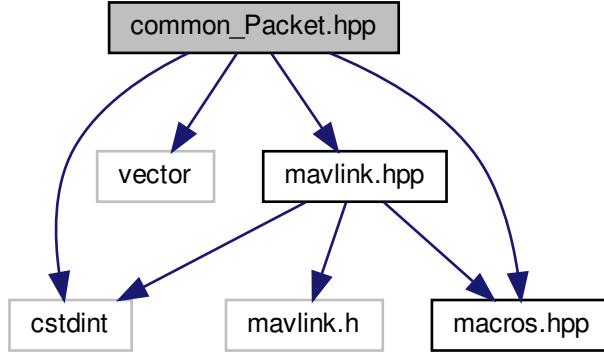
00057     public:
00058         /** Replace std::cout with this mock.
00059          */
00060         MockCOut()
00061             : sbuf_(std::cout.rdbuf())
00062         {
00063             std::cout.rdbuf(buffer_.rdbuf());
00064         }
00065         /** Restore std::cout.
00066          */
00067         ~MockCOut()
00068         {
00069             std::cout.rdbuf(sbuf_);
00070         }
00071         /** Return the contents of the mocked std::cout buffer as a string.
00072          *
00073          * \returns The contents of the mocked buffer.
00074          */
00075         std::string buffer()
00076         {
00077             return buffer_.str();
00078         }
00079         /** Erase the mocked buffer.
00080          */
00081         void reset()
00082         {
00083             buffer_.str("");
00084         }
00085
00086     private:
00087         std::stringstream buffer_;
00088         std::streambuf *sbuf_;
00089     };
00090
00091
00092     /** RAII class to replace std::cerr with a mocked buffer.
00093      */
00094     class MockCErr
00095     {
00096         public:
00097             /** Replace std::cerr with this mock.
00098              */
00099             MockCErr()
00100                 : sbuf_(std::cerr.rdbuf())
00101             {
00102                 std::cerr.rdbuf(buffer_.rdbuf());
00103             }
00104             /** Restore std::cerr.
00105              */
00106             ~MockCErr()
00107             {
00108                 std::cerr.rdbuf(sbuf_);
00109             }
00110             /** Return the contents of the mocked std::cerr buffer as a string.
00111              *
00112              * \returns The contents of the mocked buffer.
00113              */
00114             std::string buffer()
00115             {
00116                 return buffer_.str();
00117             }
00118             /** Erase the mocked buffer.
00119              */
00120             void reset()
00121             {
00122                 buffer_.str("");
00123             }
00124
00125     private:
00126         std::stringstream buffer_;
00127         std::streambuf *sbuf_;
00128     };

```

16.27 common_Packet.hpp File Reference

```
#include <cstdint>
```

```
#include <vector>
#include "macros.hpp"
#include "mavlink.hpp"
Include dependency graph for common_Packet.hpp:
```



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



16.28 common_Packet.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <cstdint>
00019 #include <vector>
00020
00021 #include "macros.hpp"
00022 #include "mavlink.hpp"
00023
00024
00025 namespace
00026 {
00027 
```

```

00028 // HEARTBEAT v1.0 structure for testing packets without target
00029 // system/component.
00030 struct PACKED HeartbeatV1
00031 {
00032     struct PACKED payload
00033     {
00034         uint8_t type = 1;
00035         uint8_t autopilot = 2;
00036         uint8_t base_mode = 3;
00037         uint32_t custom_mode = 4;
00038         uint8_t system_status = 5;
00039         uint8_t mavlink_version = 6;
00040     };
00041     uint8_t magic = 0xFFE;
00042     uint8_t len = sizeof(payload);
00043     uint8_t seq = 0xFFE; // test internal magic byte
00044     uint8_t sysid = 127;
00045     uint8_t compid = 1;
00046     uint8_t msgid = 0;
00047     payload payload;
00048     uint16_t checksum = 0xFACE;
00049 };
00050
00051
00052 // PING v1.0 structure for testing target system/compoent.
00053 struct PACKED PingV1
00054 {
00055     struct PACKED payload
00056     {
00057         uint64_t time_usec = 2951280000000000;
00058         uint32_t seq = 0xBA5EBA11;
00059         uint8_t target_system = 127;
00060         uint8_t target_component = 1;
00061     };
00062     uint8_t magic = 0xFFE;
00063     uint8_t len = sizeof(payload);
00064     uint8_t seq = 0xFFE; // test internal magic byte
00065     uint8_t sysid = 192;
00066     uint8_t compid = 168;
00067     uint8_t msgid = 4;
00068     payload payload;
00069     uint16_t checksum = 0xFACE;
00070 };
00071
00072
00073 // SET_MODE v1.0 structure for testing target system only.
00074 struct PACKED SetModeV1
00075 {
00076     struct PACKED payload
00077     {
00078         uint32_t custom_mode = 2;
00079         uint8_t target_system = 123;
00080         uint8_t base_mode = 0xFFE; // test magic byte in payload
00081     };
00082     uint8_t magic = 0xFFE;
00083     uint8_t len = sizeof(payload);
00084     uint8_t seq = 0xFFE; // test internal magic byte
00085     uint8_t sysid = 172;
00086     uint8_t compid = 0;
00087     uint8_t msgid = 11;
00088     payload payload;
00089     uint16_t checksum = 0xFACE;
00090 };
00091
00092
00093 // ENCAPSULATED_DATA v1.0 structure for testing maximum length packets.
00094 struct PACKED EncapsulatedDataV1
00095 {
00096     struct PACKED payload
00097     {
00098         uint16_t seqnr = 0;
00099         uint8_t data[253] = {
00100             0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
00101             10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
00102             20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
00103             30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
00104             40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
00105             50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
00106             60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
00107             70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
00108             80, 81, 82, 83, 84, 85, 86, 87, 88, 89,

```

```

00109         90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
00110         100, 101, 102, 103, 104, 105, 106, 107, 108, 109,
00111         110, 111, 112, 113, 114, 115, 116, 117, 118, 119,
00112         120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
00113         130, 131, 132, 133, 134, 135, 136, 137, 138, 139,
00114         140, 141, 142, 143, 144, 145, 146, 147, 148, 149,
00115         150, 151, 152, 153, 154, 155, 156, 157, 158, 159,
00116         160, 161, 162, 163, 164, 165, 166, 167, 168, 169,
00117         170, 171, 172, 173, 174, 175, 176, 177, 178, 179,
00118         180, 181, 182, 183, 184, 185, 186, 187, 188, 189,
00119         190, 191, 192, 193, 194, 195, 196, 197, 198, 199,
00120         200, 201, 202, 203, 204, 205, 206, 207, 208, 209,
00121         210, 211, 212, 213, 214, 215, 216, 217, 218, 219,
00122         220, 221, 222, 223, 224, 225, 226, 227, 228, 229,
00123         230, 231, 232, 233, 234, 235, 236, 237, 238, 239,
00124         240, 241, 242, 243, 244, 245, 246, 247, 248, 249,
00125         250, 251, 252
00126     );
00127 };
00128     uint8_t magic = 0xFE;
00129     uint8_t len = 255;
00130     uint8_t seq = 0xFE; // test internal magic byte
00131     uint8_t sysid = 224;
00132     uint8_t compid = 255;
00133     uint8_t msgid = 131;
00134     payload payload;
00135     uint16_t checksum = 0xFACE;
00136 };
00137
00138
00139 #ifdef __clang__
00140     #pragma clang diagnostic push
00141     #pragma clang diagnostic ignored "-Wunused-member-function"
00142 #endif
00143
00144 // HEARTBEAT v2.0 structure for testing packets without target
00145 // system/component.
00146 struct PACKED HeartbeatV2
00147 {
00148     struct PACKED payload
00149     {
00150         uint8_t type = 1;
00151         uint8_t autopilot = 2;
00152         uint8_t base_mode = 3;
00153         uint32_t custom_mode = 4;
00154         uint8_t system_status = 5;
00155         uint8_t mavlink_version = 6;
00156     };
00157     uint8_t magic = 0xFD;
00158     uint8_t len = sizeof(payload);
00159     uint8_t incompat_flags = 0;
00160     uint8_t compat_flags = 0;
00161     uint8_t seq = 0xFD; // test internal magic byte
00162     uint8_t sysid = 127;
00163     uint8_t compid = 1;
00164     uint32_t msgid : 24;
00165     payload payload;
00166     uint16_t checksum = 0xFACE;
00167     HeartbeatV2() : msgid(0) {}
00168 };
00169
00170
00171 // PING v2.0 structure for testing target system/component.
00172 struct PACKED PingV2
00173 {
00174     struct PACKED payload
00175     {
00176         uint64_t time_usec = 295128000000000;
00177         uint32_t seq = 0xBA5EBA11;
00178         uint8_t target_system = 127;
00179         uint8_t target_component = 1;
00180     };
00181     uint8_t magic = 0xFD;
00182     uint8_t len = sizeof(payload);
00183     uint8_t incompat_flags = 0;
00184     uint8_t compat_flags = 0;
00185     uint8_t seq = 0xFD; // test internal magic byte
00186     uint8_t sysid = 192;
00187     uint8_t compid = 168;
00188     uint32_t msgid : 24;
00189     payload payload;

```

```

00190     uint16_t checksum = 0xFACE;
00191     PingV2() : msgid(4) {}
00192 };
00193
00194
00195 // SET_MODE v2.0 structure for testing target system only.
00196 struct PACKED SetModeV2
00197 {
00198     struct PACKED payload
00199     {
00200         uint32_t custom_mode = 2;
00201         uint8_t target_system = 123;
00202         uint8_t base_mode = 0xFD; // test magic byte in payload
00203     };
00204     uint8_t magic = 0xFD;
00205     uint8_t len = sizeof(payload);
00206     uint8_t incompat_flags = 0;
00207     uint8_t compat_flags = 0;
00208     uint8_t seq = 0xFD; // test internal magic byte
00209     uint8_t sysid = 172;
00210     uint8_t compid = 0;
00211     uint32_t msgid : 24;
00212     payload payload;
00213     uint16_t checksum = 0xFACE;
00214     SetModeV2() : msgid(11) {}
00215 };
00216
00217
00218 // MISSION_SET_CURRENT v2.0 for testing trimmed out target system and
00219 // components.
00220 struct PACKED MissionSetCurrentV2
00221 {
00222     struct PACKED payload
00223     {
00224         uint16_t seq = 0xFD; // test magic byte in payload
00225         // uint8_t target_system = 0;
00226         // uint8_t target_component = 0;
00227     };
00228     uint8_t magic = 0xFD;
00229     uint8_t len = sizeof(payload);
00230     uint8_t incompat_flags = 0;
00231     uint8_t compat_flags = 0;
00232     uint8_t seq = 0xFD; // test internal magic byte
00233     uint8_t sysid = 255;
00234     uint8_t compid = 0;
00235     uint32_t msgid : 24;
00236     payload payload;
00237     uint16_t checksum = 0xFACE;
00238     MissionSetCurrentV2() : msgid(41) {}
00239 };
00240
00241
00242 // ENCAPSULATED_DATA v2.0 structure for testing maximum length packets.
00243 struct PACKED EncapsulatedDataV2
00244 {
00245     struct PACKED payload
00246     {
00247         uint16_t seqnr = 0;
00248         uint8_t data[253] = {
00249             0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
00250             10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
00251             20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
00252             30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
00253             40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
00254             50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
00255             60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
00256             70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
00257             80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
00258             90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
00259             100, 101, 102, 103, 104, 105, 106, 107, 108, 109,
00260             110, 111, 112, 113, 114, 115, 116, 117, 118, 119,
00261             120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
00262             130, 131, 132, 133, 134, 135, 136, 137, 138, 139,
00263             140, 141, 142, 143, 144, 145, 146, 147, 148, 149,
00264             150, 151, 152, 153, 154, 155, 156, 157, 158, 159,
00265             160, 161, 162, 163, 164, 165, 166, 167, 168, 169,
00266             170, 171, 172, 173, 174, 175, 176, 177, 178, 179,
00267             180, 181, 182, 183, 184, 185, 186, 187, 188, 189,
00268             190, 191, 192, 193, 194, 195, 196, 197, 198, 199,
00269             200, 201, 202, 203, 204, 205, 206, 207, 208, 209,
00270             210, 211, 212, 213, 214, 215, 216, 217, 218, 219,

```

```

00271             220, 221, 222, 223, 224, 225, 226, 227, 228, 229,
00272             230, 231, 232, 233, 234, 235, 236, 237, 238, 239,
00273             240, 241, 242, 243, 244, 245, 246, 247, 248, 249,
00274             250, 251, 252
00275         );
00276     };
00277     uint8_t magic = 0xFD;
00278     uint8_t len = sizeof(payload);
00279     uint8_t incompat_flags = 0;
00280     uint8_t compat_flags = 0;
00281     uint8_t seq = 0xFD; // test internal magic byte
00282     uint8_t sysid = 224;
00283     uint8_t compid = 255;
00284     uint32_t msgid : 24;
00285     payload payload;
00286     uint16_t checksum = 0xFACE;
00287     EncapsulatedDataV2() : msgid(131) {}
00288 };
00289
00290
00291 // PARAM_EXT_REQUEST_LIST v2.0 structure for testing message ID's beyond
00292 // 255.
00293 struct PACKED ParamExtRequestListV2
00294 {
00295     struct PACKED payload
00296     {
00297         uint8_t target_system = 32;
00298         uint8_t target_component = 64;
00299     };
00300     uint8_t magic = 0xFD;
00301     uint8_t len = sizeof(payload);
00302     uint8_t incompat_flags = 0;
00303     uint8_t compat_flags = 0;
00304     uint8_t seq = 0xFD; // test internal magic byte
00305     uint8_t sysid = 0;
00306     uint8_t compid = 255;
00307     uint32_t msgid : 24;
00308     payload payload;
00309     uint16_t checksum = 0xFACE;
00310     ParamExtRequestListV2() : msgid(321) {}
00311 };
00312
00313 #ifdef __clang__
00314     #pragma clang diagnostic pop
00315 #endif
00316
00317
00318 #ifdef __clang__
00319     #pragma clang diagnostic push
00320     #pragma clang diagnostic ignored "-Wunused-template"
00321 #endif
00322
00323 // Convert a MAVLink packet structure to a vector of bytes.
00324 template <class T>
00325 static std::vector<uint8_t> to_vector(T packet)
00326 {
00327     std::vector<uint8_t> data;
00328     data.assign(reinterpret_cast<uint8_t *>(&packet),
00329                 reinterpret_cast<uint8_t *>(&packet) + sizeof(packet));
00330     return data;
00331 }
00332
00333
00334 // Convert a MAVLink packet structure to a vector of bytes with signature.
00335 template <class T>
00336 static std::vector<uint8_t> to_vector_with_sig(T packet)
00337 {
00338     std::vector<uint8_t> data;
00339     packet.incompat_flags |= MAVLINK_IFLAG_SIGNED;
00340
00341     data.assign(reinterpret_cast<uint8_t *>(&packet),
00342                 reinterpret_cast<uint8_t *>(&packet) + sizeof(packet));
00343     std::vector<uint8_t> sig =
00344     {
00345         1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
00346     };
00347     data.insert(std::end(data), std::begin(sig), std::end(sig));
00348     return data;
00349 }
00350
00351 #ifdef __clang__

```

```

00352     #pragma clang diagnostic pop
00353 #endif
00354
00355 }

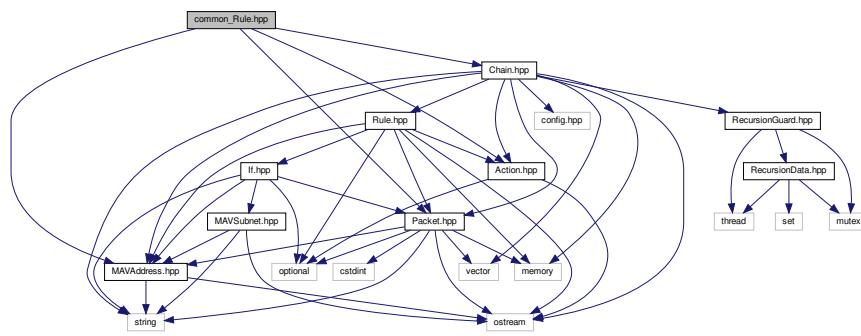
```

16.29 common_Rule.hpp File Reference

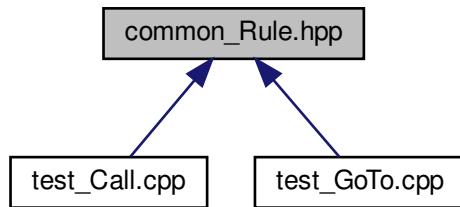
```

#include "Action.hpp"
#include "Chain.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
Include dependency graph for common_Rule.hpp:

```



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



16.30 common_Rule.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify

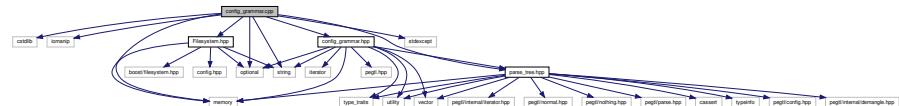
```

```
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include "Action.hpp"
00019 #include "Chain.hpp"
00020 #include "MAVAddress.hpp"
00021 #include "Packet.hpp"
00022
00023
00024 namespace
00025 {
00026
00027 #ifdef __clang__
00028     #pragma clang diagnostic push
00029     #pragma clang diagnostic ignored "-Wweak-vtables"
00030 #endif
00031     class TestChain : public Chain
00032     {
00033         public:
00034             TestChain()
00035                 : Chain("test_chain")
00036             {
00037             }
00038             // LCOV_EXCL_START
00039             ~TestChain() = default;
00040             // LCOV_EXCL_STOP
00041             virtual Action action(
00042                 const Packet &packet, const MAVAddress &address)
00043             {
00044                 (void)packet;
00045                 (void)address;
00046                 return Action::make_accept();
00047             }
00048     };
00049 #ifdef __clang__
00050     #pragma clang diagnostic pop
00051 #endif
00052
00053 }
```

16.31 config_grammar.cpp File Reference

```
#include <cstdlib>
#include <iomanip>
#include <memory>
#include <optional>
#include <stdexcept>
#include <string>
#include "config_grammar.hpp"
#include "Filesystem.hpp"
#include "parse_tree.hpp"
Include dependency graph for config_grammar.cpp:
```

Include dependency graph for config_grammar.cpp:



Namespaces

- config

Functions

- std::ostream & config::print_node (std::ostream &os, const config::parse_tree::node &node, bool print_location, const std::string &prefix)
- std::ostream & operator<< (std::ostream &os, const config::parse_tree::node &node)

16.32 config_grammar.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <cstdlib>
00019 #include <iomanip>
00020 #include <memory>
00021 #include <optional>
00022 #include <stdexcept>
00023 #include <string>
00024
00025 #include "config_grammar.hpp"
00026 #include "Filesystem.hpp"
00027 #include "parse_tree.hpp"
00028
00029
00030 /** \defgroup config Configuration functions.
00031 *
00032 * These functions are used to parse the configuration file.
00033 */
00034
00035
00036 // custom error messages
00037 namespace config
00038 {
00039
00040 /// @cond INTERNAL
00041
00042 #ifdef __clang__
00043     #pragma clang diagnostic push
00044     #pragma clang diagnostic ignored "-Wglobal-constructors"
00045     #pragma clang diagnostic ignored "-Wexit-time-destructors"
00046 #endif
00047
00048     template<>
00049     const std::string error<eos>::error_message =
00050         "expected end of statement ';' character";
00051
00052     template<>
00053     const std::string error<opening_brace>::error_message =
00054         "expected opening brace '{'";
00055
00056     template<>
00057     const std::string error<closing_brace>::error_message =
```

```
00058     "expected closing brace '}'";
00059
00060     template<>
00061     const std::string error<unsupported_statement>::error_message =
00062         "unsupported statement";
00063
00064     template<>
00065     const std::string error<default_action_option>::error_message =
00066         "expected 'accept' or 'reject'";
00067
00068     template<>
00069     const std::string error<port>::error_message =
00070         "expected a valid port number";
00071
00072     template<>
00073     const std::string error<address>::error_message =
00074         "expected a valid IP address";
00075
00076     template<>
00077     const std::string error<max_bitrate>::error_message =
00078         "expected a valid bitrate";
00079
00080     template<>
00081     const std::string error<device>::error_message =
00082         "expected a valid serial port device name";
00083
00084     template<>
00085     const std::string error<baudrate>::error_message =
00086         "expected a valid baud rate";
00087
00088     template<>
00089     const std::string error<flow_control>::error_message =
00090         "expected 'yes' or 'no'";
00091
00092     template<>
00093     const std::string error<preload>::error_message =
00094         "expected a valid MAVLink address";
00095
00096     template<>
00097     const std::string error<chain_name>::error_message =
00098         "expected a valid chain name";
00099
00100    template<>
00101    const std::string error<chain>::error_message =
00102        "expected a valid chain name";
00103
00104    template<>
00105    const std::string error<call>::error_message =
00106        "expected a valid chain name";
00107
00108    template<>
00109    const std::string error<goto_>::error_message =
00110        "expected a valid chain name";
00111
00112    template<>
00113    const std::string error<invalid_rule>::error_message =
00114        "expected a valid rule";
00115
00116    template<>
00117    const std::string error<condition_value>::error_message =
00118        "condition is empty or invalid";
00119
00120    template<>
00121    const std::string error<dest>::error_message =
00122        "expected a valid MAVLink subnet";
00123
00124    template<>
00125    const std::string error<source>::error_message =
00126        "expected a valid MAVLink subnet";
00127
00128    template<>
00129    const std::string error<mavaddr>::error_message =
00130        "expected a valid MAVLink address";
00131
00132    template<>
00133    const std::string error<integer>::error_message =
00134        "expected an integer";
00135
00136    template<>
00137    const std::string error<priority>::error_message =
00138        "expected priority level";
```

```
00139
00140     template<>
00141     const std::string error<priority_keyword>::error_message =
00142         "'with' keyword must be followed by the 'priority' keyword";
00143
00144     template<>
00145     const std::string error<elements>::error_message =
00146         "expected at least one valid statement or block";
00147
00148 #ifdef __clang__
00149     #pragma clang diagnostic pop
00150 #endif
00151
00152 /// @endcond
00153
00154
00155     /** Print an AST node and all its children.
00156     *
00157     * \ingroup config
00158     * \param os The output stream to print to.
00159     * \param node The node to print, also prints it's children.
00160     * \param print_location Set to true to print file and line numbers of each
00161     *     AST node.
00162     * \param prefix A string to prefix to each AST element. This is reserved
00163     *     for internal use.
00164     * \returns The output stream.
00165     */
00166     std::ostream &print_node(
00167         std::ostream &os,
00168         const config::parse_tree::node &node,
00169         bool print_location, const std::string &prefix)
00170     {
00171         auto new_prefix = prefix;
00172
00173         // If not the root node.
00174         if (!node.is_root())
00175         {
00176             // Add 2 spaces to the indent.
00177             new_prefix = prefix + "  ";
00178             // Remove "config::" prefix from node name.
00179             auto node_name = node.name();
00180             node_name.erase(0, 8);
00181             const auto begin = node_name.find_first_not_of("_");
00182             const auto end = node_name.find_last_not_of("_");
00183             node_name = node_name.substr(begin, end - begin + 1);
00184
00185             // Print location.
00186             if (print_location)
00187             {
00188                 os << ":" << std::setfill('0') << std::setw(3)
00189                 << node.begin().line << ":"  ";
00190             }
00191
00192             // Print name.
00193             os << prefix << node_name;
00194
00195             // Print node content.
00196             if (node.has_content())
00197             {
00198                 os << " " << node.content() << "";
00199             }
00200
00201             os << std::endl;
00202         }
00203
00204         // Print all child nodes
00205         if (!node.children.empty())
00206         {
00207             for (auto &up : node.children)
00208             {
00209                 print_node(os, *up, print_location, new_prefix);
00210             }
00211         }
00212
00213         return os;
00214     }
00215
00216 }
00217
00218
00219 /** Print AST node to the given output stream.
```

```

00220  *
00221  * This is the same as calling \ref config::print_node with the location option
00222  * set to true.
00223  *
00224  * \ingroup config
00225  * \param os The output stream to print to.
00226  * \param node The node to print, also prints it's children.
00227  * \returns The output stream.
00228 */
00229 std::ostream &operator<<(
00230     std::ostream &os, const config::parse_tree::node &node)
00231 {
00232     return config::print_node(os, node, true);
00233 }

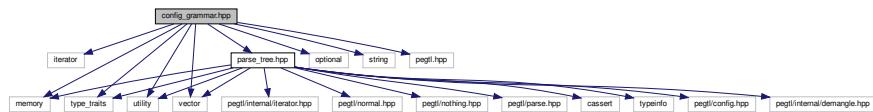
```

16.33 config_grammar.hpp File Reference

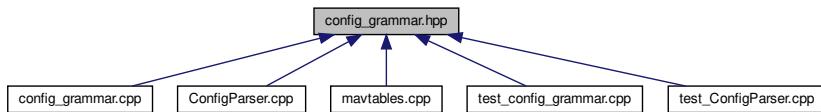
```

#include <iterator>
#include <memory>
#include <optional>
#include <string>
#include <type_traits>
#include <utility>
#include <vector>
#include <pegtl.hpp>
#include "parse_tree.hpp"
Include dependency graph for config_grammar.hpp:

```



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



Namespaces

- config

Functions

- template<typename Input>
std::unique_ptr<config::parse_tree::node> config::parse (Input &in)
- std::ostream & config::print_node (std::ostream &os, const config::parse_tree::node &node, bool print_location, const std::string &prefix)
- std::ostream & operator<< (std::ostream &os, const config::parse_tree::node &node)

16.34 config_grammar.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef CONFIG_GRAMMAR_HPP_
00019 #define CONFIG_GRAMMAR_HPP_
00020
00021
00022 #include <iterator>
00023 #include <memory>
00024 #include <optional>
00025 #include <string>
00026 #include <type_traits>
00027 #include <utility>
00028 #include <vector>
00029
00030 #include <pegtl.hpp>
00031
00032 #include "parse_tree.hpp"
00033
00034
00035 namespace config
00036 {
00037
00038     /// @cond INTERNAL
00039
00040     using namespace tao::pegtl;
00041     using namespace tao::pegtl::ascii;
00042
00043
00044     // Error message mixin.
00045     template<typename>
00046     struct error
00047     {
00048         static const std::string error_message;
00049     };
00050
00051
00052     // Store node mixin.
00053     template<typename T>
00054     struct yes : std::true_type, error<T> {};
00055
00056     // Do not store node mixin.
00057     template<typename T>
00058     struct no : std::false_type, error<T> {};
00059
00060     // Remove content mixin for store object's.
00061     template<typename T>
00062     struct yes_without_content : std::true_type, error<T>

```

```

00063     {
00064         static void transform(std::unique_ptr<config::parse_tree::node> &node)
00065     {
00066         node->remove_content();
00067     }
00068 };
00069
00070 // Replace with first child storage mixin. Removes any children of the
00071 // first child.
00072 template <typename T>
00073 struct replace_with_first_child : std::true_type, error<T>
00074 {
00075     static void transform(std::unique_ptr<config::parse_tree::node> &node)
00076     {
00077         auto children = std::move(node->children);
00078         node = std::move(children.front());
00079         children.erase(children.begin());
00080         node->children = std::move(children);
00081     }
00082 };
00083
00084 // Do not store AST node content by default
00085 template<typename T> struct store : no<T> {};
00086
00087
00088 // Comments and ignored components.
00089 struct comment : seq<one<'#'>, star<not_at<eol>, not_at<eof>, any>> {};
00090 struct ignored : sor<space, comment> {};
00091
00092 // Pad a rule.
00093 template <typename N>
00094 struct p : pad<N, ignored> {};
00095
00096 // End of statement terminator.
00097 struct eos : one< ';' > {};
00098
00099 // Generic statement with 0 or 1 values.
00100 template <typename K>
00101 struct a0_statement : seq<K, p<must<eos>>> {};
00102 template <typename K, typename V = success>
00103 struct a1_statement : seq<K, p<must<V>>, p<must<eos>>> {};
00104
00105 // Generic block.
00106 struct opening_brace : one<'{ '> {};
00107 struct closing_brace : one<'} '> {};
00108 template <typename K, typename... Statements>
00109 struct t_block : seq<K,
00110     p<must<opening_brace>>,
00111     star<p<sor<Statements...>>>,
00112     p<must<closing_brace>>> {};
00113
00114 // Generic named block.
00115 template <typename K, typename N, typename... Statements>
00116 struct t_named_block
00117     : seq<K, p<must<N>>, p<must<opening_brace>>,
00118     star<p<sor<Statements...>>>, p<must<closing_brace>>> {};
00119
00120 // Yes/No (boolean) value.
00121 struct yesno
00122     : sor<TAO_PEGTL_STRING("yes"), TAO_PEGTL_STRING("no")> {};
00123
00124 // Unsigned integer value.
00125 struct integer : plus<digit> {};
00126
00127 // Signed integer value.
00128 struct signed_integer : seq<opt<one<'+', '-'>>, integer> {};
00129
00130 // MAVLink addresses and masks.
00131 struct mavaddr : seq<integer, one<'.'>, integer> {};
00132 struct full_mask : if_must<one<':'>, mavaddr> {};
00133 struct forward_mask : if_must<one<'/'>, integer> {};
00134 struct backward_mask : if_must<one<'\\'>, integer> {};
00135 struct mavmask
00136     : seq<mavaddr, opt<sor<full_mask, forward_mask, backward_mask>>> {};
00137
00138 // Port number.
00139 struct port : integer {};
00140 template<> struct store<port> : yes<port> {};
00141
00142 // IP address.
00143 struct address : seq<integer, rep<3, seq<one<'.'>, integer>>> {};

```

```

00144     template<> struct store<address> : yes<address> {};
00145
00146     // Maximum bitrate number.
00147     struct max_bitrate : integer {};
00148     template<> struct store<max_bitrate> : yes<max_bitrate> {};
00149
00150     // Serial port device name.
00151     struct device : plus<sor<alnum, one'.', '_', '/>>> {};
00152     template<> struct store<device> : yes<device> {};
00153
00154     // Serial port baud rate.
00155     struct baudrate : integer {};
00156     template<> struct store<baudrate> : yes<baudrate> {};
00157
00158     // Serial port flow control.
00159     struct flow_control : yesno {};
00160     template<> struct store<flow_control> : yes<flow_control> {};
00161
00162     // Serial port address preload.
00163     struct preload : mavaddr {};
00164     template<> struct store<preload> : yes<preload> {};
00165
00166     // Chain name.
00167     struct chain_name : identifier {};
00168     template<> struct store<chain_name> : yes<chain_name> {};
00169
00170     // Rule actions.
00171     // accept
00172     struct accept : TAO_PEGTL_STRING("accept") {};
00173     template<> struct store<accept> : yes_without_content<accept> {};
00174     // reject
00175     struct reject : TAO_PEGTL_STRING("reject") {};
00176     template<> struct store<reject> : yes_without_content<reject> {};
00177     // call
00178     struct call : chain_name {};
00179     template<> struct store<call> : yes<call> {};
00180     struct call_container : seq<TAO_PEGTL_STRING("call"), p<must<call>>> {};
00181     template<> struct store<call_container>
00182         : replace_with_first_child<call_container> {};
00183     // goto
00184     struct goto_ : chain_name {};
00185     template<> struct store<goto_> : yes<goto_> {};
00186     struct goto_container : seq<TAO_PEGTL_STRING("goto"), p<must<goto_>>> {};
00187     template<> struct store<goto_container>
00188         : replace_with_first_child<goto_container> {};
00189     // generic action
00190     struct action : sor<accept, reject, call_container, goto_container> {};
00191
00192     // Conditional.
00193     struct packet_type : plus<sor<upper, digit, one'_>>> {};
00194     template<> struct store<packet_type> : yes<packet_type> {};
00195     struct source : mavmask {};
00196     template<> struct store<source> : yes<source> {};
00197     struct source_command
00198         : seq<TAO_PEGTL_STRING("from"), p<must<source>>> {};
00199     struct dest : mavmask {};
00200     template<> struct store<dest> : yes<dest> {};
00201     struct dest_command
00202         : seq<TAO_PEGTL_STRING("to"), p<must<dest>>> {};
00203     struct start_with_packet_type
00204         : seq<packet_type, opt<p<source_command>>, opt<p<dest_command>>> {};
00205     struct start_with_source : seq<source_command, opt<p<dest_command>>> {};
00206     struct start_with_dest : dest_command {};
00207     struct condition_value
00208         : p<sor<start_with_packet_type, start_with_source, start_with_dest>> {};
00209     struct condition : if_must<TAO_PEGTL_STRING("if"), condition_value> {};
00210     template<> struct store<condition> : yes_without_content<condition> {};
00211
00212     // Priority.
00213     struct priority : signed_integer {};
00214     template<> struct store<priority> : yes<priority> {};
00215     struct priority_keyword : TAO_PEGTL_STRING("priority") {};
00216     struct priority_command
00217         : seq<TAO_PEGTL_STRING("with"), p<must<priority_keyword>>,
00218             p<must<priority>>> {};
00219
00220     // Catch unsupported statements.
00221     struct unsupported_statement : failure {};
00222     struct sa0Catch
00223         : if_must<a0_statement<identifier>, unsupported_statement> {};
00224     struct salCatch

```

```

00225      : if_must<a1_statement<identifier,
00226          star<not_one';'>>, unsupported_statement> {};
00227  struct s_catch : sor<sa0_catch, sal_catch> {};
00228
00229 // Catch unsuported rules.
00230  struct invalid_rule : failure {};
00231  struct sa0_ruleCatch : if_must<a0_statement<identifier>, invalid_rule> {};
00232  struct sal_ruleCatch
00233      : if_must<a1_statement<identifier,
00234          star<not_one';'>>, invalid_rule> {};
00235  struct ruleCatch : sor<sa0_ruleCatch, sal_ruleCatch> {};
00236
00237 // Filter chain.
00238  struct rule
00239      : if_must<sor<action, ruleCatch>, opt<p<priority_command>>,
00240          opt<p<condition>>, eos> {};
00241  template<> struct store<rule> : replace_with_first_child<rule> {};
00242  struct chain : chain_name {};
00243  template<> struct store<chain> : yes<chain> {};
00244  struct chain_container
00245      : t_named_block<TAO_PEGTL_STRING("chain"), chain, rule> {};
00246  template<> struct store<chain_container>
00247      : replace_with_first_child<chain_container> {};
00248
00249 // Default filter action (keep node, no content).
00250  struct default_action_option : sor<accept, reject> {};
00251  struct default_action
00252      : al_statement<TAO_PEGTL_STRING("default_action"),
00253          default_action_option> {};
00254  template<> struct store<default_action>
00255      : yes_without_content<default_action> {};
00256
00257 // UDP connection block.
00258  struct s_port : al_statement<TAO_PEGTL_STRING("port"), port> {};
00259  struct s_address : al_statement<TAO_PEGTL_STRING("address"), address> {};
00260  struct s_max_bitrate
00261      : al_statement<TAO_PEGTL_STRING("max_bitrate"), max_bitrate> {};
00262  struct udp
00263      : t_block<TAO_PEGTL_STRING("udp"),
00264          s_port, s_address, s_max_bitrate, s_catch> {};
00265  template<> struct store<udp> : yes_without_content<udp> {};
00266
00267 // Serial port block.
00268  struct s_device : al_statement<TAO_PEGTL_STRING("device"), device> {};
00269  struct s_baudrate : al_statement<TAO_PEGTL_STRING("baudrate"), baudrate> {};
00270  struct s_flow_control
00271      : al_statement<TAO_PEGTL_STRING("flow_control"), flow_control> {};
00272  struct s_reload : al_statement<TAO_PEGTL_STRING("preload"), preload> {};
00273  struct serial
00274      : t_block<TAO_PEGTL_STRING("serial"),
00275          s_device, s_baudrate, s_flow_control, s_reload, s_catch> {};
00276  template<> struct store<serial> : yes_without_content<serial> {};
00277
00278 // Combine grammar.
00279  struct block : sor<udp, serial, chain_container> {};
00280  struct statement : sor<default_action, s_catch> {};
00281  struct element : sor<comment, block, statement> {};
00282  struct elements : plus<pad<element, ignored>> {};
00283  struct grammar : seq<must<elements>, eof> {};
00284
00285 #ifdef __clang__
00286     #pragma clang diagnostic push
00287     #pragma clang diagnostic ignored "-Wglobal-constructors"
00288     #pragma clang diagnostic ignored "-Wexit-time-destructors"
00289 #endif
00290
00291 // Error messages
00292
00293  template<>
00294  const std::string error<eos>::error_message;
00295
00296  template<>
00297  const std::string error<opening_brace>::error_message;
00298
00299  template<>
00300  const std::string error<closing_brace>::error_message;
00301
00302  template<>
00303  const std::string error<unsupported_statement>::error_message;
00304
00305  template<>
```

```
00306     const std::string error<default_action_option>::error_message;
00307
00308     template<>
00309     const std::string error<port>::error_message;
00310
00311     template<>
00312     const std::string error<address>::error_message;
00313
00314     template<>
00315     const std::string error<max_bitrate>::error_message;
00316
00317     template<>
00318     const std::string error<device>::error_message;
00319
00320     template<>
00321     const std::string error<baudrate>::error_message;
00322
00323     template<>
00324     const std::string error<flow_control>::error_message;
00325
00326     template<>
00327     const std::string error<preload>::error_message;
00328
00329     template<>
00330     const std::string error<chain_name>::error_message;
00331
00332     template<>
00333     const std::string error<chain>::error_message;
00334
00335     template<>
00336     const std::string error<call>::error_message;
00337
00338     template<>
00339     const std::string error<goto_>::error_message;
00340
00341     template<>
00342     const std::string error<invalid_rule>::error_message;
00343
00344     template<>
00345     const std::string error<condition_value>::error_message;
00346
00347     template<>
00348     const std::string error<dest>::error_message;
00349
00350     template<>
00351     const std::string error<source>::error_message;
00352
00353     template<>
00354     const std::string error<mavaddr>::error_message;
00355
00356     template<>
00357     const std::string error<integer>::error_message;
00358
00359     template<>
00360     const std::string error<priority>::error_message;
00361
00362     template<>
00363     const std::string error<priority_keyword>::error_message;
00364
00365     template<>
00366     const std::string error<elements>::error_message;
00367
00368 // Default error message.
00369     template<typename T>
00370     const std::string error<T>::error_message =
00371         "parse error matching " + internal::demangle<T>();
00372
00373 #ifdef __clang__
00374     #pragma clang diagnostic pop
00375 #endif
00376
00377     /// @endcond
00378
00379
00380     /** Parses given input into an abstract syntax tree.
00381     *
00382     * \note The returned AST is only valid while the input exists. Therefore,
00383     *       the input should be kept until the AST passes out of scope.
00384     *
00385     * \tparam Input The type of input, usually either read_input or
00386     *               string_input (see PEGTL).
00386
```

```

00387     * \param in The input, from read_input etc, to parse.
00388     * \returns The abstract syntax tree parsed from the input.
00389     */
00390     template <typename Input>
00391     std::unique_ptr<config::parse_tree::node> parse(Input &in)
00392     {
00393         return parse_tree::parse<grammar, config::store>(in);
00394     }
00395
00396     std::ostream &print_node(
00397         std::ostream &os, const config::parse_tree::node &n,
00398         bool print_location, const std::string &s = "");
00399 }
00400
00401
00402     std::ostream &operator<<(
00403         std::ostream &os, const config::parse_tree::node &node);
00404
00405
00406 #endif // CONFIG_GRAMMAR_HPP_

```

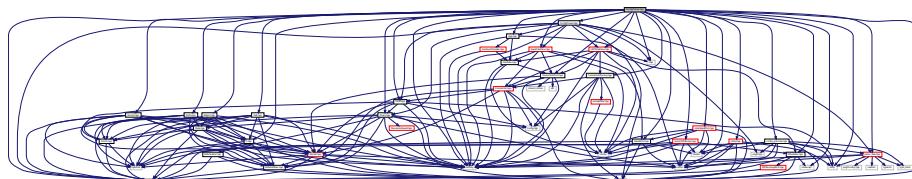
16.35 ConfigParser.cpp File Reference

```

#include <map>
#include <memory>
#include <iostream>
#include <stdexcept>
#include <string>
#include <utility>
#include <vector>
#include "Accept.hpp"
#include "App.hpp"
#include "Call.hpp"
#include "Chain.hpp"
#include "ConfigParser.hpp"
#include "config_grammar.hpp"
#include "ConnectionFactory.hpp"
#include "ConnectionPool.hpp"
#include "Filter.hpp"
#include "GoTo.hpp"
#include "If.hpp"
#include "IPAddress.hpp"
#include "parse_tree.hpp"
#include "Reject.hpp"
#include "SerialInterface.hpp"
#include "SerialPort.hpp"
#include "UDPInterface.hpp"
#include "UnixSerialPort.hpp"
#include "UnixUDPSocket.hpp"
#include "utility.hpp"

```

Include dependency graph for ConfigParser.cpp:



16.36 ConfigParser.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <map>
00019 #include <memory>
00020 #include <iostream>
00021 #include <stdexcept>
00022 #include <string>
00023 #include <utility>
00024 #include <vector>
00025
00026 #include "Accept.hpp"
00027 #include "App.hpp"
00028 #include "Call.hpp"
00029 #include "Chain.hpp"
00030 #include "ConfigParser.hpp"
00031 #include "config_grammar.hpp"
00032 #include "ConnectionFactory.hpp"
00033 #include "ConnectionPool.hpp"
00034 #include "Filter.hpp"
00035 #include "GoTo.hpp"
00036 #include "If.hpp"
00037 #include "IPAddress.hpp"
00038 #include "parse_tree.hpp"
00039 #include "Reject.hpp"
00040 #include "SerialInterface.hpp"
00041 #include "SerialPort.hpp"
00042 #include "UDPInterface.hpp"
00043 #include "UnixSerialPort.hpp"
00044 #include "UnixUDPSocket.hpp"
00045 #include "utility.hpp"
00046
00047
00048 /** Construct a map of non default chains.
00049 *
00050 * \relates ConfigParser
00051 * \param root Root of configuration AST.
00052 * \returns Map of chain names to chains.
00053 */
00054 std::map<std::string, std::shared_ptr<Chain>> init_chains(
00055     const config::parse_tree::node &root)
00056 {
00057     std::map<std::string, std::shared_ptr<Chain>> chains;
00058
00059     // Loop through top AST nodes creating chains for.
00060     for (auto &node : root.children)
00061     {
00062         if (node->name() == "config::chain")
00063         {
00064             if (node->has_content() && node->content() != "default")
00065             {
00066                 chains[node->content()]
00067                     = std::make_shared<Chain>(node->content());
00068             }
00069         }
00070     }
00071
00072     return chains;
00073 }
00074
00075
00076 /** Construct a \ref Rule with action from AST, priority, and condition.
00077 *
```

```

00078 * \relates ConfigParser
00079 * \param root AST action node.
00080 * \param priority The priority to use when constructing the action. No
00081 *      priority if {}. If the AST node is a reject action the priority will be
00082 *      ignored.
00083 * \param condition The condition to use when constructing the action.
00084 * \param chains Map of chain names to chains for call and goto actions.
00085 * \returns The action (or rule) parsed from the given AST node, priority, and
00086 *      condition.
00087 * \throws std::invalid_argument if the action attempts to 'call' or 'goto' the
00088 *      default chain.
00089 * \throws std::runtime_error if the action is not one of 'accept', 'reject',
00090 *      'call', or 'goto'.
00091 */
00092 std::unique_ptr<Rule> parse_action(
00093     const config::parse_tree::node &root,
00094     std::optional<int> priority,
00095     std::optional<If> condition,
00096     const std::map<std::string, std::shared_ptr<Chain>> &chains)
00097 {
00098     // Parse accept rule.
00099     if (root.name() == "config::accept")
00100     {
00101         if (priority)
00102         {
00103             return std::make_unique<Accept>(
00104                 priority.value(), std::move(condition));
00105         }
00106         else
00107         {
00108             return std::make_unique<Accept>(std::move(condition));
00109         }
00110     }
00111     // Parse reject rule.
00112     else if (root.name() == "config::reject")
00113     {
00114         return std::make_unique<Reject>(std::move(condition));
00115     }
00116     // Parse call rule.
00117     else if (root.name() == "config::call")
00118     {
00119         if (root.content() == "default")
00120         {
00121             throw std::invalid_argument("cannot 'call' the default chain");
00122         }
00123         if (priority)
00124         {
00125             return std::make_unique<Call>(
00126                 chains.at(root.content()),
00127                 priority.value(),
00128                 std::move(condition));
00129         }
00130         else
00131         {
00132             return std::make_unique<Call>(
00133                 chains.at(root.content()),
00134                 std::move(condition));
00135         }
00136     }
00137 }
00138 // Parse call goto.
00139 else if (root.name() == "config::goto_")
00140 {
00141     if (root.content() == "default")
00142     {
00143         throw std::invalid_argument("cannot 'goto' the default chain");
00144     }
00145     if (priority)
00146     {
00147         return std::make_unique<GoTo>(
00148             chains.at(root.content()),
00149             priority.value(),
00150             std::move(condition));
00151     }
00152     else
00153     {
00154         return std::make_unique<GoTo>(
00155             chains.at(root.content()),
00156             std::move(condition));
00157     }
00158 }
```

```
00159     }
00160
00161     // Only called if the AST is invalid, this can't be called as long as
00162     // config_grammar.hpp does not contain any bugs.
00163     // LCOV_EXCL_START
00164     throw std::runtime_error("unknown action " + root.name());
00165     // LCOV_EXCL_STOP
00166 }
00167
00168
00169 /** Add \ref Rule's from AST to a \ref Chain.
00170 */
00171 * \relates ConfigParser
00172 * \param chain Chain to add rules to.
00173 * \param root AST chain node containing rules.
00174 * \param chains Map of chain names to chains for call and goto actions.
00175 */
00176 void parse_chain(
00177     Chain &chain,
00178     const config::parse_tree::node &root,
00179     const std::map<std::string, std::shared_ptr<Chain>> &chains)
00180 {
00181     // Loop through each children.
00182     for (auto &node : root.children)
00183     {
00184         std::optional<int> priority;
00185         std::optional<If> condition;
00186
00187         // Loop through rule options.
00188         for (auto &child : node->children)
00189         {
00190             // Extract priority.
00191             if (child->name() == "config::priority")
00192             {
00193                 priority = std::stoi(child->content());
00194             }
00195             // Extract condition.
00196             else if (child->name() == "config::condition")
00197             {
00198                 condition = parse_condition(*child);
00199             }
00200         }
00201
00202         // Create and add the new rule.
00203         chain.append(
00204             parse_action(
00205                 *node, std::move(priority), std::move(condition), chains));
00206     }
00207 }
00208
00209
00210 /** Construct conditional (\ref If) from AST.
00211 */
00212 * \relates ConfigParser
00213 * \param root AST conditional node.
00214 * \returns The conditional constructed from the given AST node.
00215 */
00216 If parse_condition(const config::parse_tree::node &root)
00217 {
00218     If condition;
00219
00220     // Parse condition options.
00221     for (auto &child : root.children)
00222     {
00223         // Parse packet type.
00224         if (child->name() == "config::packet_type")
00225         {
00226             condition.type(child->content());
00227         }
00228         // Parse source MAVLink address.
00229         else if (child->name() == "config::source")
00230         {
00231             condition.from(child->content());
00232         }
00233         // Parse destination MAVLink address.
00234         else if (child->name() == "config::dest")
00235         {
00236             condition.to(child->content());
00237         }
00238     }
00239 }
```

```

00240     return condition;
00241 }
00242
00243
00244 /** Parse \ref Filter from AST.
00245 *
00246 * \relates ConfigParser
00247 * \param root Root of configuration AST.
00248 * \returns The \ref Filter parsed from the AST.
00249 */
00250 std::unique_ptr<Filter> parse_filter(const config::parse_tree::node &root)
00251 {
00252     Chain default_chain("default");
00253     bool default_action = false;
00254     std::map<std::string, std::shared_ptr<Chain>> chains = init_chains(root);
00255
00256     // Look through top nodes.
00257     for (auto &node : root.children)
00258     {
00259         // Parse chain.
00260         if (node->name() == "config::chain")
00261         {
00262             // Parse default chain.
00263             if (node->content() == "default")
00264             {
00265                 parse_chain(default_chain, *node, chains);
00266             }
00267             // Parse named chain.
00268             else
00269             {
00270                 parse_chain(*chains.at(node->content()), *node, chains);
00271             }
00272         }
00273         // Parse default filter action.
00274         else if (node->name() == "config::default_action")
00275         {
00276             default_action = node->children[0]->name() == "config::accept";
00277         }
00278     }
00279
00280     // Construct the filter.
00281     return std::make_unique<Filter>(std::move(default_chain), default_action);
00282 }
00283
00284
00285 /** Parse UDP and serial port interfaces from AST root.
00286 *
00287 * \relates ConfigParser
00288 * \param root The root of the AST to create \ref Interface's from.
00289 * \param filter The packet \ref Filter to use for the interfaces.
00290 * \returns A vector of UDP and serial port interfaces.
00291 */
00292 std::vector<std::unique_ptr<Interface>> parse_interfaces(
00293     const config::parse_tree::node &root, std::unique_ptr<Filter> filter)
00294 {
00295     std::shared_ptr<Filter> shared_filter = std::move(filter);
00296     std::vector<std::unique_ptr<Interface>> interfaces;
00297     auto connection_pool = std::make_shared<ConnectionPool>();
00298
00299     // Loop over each node of the root AST node.
00300     for (auto &node : root.children)
00301     {
00302         // Parse UDP interface.
00303         if (node->name() == "config::udp")
00304         {
00305             interfaces.push_back(
00306                 parse_udp(*node, shared_filter, connection_pool));
00307         }
00308         // Parse serial port interface.
00309         else if (node->name() == "config::serial")
00310         {
00311             interfaces.push_back(
00312                 parse_serial(*node, shared_filter, connection_pool));
00313         }
00314     }
00315
00316     return interfaces;
00317 }
00318
00319
00320 /** Parse a serial port interface from an AST.

```

```

00321  *
00322  * \relates ConfigParser
00323  * \param root The serial port node to parse.
00324  * \param filter The \ref Filter to use for the \ref SerialInterface.
00325  * \param pool The connection pool to add the interface's connection to.
00326  * \returns The serial port interface parsed from the AST and using the given
00327  *         filter and connection pool.
00328  * \throws std::invalid_argument if the device string is missing.
00329  */
00330 std::unique_ptr<SerialInterface> parse_serial(
00331     const config::parse_tree::node &root,
00332     std::shared_ptr<Filter> filter,
00333     std::shared_ptr<ConnectionPool> pool)
00334 {
00335     // Default settings.
00336     std::optional<std::string> device;
00337     unsigned long baud_rate = 9600;
00338     SerialPort::Feature features = SerialPort::DEFAULT;
00339     std::vector<MAVAddress> preload;
00340
00341     // Extract settings from AST.
00342     for (auto &node : root.children)
00343     {
00344         // Extract device string.
00345         if (node->name() == "config::device")
00346         {
00347             device = node->content();
00348         }
00349         // Extract device baud rate.
00350         else if (node->name() == "config::baudrate")
00351         {
00352             baud_rate = static_cast<unsigned long>(std::stol(node->content()));
00353         }
00354         // Extract flow control.
00355         else if (node->name() == "config::flow_control")
00356         {
00357             if (to_lower(node->content()) == "yes")
00358             {
00359                 features = SerialPort::FLOW_CONTROL;
00360             }
00361         }
00362         // Extract preloaded address.
00363         else if (node->name() == "config::preload")
00364         {
00365             preload.push_back(MAVAddress(node->content()));
00366         }
00367     }
00368
00369     // Throw error if no device was given.
00370     if (!device.has_value())
00371     {
00372         throw std::invalid_argument("missing device string");
00373     }
00374
00375     // Construct serial interface.
00376     auto port = std::make_unique<UnixSerialPort>(
00377         device.value(), baud_rate, features);
00378     auto connection = std::make_unique<Connection>(
00379         device.value(), filter, false);
00380
00381     for (const auto &addr : preload)
00382     {
00383         connection->add_address(addr);
00384     }
00385
00386     return std::make_unique<SerialInterface>(
00387         std::move(port), pool, std::move(connection));
00388 }
00389
00390
00391 /** Parse a UDP interface from an AST.
00392  *
00393  * \relates ConfigParser
00394  * \param root The UDP node to parse.
00395  * \param filter The \ref Filter to use for the \ref UDPInterface.
00396  * \param pool The connection pool to add the interface's connections to.
00397  * \returns The UDP interface parsed from the AST and using the given filter
00398  *         and connection pool.
00399  */
00400 std::unique_ptr<UDPInterface> parse_udp(
00401     const config::parse_tree::node &root,

```

```

00402     std::shared_ptr<Filter> filter,
00403     std::shared_ptr<ConnectionPool> pool)
00404 {
00405     unsigned int port = 14500;
00406     std::optional<IPAddress> address;
00407     unsigned long max_bitrate = 0;
00408
00409     // Loop over options for UDP interface.
00410     for (auto &node : root.children)
00411     {
00412         // Parse port number.
00413         if (node->name() == "config::port")
00414         {
00415             port = static_cast<unsigned int>(std::stol(node->content()));
00416         }
00417         // Parse IP address and optionally port number.
00418         else if (node->name() == "config::address")
00419         {
00420             address = IPAddress(node->content());
00421         }
00422         // Parse bitrate limit.
00423         else if (node->name() == "config::max_bitrate")
00424         {
00425             max_bitrate = static_cast<unsigned long>(
00426                 std::stoll(node->content()));
00427         }
00428     }
00429
00430     // Construct the UDP interface.
00431     auto socket = std::make_unique<UnixUDPSocket>(port, address, max_bitrate);
00432     auto factory = std::make_unique<ConnectionFactory><>(filter, false);
00433     return std::make_unique<UDPInterface>(
00434         std::move(socket), pool, std::move(factory));
00435 }
00436
00437
00438 /** Construct a configuration parser from a file.
00439 */
00440 * \param filename The path of the configuration file to parse.
00441 * \throws std::runtime_error if the configuration file cannot be parsed.
00442 */
00443 ConfigParser::ConfigParser(std::string filename)
00444     : in_(filename)
00445 {
00446     root_ = config::parse(in_);
00447
00448     if (root_ == nullptr)
00449     {
00450         // It is technically impossible parsing errors should be raised as a
00451         // parse_error.
00452         // LCOV_EXCL_START
00453         throw std::runtime_error(
00454             "Unexpected error while parsing configuration file.");
00455         // LCOV_EXCL_STOP
00456     }
00457 }
00458
00459
00460 /** Build a mavtables application from the AST contained by the parser.
00461 */
00462 * \returns A mavtables application.
00463 */
00464 std::unique_ptr<App> ConfigParser::make_app()
00465 {
00466     auto filter = parse_filter(*root_);
00467     auto interfaces = parse_interfaces(*root_, std::move(filter));
00468     return std::make_unique<App>(std::move(interfaces));
00469 }
00470
00471
00472 /** Print the configuration settings to the given output stream.
00473 */
00474 * An example (that of test/mavtables.conf) is:
00475 *
00476 * ``
00477 * ===== test/mavtables.conf =====
00478 * :001: default_action
00479 * :001: | accept
00480 * :004: udp
00481 * :005: | port 14500
00482 * :006: | address 127.0.0.1

```

```

00483 * :007: | max_bitrate 8388608
00484 * :011: serial
00485 * :012: | device ./ttyS0
00486 * :013: | baudrate 115200
00487 * :014: | flow_control yes
00488 * :015: | preload 1.1
00489 * :016: | preload 62.34
00490 * :020: chain default
00491 * :022: | call some_chain10
00492 * :022: | | condition
00493 * :022: | | | source 127.1
00494 * :022: | | | dest 192.0
00495 * :023: | reject
00496 * :027: chain some_chain10
00497 * :029: | accept
00498 * :029: | | priority 99
00499 * :029: | | condition
00500 * :029: | | | dest 192.0
00501 * :030: | accept
00502 * :030: | | condition
00503 * :030: | | | packet_type PING
00504 * :031: | accept
00505 *
00506 *
00507 * \relates ConfigParser
00508 * \param os The output stream to print to.
00509 * \param config_parser The configuration parser to print.
00510 * \returns The output stream.
00511 */
00512 std::ostream &operator<<(std::ostream &os, const ConfigParser &config_parser)
00513 {
00514     os << *config_parser.root_;
00515     return os;
00516 }

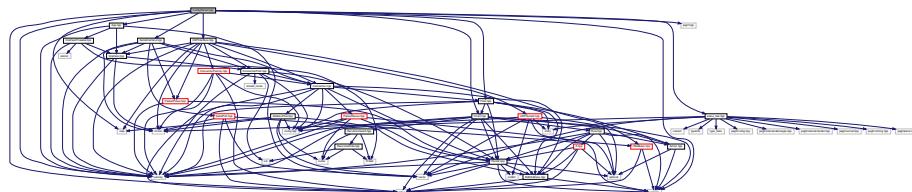
```

16.37 ConfigParser.hpp File Reference

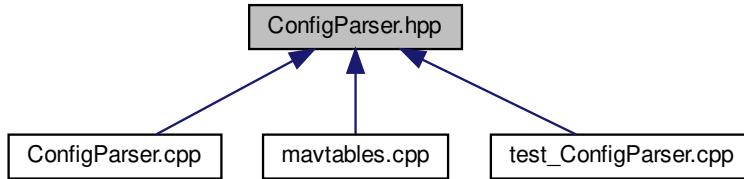
```

#include <map>
#include <memory>
#include <iostream>
#include <string>
#include <pegtl.hpp>
#include "App.hpp"
#include "Chain.hpp"
#include "Filter.hpp"
#include "parse_tree.hpp"
#include "SerialInterface.hpp"
#include "UDPIface.hpp"
Include dependency graph for ConfigParser.hpp:

```



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



Classes

- class [ConfigParser](#)

Functions

- `std::map< std::string, std::shared_ptr< Chain > > init_chains (const config::parse_tree::node &root)`
- `std::unique_ptr< Rule > parse_action (const config::parse_tree::node &root, std::optional< int > priority, std::optional< If > condition, const std::map< std::string, std::shared_ptr< Chain > >&chains)`
- `void parse_chain (Chain &chain, const config::parse_tree::node &root, const std::map< std::string, std::shared_ptr< Chain > >&chains)`
- `If parse_condition (const config::parse_tree::node &root)`
- `std::unique_ptr< Filter > parse_filter (const config::parse_tree::node &root)`
- `std::vector< std::unique_ptr< Interface > > parse_interfaces (const config::parse_tree::node &root, std::unique_ptr< Filter > filter)`
- `std::unique_ptr< SerialInterface > parse_serial (const config::parse_tree::node &root, std::shared_ptr< Filter > filter, std::shared_ptr< ConnectionPool > pool)`
- `std::unique_ptr< UDPInterface > parse_udp (const config::parse_tree::node &root, std::shared_ptr< Filter > filter, std::shared_ptr< ConnectionPool > pool)`
- `std::ostream & operator<< (std::ostream &os, const ConfigParser &config_parser)`

16.37.1 Function Documentation

16.37.1.1 init_chains()

```
std::map<std::string, std::shared_ptr<Chain> > init_chains (
    const config::parse_tree::node & root ) [related]
```

16.37.1.2 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const ConfigParser & config_parser )
```

16.37.1.3 parse_action()

```
std::unique_ptr<Rule> parse_action (
    const config::parse_tree::node & root,
    std::optional< int > priority,
    std::optional< If > condition,
    const std::map< std::string, std::shared_ptr< Chain >> & chains ) [related]
```

16.37.1.4 parse_chain()

```
void parse_chain (
    Chain & chain,
    const config::parse_tree::node & root,
    const std::map< std::string, std::shared_ptr< Chain >> & chains ) [related]
```

16.37.1.5 parse_condition()

```
If parse_condition (
    const config::parse_tree::node & root ) [related]
```

16.37.1.6 parse_filter()

```
std::unique_ptr<Filter> parse_filter (
    const config::parse_tree::node & root ) [related]
```

16.37.1.7 parse_interfaces()

```
std::vector<std::unique_ptr<Interface>> parse_interfaces (
    const config::parse_tree::node & root,
    std::unique_ptr< Filter > filter ) [related]
```

16.37.1.8 parse_serial()

```
std::unique_ptr<SerialInterface> parse_serial (
    const config::parse_tree::node & root,
    std::shared_ptr< Filter > filter,
    std::shared_ptr< ConnectionPool > pool ) [related]
```

16.37.1.9 parse_udp()

```
std::unique_ptr<UDPIInterface> parse_udp (
    const config::parse_tree::node & root,
    std::shared_ptr< Filter > filter,
    std::shared_ptr< ConnectionPool > pool ) [related]
```

16.38 ConfigParser.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef CONFIGPARSER_HPP_
00019 #define CONFIGPARSER_HPP_
00020
00021
00022 #include <map>
00023 #include <memory>
00024 #include <iostream>
00025 #include <string>
00026
00027 #include <pegtl.hpp>
00028
00029 #include "App.hpp"
00030 #include "Chain.hpp"
00031 #include "Filter.hpp"
00032 #include "parse_tree.hpp"
00033 #include "SerialInterface.hpp"
00034 #include "UDPIInterface.hpp"
00035
00036
00037 std::map<std::string, std::shared_ptr<Chain>> init_chains(
00038     const config::parse_tree::node &root);
00039
00040 std::unique_ptr<Rule> parse_action(
00041     const config::parse_tree::node &root,
00042     std::optional<int> priority,
00043     std::optional<If> condition,
00044     const std::map<std::string, std::shared_ptr<Chain>> &chains);
00045
00046 void parse_chain(
00047     Chain &chain,
```

```

00048     const config::parse_tree::node &root,
00049     const std::map<std::string, std::shared_ptr<Chain>> &chains);
00050
00051 If parse_condition(const config::parse_tree::node &root);
00052
00053 std::unique_ptr<Filter> parse_filter(const config::parse_tree::node &root);
00054
00055 std::vector<std::unique_ptr<Interface>> parse_interfaces(
00056     const config::parse_tree::node &root, std::unique_ptr<Filter> filter);
00057
00058 std::unique_ptr<SerialInterface> parse_serial(
00059     const config::parse_tree::node &root,
00060     std::shared_ptr<Filter> filter,
00061     std::shared_ptr<ConnectionPool> pool);
00062
00063 std::unique_ptr<UDPInterface> parse_udp(
00064     const config::parse_tree::node &root,
00065     std::shared_ptr<Filter> filter,
00066     std::shared_ptr<ConnectionPool> pool);
00067
00068
00069 /** Configuration file parser.
00070 */
00071 * Used to parse a configuration file and create an instance of the mavtables
00072 * application.
00073 */
00074 class ConfigParser
00075 {
00076     public:
00077         ConfigParser(std::string filename);
00078         ConfigParser(const ConfigParser &other) = delete;
00079         ConfigParser(ConfigParser &&other) = delete;
00080         std::unique_ptr<App> make_app();
00081         ConfigParser &operator=(const ConfigParser &other) = delete;
00082         ConfigParser &operator=(ConfigParser &&other) = delete;
00083
00084     friend std::ostream &operator<<(
00085         std::ostream &os, const ConfigParser &config_parser);
00086
00087     protected:
00088         tao::pegtl::read_input<> in_;
00089         std::unique_ptr<config::parse_tree::node> root_;
00090     };
00091
00092
00093 std::ostream &operator<<(std::ostream &os, const ConfigParser &config_parser);
00094
00095
00096 #endif // CONFIGPARSER_HPP_

```

16.39 configuration.md File Reference

16.40 configuration.md

```

00001 Configuration {#configuration}
00002 -----
00003
00004 * [MAVLink Addressing] (#mavlink-addressing)
00005 * [MAVLink Subnets] (#mavlink-subnets)
00006 * [Basic Grammar] (#basic-grammar)
00007 * [Identifiers] (#identifiers)
00008 * [Statements] (#statements)
00009 * [Blocks] (#blocks)
00010 * [Comments] (#comments)
00011 * [Global Settings] (#global-settings)
00012 * [udp block] (#udp-block)
00013 * [port statement] (#port-statement)
00014 * [address statement] (#address-statement)
00015 * [max_bitrate statement] (#max_bitrate-statement)
00016 * [serial block] (#serial-block)
00017 * [device statement] (#device-statement)
00018 * [baudrate statement] (#baudrate-statement)

```

```

00019 * [flow_control statement] (#flow_control-statement)
00020 * [preload statement] (#preload-statement)
00021 * [chain block] (#chain-block)
00022 * [Rules] (#rules)
00023 * [Action] (#action)
00024 * [Priority] (#priority)
00025 * [Condition] (#condition)
00026 * [\<packet type\>] (#packet-type)
00027 * [\<source address\>] (#source-address)
00028 * [\<dest address\>] (#dest-address)
00029
00030
00031 mavtables' configuration files are used to define default actions, interfaces,
00032 and filter chains. A full example configuration file can be found at
00033 ['./examples/mavtables.conf'] (./examples/mavtables.conf).
00034
00035
00036 # MAVLink Addressing
00037
00038 A MAVLink address consists of 2 octets. The first being the system ID and the
00039 second the component ID. Therefore, a MAVLink address looks like half of an
00040 IPv4 address.
00041 ``
00042 <system ID>.<component ID>
00043 ``
00044
00045
00046 ## MAVLink Subnets
00047
00048 MAVLink subnets are similar to IP subnets and thus it is recommended that the
00049 reader understand IP subnets before continuing.
00050
00051 There are four representations of MAVLink subnets:
00052
00053 1. "\<System ID>.<Component ID\>:\<System ID mask\>.\<Component ID mask\>""
00054 2. "\<System ID>.<Component ID\>/\<bits\>"
00055 3. "\<System ID>.<Component ID\>\\\<bits\>"
00056 4. "\<System ID>.<Component ID\>"
00057
00058 The first form is self explanatory, but the 2nd and 3rd are not as simple.
00059 In the 2nd case the number of bits (0 - 16) is the number of bits from the
00060 left that must match for an address to be in the subnet. The 3rd form is
00061 like the 2nd, but does not require the system ID (first octet) to match and
00062 starts with the number of bits of the component ID (0 - 8) that must match
00063 from the left for an address to be in the subnet. The last form is
00064 shorthand for "<System ID>.<Component ID>/16", that is an exact match.
00065
00066 Below is a table relating the slash postfix to the subnet mask in \<System
00067 mask\>.\<Component mask\> notation.
00068
00069 | Mask with '/' | Mask with '\' | Postfix (\#bits) |
00070 | -----:| -----:| -----:|
00071 | 255.255 | out of range | 16 |
00072 | 255.254 | out of range | 15 |
00073 | 255.252 | out of range | 14 |
00074 | 255.248 | out of range | 13 |
00075 | 255.240 | out of range | 12 |
00076 | 255.224 | out of range | 11 |
00077 | 255.192 | out of range | 10 |
00078 | 255.128 | out of range | 9 |
00079 | 255.0 | 0.255 | 8 |
00080 | 254.0 | 0.254 | 7 |
00081 | 252.0 | 0.252 | 6 |
00082 | 248.0 | 0.248 | 5 |
00083 | 240.0 | 0.240 | 4 |
00084 | 224.0 | 0.224 | 3 |
00085 | 192.0 | 0.192 | 2 |
00086 | 128.0 | 0.128 | 1 |
00087 | 0.0 | 0 | 0 |
00088
00089
00090 # Basic Grammar
00091
00092 There are four major types of grammar elements in configuration files. These
00093 are identifiers, statements, blocks, and comments. Configuration files have
00094 a similar syntax to the C programming language and like C they are not
00095 whitespace sensitive.
00096
00097
00098 ## Identifiers
00099

```

```
00100 An identifier is a name containing any combination of letters (upper and lower),
00101 digits, and the underscores ('\_'). However, it must not start with a number.
00102 This is the same definition as identifiers in the C programming language.
00103
00104
00105 ## Statements
00106
00107 A statement is defined as an identifier, followed by a value, and then the *end
00108 of statement* character ';''. Therefore, it takes the form of
00109 ``
00110 <identifier> <value>;
00111 ``
00112 An example would be:
00113 ``
00114 boadrate 57600;
00115 ``
00116 where 'boadrate' is the identifier, '57600' is the value (in this case
00117 a number) and it ends with ';'.
00118
00119
00120 ## Blocks
00121
00122 A block is a sectioning construct that can contain one or more statements, or in
00123 the special case of filter blocks, a rule. Blocks consist of a type and
00124 optionally an identifier followed by '{', one or more statements (or rules)
00125 and finally ending in '}'. The format is:
00126 ``
00127 <type> <optional identifier> {
00128     <statement 1>;
00129     <statement 2>;
00130     <statement ...>;
00131     <statement n>;
00132 }
00133 ``
00134 An example would be:
00135 ``
00136 udp {
00137     port 14555;
00138     address 127.0.0.1;
00139 }
00140 ``
00141
00142
00143 ## Comments
00144
00145 The comment character is '#'. Everything after this character to the end of
00146 the line is considered a comment and will be ignored. An example of using
00147 comments is:
00148 ``
00149 udp { # UDP Inteface
00150     port 14555;           # port number, the default is 14500
00151     address 127.0.0.1;   # listen on localhost only, the default is any address
00152 }
00153 ``
00154
00155
00156
00157 # Global Settings
00158
00159 The following sections document the components of a configuration file.
00160
00161 ## default_action statement
00162
00163 Set the default action to take when the filter does not determine what to do
00164 with a packet. The options are:
00165 * 'accept' - to accept packets by default
00166 * 'reject' - to reject packets by default
00167
00168 To accept packets by default use:
00169 ``
00170 default_action accept;
00171 ``
00172 To reject packets by default use:
00173 ``
00174 default_action reject;
00175 ``
00176
00177
00178
00179 # udp block
00180
```

```
00181 The 'udp' block defines a UDP interface to listen for connections on. An
00182 example is:
00183 ``
00184 udp {
00185     port 14555;
00186     address 127.0.0.1;
00187     max_bitrate 8388608;
00188 }
00189 ``
00190
00191 There is no limit to the number of UDP interfaces that can be defined.
00192
00193
00194 ## port statement
00195
00196 A statement that sets the port number to listen on. The format is:
00197 ``
00198 port <port number>;
00199 ``
00200
00201 An example is:
00202 ``
00203 port 14555;
00204 ``
00205
00206 This is the only required statement in a 'udp' block.
00207
00208
00209 ## address statement (optional)
00210
00211 A statement that sets the IP address to listen on. The format is:
00212 ``
00213 address <IP v4 address>;
00214 ``
00215
00216 An example is:
00217 ``
00218 address 127.0.0.1;
00219 ``
00220 which restricts mavtables to only listening for connections from 'localhost'.
00221
00222 If not provided the default is to listen on every address and therefore to
00223 accept connections from remote systems (subject to the firewall's running on the
00224 host operating system).
00225
00226
00227 ## max_bitrate statement (optional)
00228
00229 A statement that sets the bitrate limit (in bits per second) when transmitting
00230 packets over UDP. The format is:
00231 ``
00232 max_bitrate <maximum bitrate>
00233 ``
00234
00235 An example is:
00236 ``
00237 max_bitrate 8388608; # 8 Mbps
00238 ``
00239
00240 If not provided there will not be any limit on the rate of packet transmission
00241 over UDP.
00242
00243 If downstream components tend to lose packets due to the operating system's UDP
00244 buffers filling up, consider slowing down mavtables by providing a bitrate
00245 limit. This feature was added because mavtables is much faster than components
00246 written in interpreted languages such as Python and thus must be limited so it
00247 can store packets in its own buffers to avoid overflowing the operating system
00248 buffers.
00249
00250
00251
00252 # serial block
00253
00254 The 'serial' block defines a serial port interface to listen for connections on.
00255 An example is:
00256 ``
00257 serial {
00258     device /dev/ttyUSB0;
00259     baudrate 115200;
00260     flow_control yes;
00261     preload 1.1;
```

```
00262 }
00263 ``
00264
00265 There is no limit to the number of serial port interfaces that can be defined.
00266
00267
00268 ## device statement
00269
00270 A statement that sets the serial device to listen on. The format is:
00271 ``
00272 device <device string>;
00273 ``
00274 where the '<device string>' is the path to the serial device.
00275
00276 An example is:
00277 ``
00278 device /dev/ttyUSB0;
00279 ``
00280
00281 This is the only required statement in a 'serial' block.
00282
00283
00284 ## baudrate statement (optional)
00285
00286 A statement that sets the bitrate (in bits per second) of the serial port. The
00287 format is:
00288 ``
00289 baudrate <bitrate>;
00290 ``
00291
00292 An example is:
00293 ``
00294 baudrate 115200;
00295 ``
00296 which sets the baudrate to 112.5 kbps.
00297
00298 If not provided the default is to use a baudrate of 9600 (9.4 kbps)
00299
00300
00301 ## flow_control statement (optional)
00302
00303 A statement that enables/disables hardware flow control on the serial port. The
00304 format is:
00305 ``
00306 flow_control <yes/no>;
00307 ``
00308
00309 To turn on hardware flow control:
00310 ``
00311 flow_control yes;
00312 ``
00313
00314 To turn off hardware flow control:
00315 ``
00316 flow_control no;
00317 ``
00318
00319 If not provided the default is to disable hardware flow control.
00320
00321 ## preload statement (optional)
00322
00323 A statement that can preload a MAVLink address on the serial port. Typically
00324 mavtables learns about a component when that component sends a packet to the
00325 router. In some cases a component may not send a packet until it receives one,
00326 in this case the component's address can be preloaded.
00327
00328 One particular case is when two mavtables are in use. Neither instance knows
00329 about the other because mavtables will not send any packets over a serial port
00330 if nothing has been sent to it over said serial port. By preloading addresses
00331 on both instances for the serial port connection they share, the packets from
00332 other components will be sent to the other mavtables instance and thus the
00333 instances can discover each other's components.
00334
00335 The format is:
00336 ``
00337 preload <MAVLink address>;
00338 ``
00339
00340 An example is:
00341 ``
00342 preload 1.1;
```

```
00343 ``
00344 This indicates that system 1 and component 1 can be reached on this serial port,
00345 even before that component sends a packet to the router.
00346
00347 Any number of 'preload' statements are allowed. Every address listed will be
00348 added to the serial port.
00349
00350
00351
00352 # chain block
00353
00354 A 'chain' block defines a filter chain consisting of one or more rules. Filter
00355 chains are used to define firewall rules and collect them into groups. The
00356 format is:
00357 ``
00358 chain <chain name> {
00359     <rule 1>;
00360     <rule 2>;
00361     <rule ...>;
00362     <rule n>;
00363 }
00364 ``
00365
00366 An example is:
00367 ``
00368 chain default
00369 {
00370     reject if ENCAPSULATED_DATA;
00371     accept;
00372 }
00373 ``
00374 which rejects the 'ENCAPSULATED_DATA' packet, but accepts all others.
00375
00376 The ''default'' chain is a reserved name and it must exist. The ''default''
00377 chain is the one used to filter packets before re-transmitting them. All other
00378 chains are called from the default chain or another chain via the ''call'' or
00379 ''goto'' rules.
00380
00381 There is no limit to the number of filter chains, or to the number of rules in
00382 a filter chain.
00383
00384
00385 ## Rules
00386
00387 A rule has the form
00388 ``
00389 <action> <optional priority> <optional condition>;
00390 ``
00391 an example is
00392 ``
00393 accept with priority 3 if ENCAPSULATED_DATA from 192.168 to 172.0/8;
00394 ``
00395
00396 When evaluating the filter, the first rule a packet matches will decide the fate
00397 of the packet. If a condition is not provided the rule will match all packets,
00398 otherwise the condition decides whether the rule matches or not.
00399
00400 ### Action
00401
00402 There are 4 types of actions a rule can have.
00403
00404 * 'accept' - accept the packet
00405 * 'reject' - reject the packet
00406 * 'call <chain name>' - delegate accept/reject decision to another chain,
00407     '<chain name>'. Returns to parent chain if no match in the target chain.
00408 * 'goto <chain name>' - delegate accept/reject decision to another chain,
00409     '<chain name>'. If the target chain never matches the packet then the
00410     default action, set in 'default_action' is used.
00411
00412 ### Priority
00413
00414 The priority form is:
00415 ``
00416 with priority <priority level>
00417 ``
00418 where '<priority level>' is a positive or negative integer. An example is:
00419 ``
00420 with priority -4
00421 ``
00422
00423 A greater integer indicates a higher priority. Packets have a priority of 0 by
```

```

00424 default. If communications slow down and mavtables starts to build a queue of
00425 packets to send it will send out the higher priority packets first.
00426
00427 ### Condition
00428
00429 A condition determines whether or not a packet and the address it is being sent
00430 to matches a rule. Conditions have the form:
00431 ``
00432 if <packet type> from <source address> to <dest address>;
00433 ``
00434 where '<packet type>', 'from <source address>', and 'to <dest address>' are all
00435 optional, but one is required, otherwise the rule should not have a condition at
00436 all.
00437
00438 ##### \<packet type\>
00439
00440 The name of the packet type to match (upper case). If left out all packet types
00441 will match.
00442
00443 ##### \<source address\>
00444
00445 A MAVLink subnet to test for containment of the source MAVLink address of the
00446 packet.
00447
00448 ##### \<dest address\>
00449
00450 A MAVLink subnet to test for containment of the destination MAVLink address of
00451 the packet. The destination address is not regarded as the destination
00452 contained in the packet (as that may not exist) but is the MAVLink address of
00453 where mavtables is attempting to send the packet.

```

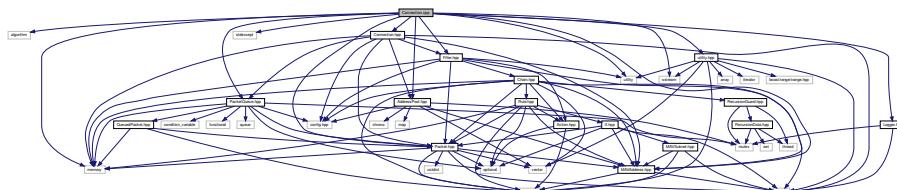
16.41 Connection.cpp File Reference

```

#include <algorithm>
#include <memory>
#include <sstream>
#include <stdexcept>
#include <utility>
#include "AddressPool.hpp"
#include "Connection.hpp"
#include "Filter.hpp"
#include "Logger.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "PacketQueue.hpp"
#include "utility.hpp"

```

Include dependency graph for Connection.cpp:



Functions

- std::ostream & [operator<<](#) (std::ostream &os, const Connection &connection)

16.41.1 Function Documentation

16.41.1.1 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const Connection & connection )
```

Print the connection name to the given output stream.

Some examples are:

- /dev/ttyUSB0
- 127.0.0.1:8000
- 127.0.0.1:14550

Definition at line 368 of file [Connection.cpp](#).

16.42 Connection.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <algorithm>
00019 #include <memory>
00020 #include <sstream>
00021 #include <stdexcept>
00022 #include <utility>
00023
00024 #include "AddressPool.hpp"
00025 #include "Connection.hpp"
00026 #include "Filter.hpp"
00027 #include "Logger.hpp"
00028 #include "MAVAddress.hpp"
00029 #include "Packet.hpp"
00030 #include "PacketQueue.hpp"
00031 #include "utility.hpp"
00032
00033
00034 /** Log an accepted/rejected packet to the \ref Logger.
00035 *
00036 * \param accept Set to true if the packet is accepted, false if the packet is
00037 * rejected.
```

```

00038 * \param packet The packet that is to be accepted/rejected.
00039 */
00040 void Connection::log_(bool accept, const Packet &packet)
00041 {
00042     if (Logger::level() >= 3)
00043     {
00044         std::stringstream ss;
00045         ss << (accept ? "accepted" : "rejected")
00046             << packet << " source ";
00047         auto connection = packet.connection();
00048
00049         if (connection == nullptr)
00050         {
00051             ss << "unknown";
00052         }
00053     else
00054     {
00055         ss << *connection;
00056     }
00057
00058     ss << " dest " << name_;
00059     Logger::log(3, ss.str());
00060 }
00061 }
00062
00063
00064 /** Send a packet to a particular address.
00065 *
00066 * If the particular address cannot be found it will be sent to every component
00067 * of the system it was sent to.
00068 *
00069 * Packets are ran through the contained \ref Filter before being placed into
00070 * the \ref PacketQueue given in the constructor. Packets are read from the
00071 * queue (for sending) by using the \ref next_packet method.
00072 *
00073 * \note This disregards the destination address of the packet.
00074 *
00075 * \param packet The packet to send.
00076 * \param dest The address to send the packet to, if this address is not
00077 * handled by this connection the packet will be silently dropped.
00078 */
00079 void Connection::send_to_address_()
00080     std::shared_ptr<const Packet> packet, const MAVAddress &dest)
00081 {
00082     // Address reachable on this connection.
00083     if (pool_->contains(dest))
00084     {
00085         // Run packet/address combination through the filter.
00086         auto [accept, priority] = filter_->will_accept(*packet, dest);
00087
00088         // Add packet to the queue.
00089         if (accept)
00090         {
00091             log_(true, *packet);
00092             queue_->push(std::move(packet), priority);
00093         }
00094     else
00095     {
00096         log_(false, *packet);
00097     }
00098 }
00099 // If the component is not reachable, send it to all components on the
00100 // system.
00101 else
00102 {
00103     bool system_found = false;
00104
00105     // Loop over addresses.
00106     for (const auto &addr : pool_->addresses())
00107     {
00108         // System can be reached on connection.
00109         if (addr.system() == dest.system())
00110         {
00111             system_found = true;
00112             auto [accept, priority] = filter_->will_accept(*packet, dest);
00113
00114             if (accept)
00115             {
00116                 log_(true, *packet);
00117                 queue_->push(std::move(packet), priority);
00118             }
00119         }
00120     }
00121 }
00122 }
```

```

00119             }
00120         }
00121     }
00122 
00123     if (system_found)
00124     {
00125         log_(false, *packet);
00126     }
00127 }
00128 }
00129
00130
00131 /** Send a packet to every address reachable on the connection.
00132 */
00133 * Packets are ran through the contained \ref Filter before being placed into
00134 * the \ref PacketQueue given in the constructor. Packets are read from the
00135 * queue (for sending) by using the \ref next_packet method.
00136 *
00137 * \note This disregards the destination address of the packet.
00138 *
00139 * \param packet The packet to send. Cannot be nullptr.
00140 */
00141 void Connection::send_to_all_(std::shared_ptr<const Packet> packet)
00142 {
00143     bool accept = false;
00144     int priority = std::numeric_limits<int>::min();
00145 
00146     // Loop over addresses.
00147     for (const auto &addr : pool_->addresses())
00148     {
00149         // Filter packet/address combination.
00150         auto [accept_, priority_] = filter_->will_accept(*packet, addr);
00151 
00152         // Update accept/priority.
00153         if (accept_)
00154         {
00155             accept = accept_;
00156             priority = std::max(priority, priority_);
00157         }
00158     }
00159 
00160     // Add packet to the queue.
00161     if (accept)
00162     {
00163         log_(true, *packet);
00164         queue_->push(std::move(packet), priority);
00165     }
00166     else
00167     {
00168         log_(false, *packet);
00169     }
00170 }
00171
00172
00173 /** Send a packet to every component of a system reachable on the connection.
00174 */
00175 * Packets are ran through the contained \ref Filter before being placed into
00176 * the \ref PacketQueue given in the constructor. Packets are read from the
00177 * queue (for sending) by using the \ref next_packet method.
00178 *
00179 * \note This disregards the destination address of the packet.
00180 *
00181 * \param packet The packet to send.
00182 */
00183 void Connection::send_to_system_()
00184     std::shared_ptr<const Packet> packet, unsigned int system)
00185 {
00186     bool system_found = false;
00187     bool accept = false;
00188     int priority = std::numeric_limits<int>::min();
00189 
00190     // Loop over addresses.
00191     for (const auto &addr : pool_->addresses())
00192     {
00193         if (system == addr.system())
00194         {
00195             system_found = true;
00196             // Filter packet/address combination.
00197             auto [accept_, priority_] = filter_->will_accept(*packet, addr);
00198 
00199             // Update accept/priority.

```

```

00200         if (accept_)
00201         {
00202             accept = accept_;
00203             priority = std::max(priority, priority_);
00204         }
00205     }
00206 }
00207
00208 // Add packet to the queue.
00209 if (system_found)
00210 {
00211     if (accept)
00212     {
00213         log_(true, *packet);
00214         queue_->push(std::move(packet), priority);
00215     }
00216     else
00217     {
00218         log_(false, *packet);
00219     }
00220 }
00221 }
00222
00223
00224 /** Construct a connection.
00225 */
00226 * \param name The name of the connection, should be the device string for a
00227 *   serial connection or the IP address and port number for a UDP
00228 *   connection.
00229 * \param filter The packet filter to use for determining whether and with what
00230 *   priority to add a packet to the queue for transmission.
00231 * \param mirror Set to true if this is to be a mirror connection. A mirror
00232 *   connection is one that will receive all packets, regardless of
00233 *   destination address. The default is false.
00234 * \param pool The \ref AddressPool to use for keeping track of the addresses
00235 *   reachable by the connection. A default address pool will be used if
00236 *   none is given.
00237 * \param queue The \ref PacketQueue to use to hold packets awaiting
00238 *   transmission. A default packet queue will be used if none is given.
00239 * \throws std::invalid_argument if the given any of the \p filter, \p pool, or
00240 *   \p queue pointers are null.
00241 * \remarks If the given \ref AddressPool and \ref PacketQueue are threadsafe
00242 *   then the connection will also be threadsafe.
00243 */
00244 Connection::Connection(
00245     std::string name,
00246     std::shared_ptr<Filter> filter, bool mirror,
00247     std::unique_ptr<AddressPool>> pool,
00248     std::unique_ptr<PacketQueue> queue)
00249 : name_(std::move(name)),
00250     filter_(std::move(filter)), pool_(std::move(pool)),
00251     queue_(std::move(queue)), mirror_(mirror)
00252 {
00253     if (filter_ == nullptr)
00254     {
00255         throw std::invalid_argument("Given filter pointer is null.");
00256     }
00257
00258     if (pool_ == nullptr)
00259     {
00260         throw std::invalid_argument("Given pool pointer is null.");
00261     }
00262
00263     if (queue_ == nullptr)
00264     {
00265         throw std::invalid_argument("Given queue pointer is null.");
00266     }
00267 }
00268
00269
00270 /** Add a MAVLink address to the connection.
00271 */
00272 * This adds an address to the list of systems/components that can be reached
00273 * on this connection.
00274 *
00275 * \note Addresses will be removed after the timeout set in the \ref
00276 *   AddressPool given in the constructor. Re-adding the address (even
00277 *   before this time runs out) will reset the timeout.
00278 *
00279 * \param address The MAVLink address to add, or update the timeout for.
00280 */

```

```

00281 void Connection::add_address(MAVAddress address)
00282 {
00283     if (Logger::level() >= 1 && !pool_->contains(address))
00284     {
00285         Logger::log(1, "new component " + str(address) + " on " + name_);
00286     }
00287     pool_->add(std::move(address));
00288 }
00289
00290
00291
00292
00293 /** Get next packet to send.
00294 *
00295 * Blocks until a packet is ready to be sent or the \p timeout expires.
00296 * Returns nullptr in the later case.
00297 *
00298 * \param timeout How long to block waiting for a packet. Set to 0s for non
00299 * blocking.
00300 * \returns The next packet to send. Or nullptr if the call times out waiting
00301 * on a packet.
00302 */
00303 std::shared_ptr<const Packet> Connection::next_packet(
00304     const std::chrono::nanoseconds &timeout)
00305 {
00306     return queue_->pop(timeout);
00307 }
00308
00309
00310 /** Send a packet out on the connection.
00311 *
00312 * Packets are ran through the contained \ref Filter before being placed into
00313 * the \ref PacketQueue given in the constructor. Packets are read from the
00314 * queue (for sending) by using the \ref next_packet method.
00315 *
00316 * \note If the packet has a destination address that is not 0.0 (the
00317 * broadcast address) it will only be sent if that system is reachable on
00318 * this connection. It will still be sent even if the particular component
00319 * cannot be found.
00320 *
00321 * \note If this is a mirror connection then the destination address of the
00322 * packet is ignored.
00323 *
00324 * \param packet The packet to send.
00325 * \throws std::invalid_argument if the \p packet pointer is null.
00326 */
00327 void Connection::send(std::shared_ptr<const Packet> packet)
00328 {
00329     if (packet == nullptr)
00330     {
00331         throw std::invalid_argument("Given packet pointer is null.");
00332     }
00333
00334     // Drop packet if the source is reachable on this connection.
00335     if (pool_->contains(packet->source()))
00336     {
00337         // log_(false, *packet);
00338         return;
00339     }
00340
00341     auto dest = packet->dest();
00342
00343     // Broadcast to all.
00344     if (!dest.has_value() || dest.value() == MAVAddress(0, 0) || mirror_)
00345     {
00346         send_to_all_(std::move(packet));
00347     }
00348     // Broadcast to all components of given system.
00349     else if (dest->component() == 0)
00350     {
00351         send_to_system_(std::move(packet), dest->system());
00352     }
00353     // Send to particular system and component.
00354     else
00355     {
00356         send_to_address_(std::move(packet), dest.value());
00357     }
00358 }
00359
00360
00361 /** Print the connection name to the given output stream.

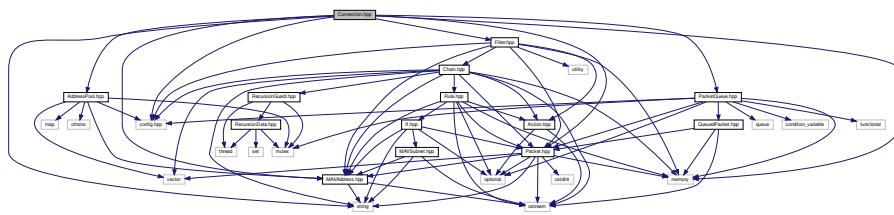
```

```
00362 *
00363 *      Some examples are:
00364 *          - '/dev/ttyUSB0'
00365 *          - '127.0.0.1:8000'
00366 *          - '127.0.0.1:14550'
00367 */
00368 std::ostream &operator<<(std::ostream &os, const Connection &connection)
00369 {
00370     os << connection.name_;
00371     return os;
00372 }
```

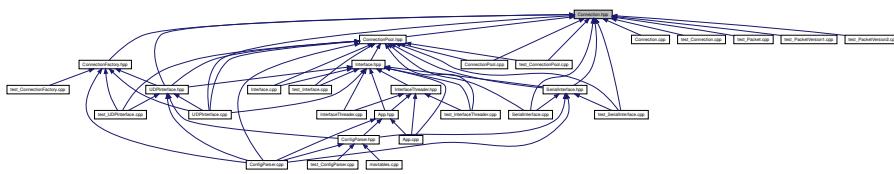
16.43 Connection.hpp File Reference

```
#include <memory>
#include <string>
#include "AddressPool.hpp"
#include "config.hpp"
#include "Filter.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "PacketQueue.hpp"
Include dependency graph for Connec
```

Include dependency graph for Connection.hpp:



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



Classes

- class Connection

Functions

- `std::ostream & operator<< (std::ostream &os, const Connection &connection)`

16.43.1 Function Documentation

16.43.1.1 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const Connection & connection )
```

Print the connection name to the given output stream.

Some examples are:

- /dev/ttyUSB0
- 127.0.0.1:8000
- 127.0.0.1:14550

Definition at line 368 of file [Connection.cpp](#).

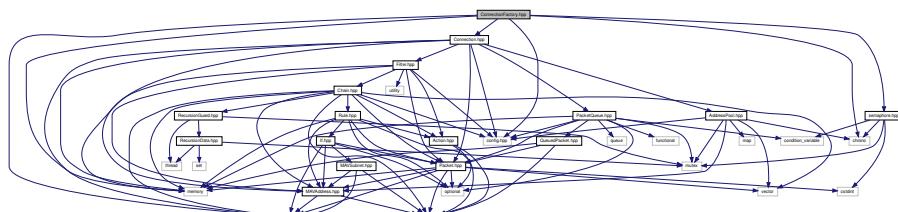
16.44 Connection.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef CONNECTION_HPP_
00019 #define CONNECTION_HPP_
00020
00021
00022 #include <memory>
00023 #include <string>
00024
00025 #include "AddressPool.hpp"
00026 #include "config.hpp"
00027 #include "Filter.hpp"
00028 #include "MAVAddress.hpp"
00029 #include "Packet.hpp"
00030 #include "PacketQueue.hpp"
00031
00032
00033 /** Represents a connection that packets can be sent over.
00034 *
00035 * The connection class does not actually send anything. It filters and sorts
00036 * packets in a queue for sending by an \ref Interface. It also maintains a
00037 * list of MAVLink addresses reachable on this connection.
```

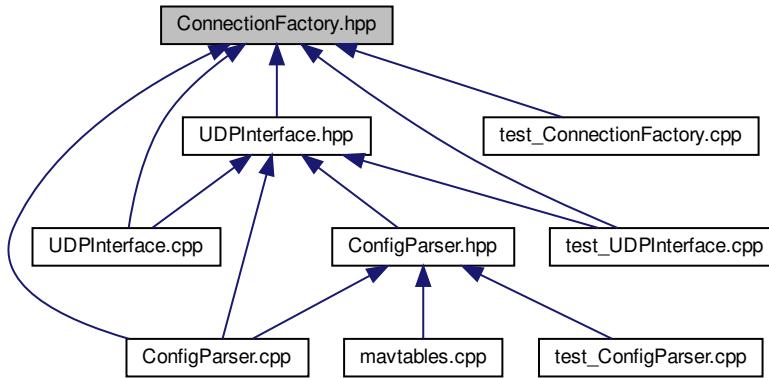
```
00038 */
00039 class Connection
00040 {
00041     public:
00042         Connection(
00043             std::string name,
00044             std::shared_ptr<Filter> filter, bool mirror = false,
00045             std::unique_ptr<AddressPool>> pool =
00046                 std::make_unique<AddressPool>>(),
00047             std::unique_ptr<PacketQueue> queue =
00048                 std::make_unique<PacketQueue>());
00049 // LCOV_EXCL_START
00050 TEST_VIRTUAL ~Connection() = default;
00051 // LCOV_EXCL_STOP
00052 TEST_VIRTUAL void add_address(MAVAddress address);
00053 TEST_VIRTUAL std::shared_ptr<const Packet> next_packet(
00054     const std::chrono::nanoseconds &timeout =
00055         std::chrono::nanoseconds(0));
00056 TEST_VIRTUAL void send(std::shared_ptr<const Packet> packet);
00057
00058     friend std::ostream &operator<<(
00059         std::ostream &os, const Connection &connection);
00060
00061 private:
00062     // Variables
00063     std::string name_;
00064     std::shared_ptr<Filter> filter_;
00065     std::unique_ptr<AddressPool>> pool_;
00066     std::unique_ptr<PacketQueue> queue_;
00067     bool mirror_;
00068     // Methods
00069     void log_(bool accept, const Packet &packet);
00070     void send_to_address_(
00071         std::shared_ptr<const Packet> packet, const MAVAddress &dest);
00072     void send_to_all_(std::shared_ptr<const Packet> packet);
00073     void send_to_system_(
00074         std::shared_ptr<const Packet> packet, unsigned int system);
00075 };
00076
00077
00078 std::ostream &operator<<(std::ostream &os, const Connection &connection);
00079
00080
00081 #endif // CONNECTION_HPP_
```

16.45 ConnectionFactory.hpp File Reference

```
#include <chrono>
#include <memory>
#include <string>
#include "config.hpp"
#include "Connection.hpp"
#include "semaphore.hpp"
Include dependency graph for ConnectionFactory.hpp:
```



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



Classes

- class [ConnectionFactory< C, AP, PQ >](#)

16.46 ConnectionFactory.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef CONNECTIONFACTORY_HPP_
00019 #define CONNECTIONFACTORY_HPP_
00020
00021
00022 #include <chrono>
00023 #include <memory>
00024 #include <string>
00025
00026 #include "config.hpp"
00027 #include "Connection.hpp"
00028 #include "semaphore.hpp"
00029
00030
00031 /** A factory for making related connections that use a common semaphore.
00032 */
00033 template <class C = Connection,
00034           class AP = AddressPool<>,
00035           class PQ = PacketQueue>
  
```

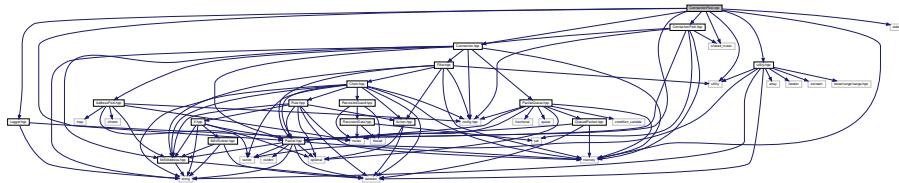
```
00036 class ConnectionFactory
00037 {
00038     public:
00039         ConnectionFactory(std::shared_ptr<Filter> filter, bool mirror = false);
00040         TEST_VIRTUAL ~ConnectionFactory() = default;
00041         TEST_VIRTUAL std::unique_ptr<C> get(std::string name = "unknown");
00042         TEST_VIRTUAL bool wait_for_packet(
00043             const std::chrono::nanoseconds &timeout);
00044
00045     private:
00046         std::shared_ptr<Filter> filter_;
00047         bool mirror_;
00048         semaphore semaphore_;
00049     };
00050
00051
00052 /** Construct a connection factory.
00053 */
00054 * \tparam C The Connection class (or derived class) to use.
00055 * \tparam AP The AddressPool class (or derived class) to use.
00056 * \tparam PQ The PacketQueue class (or derived class) to use, must accept a
00057 *     callback function in it's constructor.
00058 * \param filter The packet filter to use for determining whether and with what
00059 *     priority to add a packet to the queue for transmission. This will be
00060 *     given to each constructed \ref Connection. Cannot be nullptr.
00061 * \param mirror Set to true if all \ref Connection's made by this factory are
00062 *     to be mirror connections. A mirror connection is one that will receive
00063 *     all packets, regardless of destination address. The default is false.
00064 * \throws std::invalid_argument if the given \p filter pointer is null.
00065 */
00066 template <class C, class AP, class PQ>
00067 ConnectionFactory<C, AP, PQ>::ConnectionFactory(
00068     std::shared_ptr<Filter> filter, bool mirror)
00069     : filter_(std::move(filter)), mirror_(mirror)
00070 {
00071     if (filter_ == nullptr)
00072     {
00073         throw std::invalid_argument("Given filter pointer is null.");
00074     }
00075 }
00076
00077
00078 /** Construct and return a new connection.
00079 */
00080 * This connection will share a common semaphore with all other connections
00081 * made by this factory instance.
00082 *
00083 * \param name The name of the new connection.
00084 * \returns The new connection.
00085 */
00086 template <class C, class AP, class PQ>
00087 std::unique_ptr<C> ConnectionFactory<C, AP, PQ>::get(std::string
name)
00088 {
00089     return std::make_unique<C>(
00090         name, filter_, mirror_,
00091         std::make_unique<AP>(),
00092         std::make_unique<PQ>([this]()
00093     {
00094         semaphore_.notify();
00095     }));
00096 }
00097
00098
00099 /** Wait for a packet to be available on any connection made by this factory.
00100 */
00101 * \param timeout How long to block waiting for a packet. Set to 0s for non
00102 *     blocking.
00103 * \retval true There is a packet on at least one of the connections created by
00104 *     this factory instance.
00105 * \retval false The wait timed out, there is no packet available on any of the
00106 *     connections created by this factory instance.
00107 */
00108 template <class C, class AP, class PQ>
00109 bool ConnectionFactory<C, AP, PQ>::wait_for_packet(
00110     const std::chrono::nanoseconds &timeout)
00111 {
00112     return semaphore_.wait_for(timeout);
00113 }
00114
00115
```

```
00116 #endif // CONNECTIONFACTORY_HPP_
```

16.47 ConnectionPool.cpp File Reference

```
#include <memory>
#include <mutex>
#include <shared_mutex>
#include <stdexcept>
#include <utility>
#include "Connection.hpp"
#include "ConnectionPool.hpp"
#include "Logger.hpp"
#include "Packet.hpp"
#include "utility.hpp"

Include dependency graph for ConnectionPool.cpp:
```



16.48 ConnectionPool.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <memory>
00019 #include <mutex>
00020 #include <shared_mutex>
00021 #include <stdexcept>
00022 #include <utility>
00023
00024 #include "Connection.hpp"
00025 #include "ConnectionPool.hpp"
00026 #include "Logger.hpp"
00027 #include "Packet.hpp"
00028 #include "utility.hpp"
00029
00030
00031 /** Add a connection to the pool.
00032 *
00033 * \param connection The connection to add to the pool.
00034 */
```

```

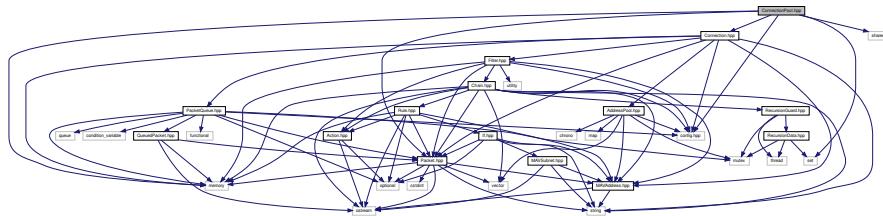
00035 void ConnectionPool::add(std::weak_ptr<Connection> connection)
00036 {
00037     std::lock_guard<std::shared_mutex> lock(mutex_);
00038     connections_.insert(std::move(connection));
00039 }
00040
00041
00042 /** Remove a connection from the pool.
00043 *
00044 * \param connection The connection to remove from the pool.
00045 */
00046 void ConnectionPool::remove(const std::weak_ptr<Connection> &connection)
00047 {
00048     std::lock_guard<std::shared_mutex> lock(mutex_);
00049     connections_.erase(connection);
00050 }
00051
00052
00053 /** Send a packet to every connection.
00054 *
00055 * \note Each connection may decide to ignore the packet based on it's filter
00056 *       rules.
00057 *
00058 * \param packet The packet to send to every connection, must not be nullptr.
00059 * \throws std::invalid_argument if the \p packet pointer is null.
00060 */
00061 void ConnectionPool::send(std::unique_ptr<const Packet> packet)
00062 {
00063     if (packet == nullptr)
00064     {
00065         throw std::invalid_argument("Given packet pointer is null.");
00066     }
00067
00068     if (Logger::level() >= 2)
00069     {
00070         std::stringstream ss;
00071         ss << "received " << str(*packet) << " source ";
00072         auto connection = packet->connection();
00073
00074         if (connection == nullptr)
00075         {
00076             ss << "unknown";
00077         }
00078         else
00079         {
00080             ss << *connection;
00081         }
00082
00083         Logger::log(2, ss.str());
00084     }
00085
00086     std::shared_lock<std::shared_mutex> lock(mutex_);
00087     std::shared_ptr<const Packet> shared = std::move(packet);
00088
00089     for (auto it = connections_.begin(); it != connections_.end();)
00090     {
00091         // Send packet on connection.
00092         if (auto connection = it->lock())
00093         {
00094             connection->send(shared);
00095             ++it;
00096         }
00097         // Remove connection.
00098         else
00099         {
00100             it = connections_.erase(it);
00101         }
00102     }
00103 }

```

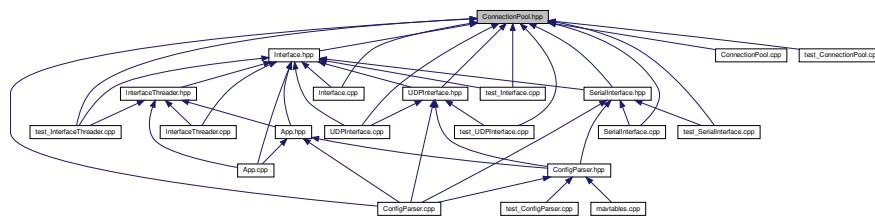
16.49 ConnectionPool.hpp File Reference

```
#include <memory>
#include <set>
```

```
#include <shared_mutex>
#include "config.hpp"
#include "Connection.hpp"
#include "Packet.hpp"
Include dependency graph for ConnectionPool.hpp:
```



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



Classes

- class [ConnectionPool](#)

16.50 ConnectionPool.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef CONNECTIONPOOL_HPP_
00019 #define CONNECTIONPOOL_HPP_
00020
00021
00022 #include <memory>
00023 #include <set>
```

```

00024 #include <shared_mutex>
00025
00026 #include "config.hpp"
00027 #include "Connection.hpp"
00028 #include "Packet.hpp"
00029
00030
00031 /** A pool of \ref Connection's to send packets out on.
00032 *
00033 * A connection pool stores a reference to all connections that packets can be
00034 * sent out over.
00035 */
00036 class ConnectionPool
00037 {
00038     public:
00039         // LCOV_EXCL_START
00040         TEST_VIRTUAL ~ConnectionPool() = default;
00041         // LCOV_EXCL_STOP
00042         TEST_VIRTUAL void add(std::weak_ptr<Connection> connection);
00043         TEST_VIRTUAL void remove(const std::weak_ptr<Connection> &connection);
00044         TEST_VIRTUAL void send(std::unique_ptr<const Packet> packet);
00045
00046     private:
00047         std::set<std::weak_ptr<Connection>,
00048                 std::owner_less<std::weak_ptr<Connection>>> connections_;
00049         std::shared_mutex mutex_;
00050 };
00051
00052
00053 #endif // CONNECTIONPOOL_HPP_

```

16.51 CONTRIBUTING.md File Reference

16.52 CONTRIBUTING.md

```

00001 Contributing {#contributing}
00002 =====
00003
00004 When contributing to this repository, please first discuss the change you wish
00005 to make via issue, email, or any other method with the owners of this repository
00006 before making a change.
00007
00008
00009 ## Pull Request Process
00010
00011 1. Create a pull request early in development processes with a checkbox list of
00012     features/changes you plan to make along with a description of what you intend
00013     to do. Prepend (WIP:) for "work in progress" to the title of this pull request
00014     until you are ready for review and merging.
00015 2. Update README.rst and the Sphinx documentation with details of any changes to
00016     the interface, this includes new environment variables, exposed ports, and
00017     useful file locations.
00018 3. Ensure test coverage did not drop below 90% and that all unit tests pass.
00019 4. Ensure the package builds, installs, and runs using 'pip install .' in
00020     a clean 'virtualenv'. List any new external dependencies in the README.rst
00021     file.
00022 5. Increase the version numbers in any examples files and the README.rst to the
00023     new version that this Pull Request would represent. The versioning scheme we
00024     use is [SemVer](http://semver.org/).
00025 6. You may merge the Pull Request in once you have the sign-off of a core
00026     developer, or if you do not have permission to do that, you may request the
00027     reviewer merge it for you.
00028
00029
00030 ## Git Etiquette
00031
00032 * Do not work on the main repository unless you are merging in changes,
00033     preparing for a version bump, or other repository management.
00034     * Create a fork, make a new branch and work on your changes there. Merge
00035         these changes in using the _Pull Request Process_ above.
00036 * Do not commit IDE/Editor specific files or any generated files. You should
00037     add these files to the '.gitignore' file to ensure you do not accidentally

```

```

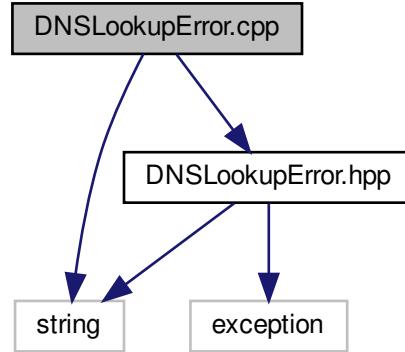
00038     commit them.
00039 * Do not commit application specific configuration files. If you wish to submit
00040     an example place it in the examples directory.
00041 * Do not commit assets such as pictures or videos. The exception is small
00042     (under 300 kB) images used in the documentation.
00043 * Do not commit other project's documentation such as PDF files.
00044     * Link to that projects documentation instead.
00045 * Do not commit another project's source code.
00046     * Use a [Git submodule](https://git-scm.com/docs/git-submodule) instead.
00047
00048
00049 ## Programming Style
00050
00051 * All C++17 features are allowed.
00052 * Simply running the 'style' rule of the Makefile will fix most style issues and
00053     should be used to ensure consistent style. This target will be ran before
00054     a merge to check for style consistency but does not check naming conventions.
00055     It does not always fix line length problems though. This is why the 'style'
00056     rule also prints lines exceeding 80 characters that must be fixed manually.
00057 * All type names such as classes, structs, type aliases, enums, and type
00058     template parameters should be 'CamelCase'.
00059 * Functions, variables, and namespaces should be 'snake_case'.
00060 * No name may begin with a single '_' (underscore) or contain '__' (double
00061     underscore).
00062 * Private data members and functions should end in an underscore.
00063     have an 'autodoc' entry in the Sphinx documentation.
00064 * Each class should have it's own source and header file.
00065 * When a file contains a class the file should be named for the class. All
00066     other file names should be lower case and should only include letters,
00067     numbers, and underscores.
00068 * C++ source code files should use the '.cpp' extension and '.hpp' for header
00069     files. '.c' and '.h' should only be used for C files.
00070 * All header files must have include guards and the name should be the file name
00071     (all uppercase) with '_HPP_' appended.
00072 * All public functions and classes should contain [Doxygen style
00073     comments](https://www.stack.nl/~dimitri/doxygen/manual/docblocks.html) and the
00074     recommended block format is
00075
00076     /**
00077     *
00078     */
00079
00080 And the '\\' (backslash) is recommended over the '@' (at symbol).

```

16.53 DNSLookupError.cpp File Reference

```
#include <string>
#include "DNSLookupError.hpp"
```

Include dependency graph for DNSLookupError.cpp:



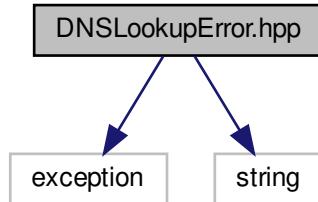
16.54 DNSLookupError.cpp

```

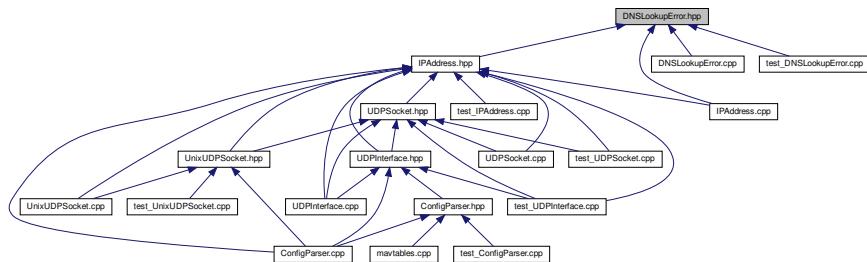
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <string>
00019
00020 #include "DNSLookupError.hpp"
00021
00022
00023 /** Construct a \ref DNSLookupError given the unresolvable URL.
00024 *
00025 * \param url The URL that could not be resolved.
00026 */
00027 DNSLookupError::DNSLookupError(std::string url)
00028 {
00029     message_ = "DNSLookupError: Could not find an IP address for \""
00030             + url + "\"";
00031 }
00032
00033
00034 /** Return error message string.
00035 *
00036 * \returns Error string containing unresolvable hostname.
00037 */
00038 const char *DNSLookupError::what() const noexcept
00039 {
00040     return message_.c_str();
00041 }
  
```

16.55 DNSLookupError.hpp File Reference

```
#include <exception>
#include <string>
Include dependency graph for DNSLookupError.hpp:
```



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



Classes

- class `DNSLookupError`

16.56 DNSLookupError.hpp

```
00001 // MAVLink router and firewall.  
00002 // Copyright (C) 2017 Michael R. Shannon <mrshannon.aerospace@gmail.com>  
00003 //  
00004 // This program is free software; you can redistribute it and/or modify  
00005 // it under the terms of the GNU General Public License as published by  
00006 // the Free Software Foundation; either version 2 of the License, or  
00007 // (at your option) any later version.  
00008 //  
00009 // This program is distributed in the hope that it will be useful,  
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of  
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
00012 // GNU General Public License for more details.  
00013 //  
00014 // You should have received a copy of the GNU General Public License  
00015 // along with this program; if not, write to the Free Software Foundation,  
00016 // Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA  
00017 //  
00018 // The original code was written by Michael R. Shannon  
00019 // and is available at https://github.com/mrshannon/aerospace
```

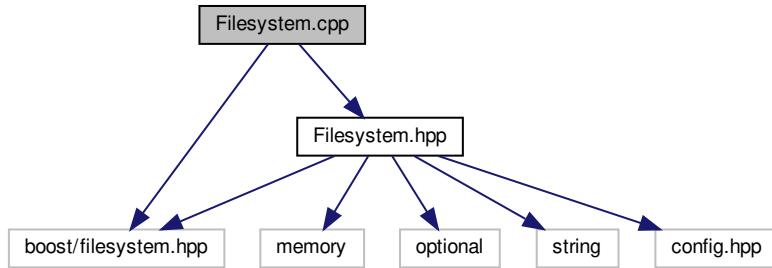
```

00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef DNSLOOKUPERROR_HPP_
00019 #define DNSLOOKUPERROR_HPP_
00020
00021
00022 #include <exception>
00023 #include <string>
00024
00025
00026 /** Exception type emitted when a DNS lookup fails.
00027 */
00028 class DNSLookupError : public std::exception
00029 {
00030     public:
00031         DNSLookupError(std::string url);
00032         const char *what() const noexcept;
00033
00034     private:
00035         std::string message_;
00036 };
00037
00038
00039 #endif // DNSLOOKUPERROR_HPP_

```

16.57 Filesystem.cpp File Reference

```
#include <boost/filesystem.hpp>
#include "Filesystem.hpp"
Include dependency graph for Filesystem.cpp:
```



16.58 Filesystem.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //

```

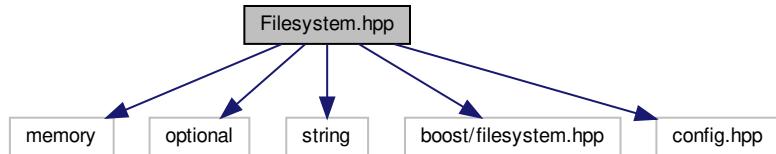
```

00009 // This program is distributed in the hope that it will be useful,
0010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
0011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
0012 // GNU General Public License for more details.
0013 //
0014 // You should have received a copy of the GNU General Public License
0015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
0016
0017
0018 #include <boost/filesystem.hpp>
0019
0020 #include "Filesystem.hpp"
0021
0022
0023 // GCC generates a seemingly uncallable destructor for pure virtual classes.
0024 // Therefore, it must be excluded from test coverage.
0025 // LCOV_EXCL_START
0026 Filesystem::~Filesystem()
0027 {
0028 }
0029 // LCOV_EXCL_STOP
0030
0031
0032 /** Returns true if the given path exists.
0033 *
0034 * \param p The path to check for existence.
0035 */
0036 bool Filesystem::exists(const Filesystem::path &p) const
0037 {
0038     return boost::filesystem::exists(p);
0039 }
```

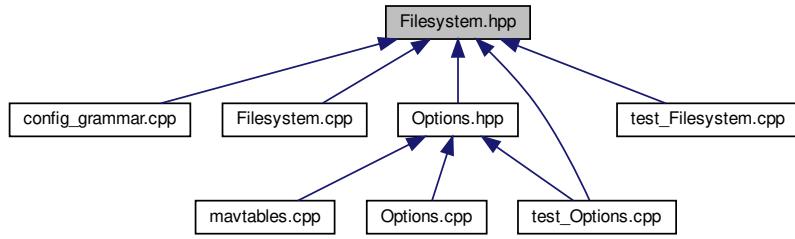
16.59 Filesystem.hpp File Reference

```
#include <memory>
#include <optional>
#include <string>
#include <boost/filesystem.hpp>
#include "config.hpp"

Include dependency graph for Filesystem.hpp:
```



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



Classes

- class [Filesystem](#)

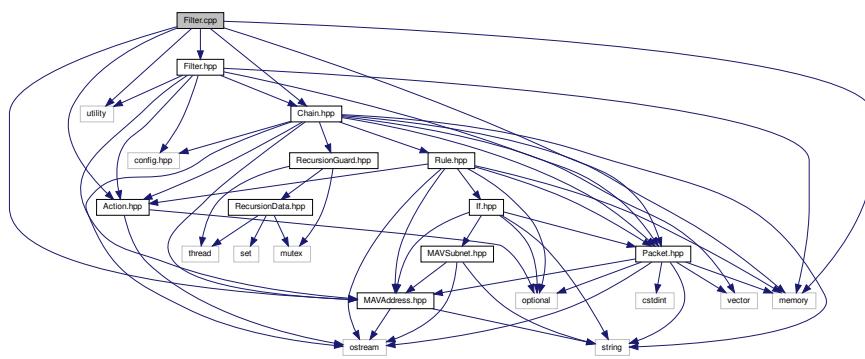
16.60 Filesystem.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef FILESYSTEM_HPP_
00019 #define FILESYSTEM_HPP_
00020
00021
00022 #include <memory>
00023 #include <optional>
00024 #include <string>
00025
00026 #include <boost/filesystem.hpp>
00027
00028 #include "config.hpp"
00029
00030
00031 /** A class containing filesystem operations.
00032 *
00033 * \note This class exists for testing purposes and to provide a level of
00034 *      indirection so the underlying filesystem library can be changed to
00035 *      std::filesystem once it is moved out of experimental.
00036 */
00037 class Filesystem
00038 {
00039     public:
00040         using path = boost::filesystem::path;
00041         TEST_VIRTUAL ~Filesystem();
00042         TEST_VIRTUAL bool exists(const path &p) const;
00043     };
00044
00045
00046 #endif // FILESYSTEM_HPP_
  
```

16.61 Filter.cpp File Reference

```
#include <memory>
#include <utility>
#include "Action.hpp"
#include "Chain.hpp"
#include "Filter.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
Include dependency graph for Filter.cpp:
```



16.62 Filter.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <memory>
00019 #include <utility>
00020
00021 #include "Action.hpp"
00022 #include "Chain.hpp"
00023 #include "Filter.hpp"
00024 #include "MAVAddress.hpp"
00025 #include "Packet.hpp"
00026
00027
00028 /** Construct a new packet filter.
00029 *
00030 * \param default_chain The \ref Chain that all filtering begins with.
00031 * \param accept_by_default Whether to accept (true) or reject (false) packets
00032 * that don't match any rules in the default chain or any chains called by
00033 * the default chain. The default value is false and thus to reject
00034 * unmatched packets.
```

```

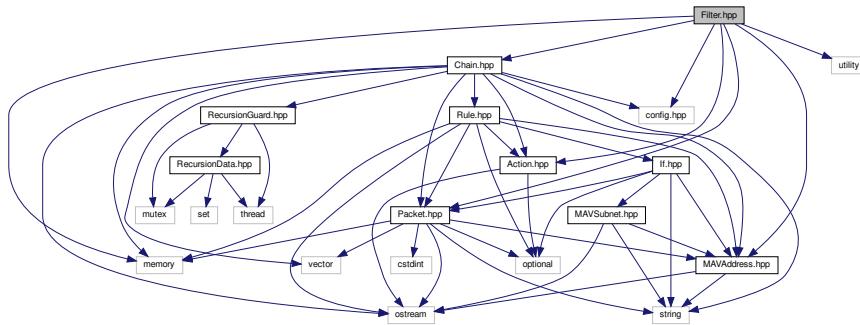
00035  */
00036 Filter::Filter(Chain default_chain, bool accept_by_default)
00037     : default_chain_(std::move(default_chain)),
00038     accept_by_default_(accept_by_default)
00039 {
00040 }
00041
00042
00043 /** Determine whether to accept or reject a packet/address combination.
00044 *
00045 * \param packet The packet to determine whether to allow or not.
00046 * \param address The address the \p packet will be sent out on if the
00047 *     action allows it.
00048 * \returns A pair (tuple) with the first value being whether to accept the
00049 *     packet or not and the second being the priority to use when sending the
00050 *     packet. The second value is only defined if the first value is true
00051 *     (accept).
00052 */
00053 std::pair<bool, int> Filter::will_accept(
00054     const Packet &packet, const MAVAddress &address)
00055 {
00056     Action result = default_chain_.action(packet, address);
00057
00058     switch (result.action())
00059     {
00060         case Action::ACCEPT:
00061             return {true, result.priority()};
00062         case Action::REJECT:
00063             return {false, 0};
00064         case Action::CONTINUE:
00065             break;
00066
00067         case Action::DEFAULT:
00068             break;
00069     }
00070
00071     return {accept_by_default_, 0};
00072 }
00073
00074
00075 /** Equality comparison.
00076 *
00077 * The default chain and default action are compared.
00078 *
00079 * \relates Filter
00080 * \param lhs The left hand side packet filter.
00081 * \param rhs The right hand side packet filter.
00082 * \retval true if \p lhs is the same as rhs.
00083 * \retval false if \p lhs is not the same as rhs.
00084 */
00085 bool operator==(const Filter &lhs, const Filter &rhs)
00086 {
00087     return (lhs.default_chain_ == rhs.default_chain_) &&
00088         (lhs.accept_by_default_ == rhs.accept_by_default_);
00089 }
00090
00091
00092 /** Inequality comparison.
00093 *
00094 * The default chain and default action are compared.
00095 *
00096 * \relates Filter
00097 * \param lhs The left hand side packet filter.
00098 * \param rhs The right hand side packet filter.
00099 * \retval true if \p lhs is not the same as rhs.
00100 * \retval false if \p lhs is the same as rhs.
00101 */
00102 bool operator!=(const Filter &lhs, const Filter &rhs)
00103 {
00104     return (lhs.default_chain_ != rhs.default_chain_) ||
00105         (lhs.accept_by_default_ != rhs.accept_by_default_);
00106 }

```

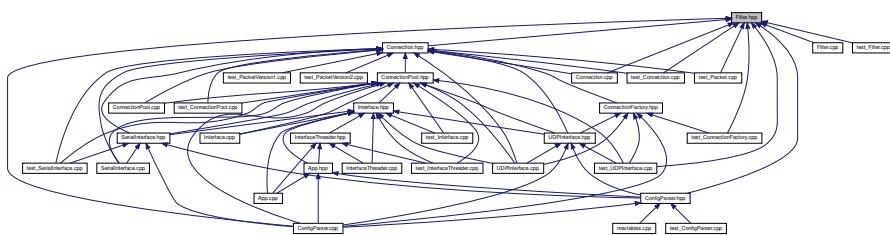
16.63 Filter.hpp File Reference

```
#include <memory>
```

```
#include <utility>
#include "Action.hpp"
#include "Chain.hpp"
#include "config.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
Include dependency graph for Filter.hpp:
```



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



Classes

- class Filter

Functions

- bool `operator==` (const `Filter` &lhs, const `Filter` &rhs)
 - bool `operator!=` (const `Filter` &lhs, const `Filter` &rhs)

16.63.1 Function Documentation

16.63.1.1 operator!=()

```
bool operator!= (
    const Filter & lhs,
    const Filter & rhs )
```

16.63.1.2 operator==()

```
bool operator== (
    const Filter & lhs,
    const Filter & rhs )
```

16.64 Filter.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef FILTER_HPP_
00019 #define FILTER_HPP_
00020
00021
00022 #include <memory>
00023 #include <utility>
00024
00025 #include "Action.hpp"
00026 #include "Chain.hpp"
00027 #include "config.hpp"
00028 #include "MAVAddress.hpp"
00029 #include "Packet.hpp"
00030
00031
00032 /** The filter used to determine whether to accept or reject a packet.
00033 *
00034 * \sa Chain
00035 * \sa Rule
00036 * \sa If
00037 */
00038 class Filter
00039 {
00040     public:
00041         /** Copy constructor.
00042         *
00043         * \param other Filter to copy from.
00044         */
00045         Filter(const Filter &other) = default;
00046         /** Move constructor.
00047         *
00048         * \param other Filter to move from.
00049         */
00050         Filter(Filter &&other) = default;
```

```

00051     Filter(Chain default_chain, bool accept_by_default = false);
00052     // LCOV_EXCL_START
00053     TEST_VIRTUAL ~Filter() = default;
00054     // LCOV_EXCL_STOP
00055     TEST_VIRTUAL std::pair<bool, int> will_accept(
00056         const Packet &packet, const MAVAddress &address);
00057     /** Assignment operator.
00058     *
00059     * \param other Filter to copy from.
00060     */
00061     Filter &operator=(const Filter &other) = default;
00062     /** Assignment operator (by move semantics).
00063     *
00064     * \param other Filter to move from.
00065     */
00066     Filter &operator=(Filter &&other) = default;
00067
00068     friend bool operator==(const Filter &lhs, const Filter &rhs);
00069     friend bool operator!=(const Filter &lhs, const Filter &rhs);
00070
00071 private:
00072     Chain default_chain_;
00073     bool accept_by_default_;
00074 };
00075
00076
00077 bool operator==(const Filter &lhs, const Filter &rhs);
00078 bool operator!=(const Filter &lhs, const Filter &rhs);
00079
00080
00081 #endif // FILTER_HPP_

```

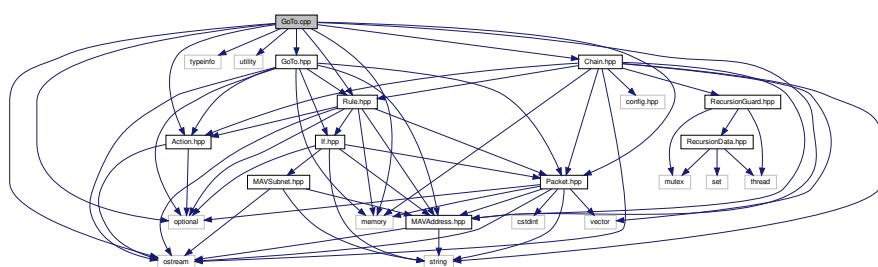
16.65 GoTo.cpp File Reference

```

#include <memory>
#include <optional>
#include <iostream>
#include <typeinfo>
#include <utility>
#include "Action.hpp"
#include "Chain.hpp"
#include "GoTo.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "Rule.hpp"
#include "RuleGuard.hpp"
#include "RuleGuardData.hpp"
#include "RecursionGuard.hpp"
#include "RecursionGuardData.hpp"
#include "Mutex.hpp"
#include "Set.hpp"
#include "Thread.hpp"

```

Include dependency graph for GoTo.cpp:



16.66 GoTo.cpp

```

00001 // MAVLink router and firewall.

```

```
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <memory>
00019 #include <optional>
00020 #include <ostream>
00021 #include <typeinfo>
00022 #include <utility>
00023
00024 #include "Action.hpp"
00025 #include "Chain.hpp"
00026 #include "GoTo.hpp"
00027 #include "MAVAddress.hpp"
00028 #include "Packet.hpp"
00029 #include "Rule.hpp"
00030
00031
00032 /** Construct a goto rule given a chain to delegate to, without a priority.
00033 *
00034 * A goto rule is used to delegate the decision on whether to accept or reject
00035 * a packet/address combination to another filter \ref Chain. If this called
00036 * chain does not make a decision then the default action should be taken.
00037 *
00038 * \param chain The chain to delegate decisions of whether to accept or reject
00039 * a packet/address combination to. nullptr is not valid.
00040 * \param condition The condition used to determine the rule matches a
00041 * particular packet/address combination given to the \ref action method.
00042 * The default is {} which indicates the rule matches any packet/address
00043 * combination.
00044 * \throws std::invalid_argument if the given pointer is null.
00045 * \sa action
00046 */
00047 GoTo::GoTo(
00048     std::shared_ptr<Chain> chain, std::optional<If> condition)
00049     : Rule(std::move(condition)), chain_(std::move(chain))
00050 {
00051     if (chain_ == nullptr)
00052     {
00053         throw std::invalid_argument("Given chain pointer is null.");
00054     }
00055 }
00056
00057
00058 /** Construct a goto rule given a chain to delegate to, with a priority.
00059 *
00060 * A goto rule is used to delegate the decision on whether to accept or reject
00061 * a packet/address combination to another filter \ref Chain. If this called
00062 * chain does not make a decision then the default action should be taken.
00063 *
00064 * \param chain The chain to delegate decisions of whether to accept or reject
00065 * a packet/address combination to. nullptr is not valid.
00066 * \param priority The priority to accept packets with. A higher number is
00067 * more important and will be routed first.
00068 * \param condition The condition used to determine the rule matches a
00069 * particular packet/address combination given to the \ref action method.
00070 * The default is {} which indicates the rule matches any packet/address
00071 * combination.
00072 * \throws std::invalid_argument if the given pointer is null.
00073 * \sa action
00074 */
00075 GoTo::GoTo(
00076     std::shared_ptr<Chain> chain, int priority,
00077     std::optional<If> condition)
00078     : Rule(std::move(condition)), chain_(std::move(chain)), priority_(priority)
00079 {
00080     if (chain_ == nullptr)
00081     {
00082         throw std::invalid_argument("Given chain pointer is null.");
```

```

00083     }
00084 }
00085
00086
00087 /** \copydoc Rule::print_(std::ostream &os) const
00088 *
00089 * Prints "goto <Chain Name> <If Statement>" or "goto <Chain Name> with
00090 * priority <If Statement> with priority <priority>" if the priority is
00091 * given.
00092 */
00093 std::ostream &GoTo::print_(std::ostream &os) const
00094 {
00095     os << "goto " << chain_>name();
00096
00097     if (priority_)
00098     {
00099         os << " with priority " << priority_.value();
00100     }
00101
00102     if (condition_)
00103     {
00104         os << " " << condition_.value();
00105     }
00106
00107     return os;
00108 }
00109
00110
00111 /** \copydoc Rule::action(const Packet&,const MAVAddress&) const
00112 *
00113 * The GoTo class delegates the action choice to the contained \ref Chain. %If
00114 * the \ref Chain decides on the continue action this method will return the
00115 * default instead since final decision for a \ref GoTo should be with the
00116 * contained \ref Chain or with the default action. In other words, once a
00117 * GoTo rule matches, no further rule in the chain should ever be ran,
00118 * regardless of the contained chain.
00119 */
00120 Action GoTo::action(
00121     const Packet &packet, const MAVAddress &address) const
00122 {
00123     if (!condition_ || condition_->check(packet, address))
00124     {
00125         auto result = chain_->action(packet, address);
00126
00127         if (priority_)
00128         {
00129             // Only has an effect if the action is accept and does not already
00130             // have a priority.
00131             result.priority(priority_.value());
00132         }
00133
00134         // Rewrite continue actions into default actions.
00135         if (result.action() == Action::CONTINUE)
00136         {
00137             return Action::make_default();
00138         }
00139
00140         return result;
00141     }
00142
00143     return Action::make_continue();
00144 }
00145
00146
00147 std::unique_ptr<Rule> GoTo::clone() const
00148 {
00149     if (priority_)
00150     {
00151         return std::make_unique<GoTo>(chain_, priority_.value(), condition_);
00152     }
00153
00154     return std::make_unique<GoTo>(chain_, condition_);
00155 }
00156
00157
00158 /** \copydoc Rule::operator==(const Rule &) const
00159 *
00160 * Compares the chain and priority (if set) associated with the rule as well.
00161 */
00162 bool GoTo::operator==(const Rule &other) const
00163 {

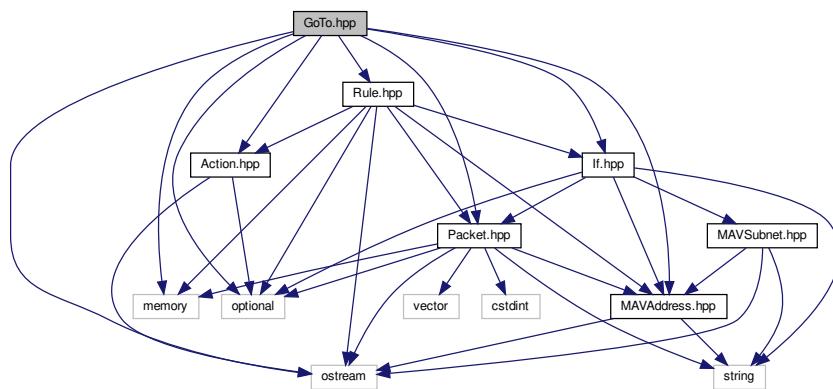
```

```

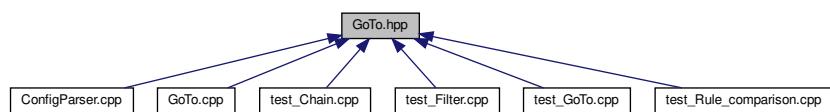
00164     return typeid(*this) == typeid(other) &&
00165         chain_ == static_cast<const GoTo &>(other).chain_ &&
00166         priority_ == static_cast<const GoTo &>(other).priority_ &&
00167         condition_ == static_cast<const GoTo &>(other).condition_;
00168 }
00169
00170
00171 /** \copydoc Rule::operator!=(const Rule &) const
00172 *
00173 * Compares the chain and priority (if set) associated with the rule as well.
00174 */
00175 bool GoTo::operator!=(const Rule &other) const
00176 {
00177     return typeid(*this) != typeid(other) ||
00178         chain_ != static_cast<const GoTo &>(other).chain_ ||
00179         priority_ != static_cast<const GoTo &>(other).priority_ ||
00180         condition_ != static_cast<const GoTo &>(other).condition_;
00181 }
```

16.67 GoTo.hpp File Reference

```
#include <memory>
#include <optional>
#include <iostream>
#include "Action.hpp"
#include "If.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "Rule.hpp"
Include dependency graph for GoTo.hpp:
```



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



Classes

- class GoTo

16.68 GoTo.hpp

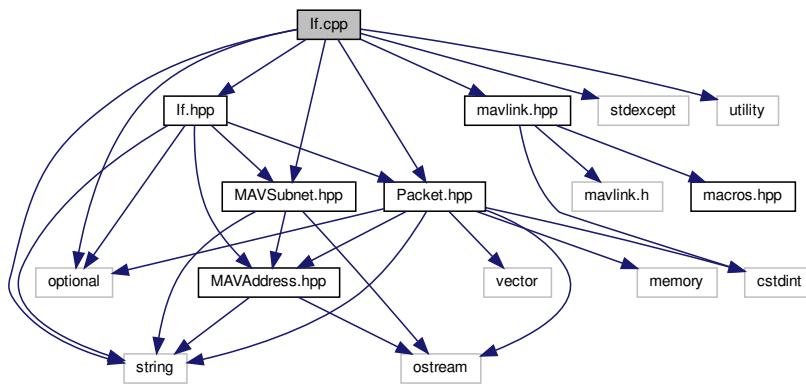
```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef GOTO_HPP_
00019 #define GOTO_HPP_
00020
00021
00022 #include <memory>
00023 #include <optional>
00024 #include <iostream>
00025
00026 #include "Action.hpp"
00027 #include "If.hpp"
00028 #include "MAVAddress.hpp"
00029 #include "Packet.hpp"
00030 #include "Rule.hpp"
00031
00032
00033 // Forward declaration of the chain class.
00034 class Chain;
00035
00036
00037 /** Delegate decision on a packet to another \ref Chain.
00038 *
00039 * Rule to delegate the decision on what to do with a packet to a filter \ref
00040 * Chain. In particular, final decision is given to this \ref Chain. %If the
00041 * \ref Chain cannot decide what to do with the \ref Packet the global default
00042 * action should be taken.
00043 */
00044 class GoTo : public Rule
00045 {
00046     public:
00047         GoTo(std::shared_ptr<Chain> chain,
00048               std::optional<If> condition = {});
00049         GoTo(std::shared_ptr<Chain> chain, int priority,
00050               std::optional<If> condition = {});
00051         virtual Action action(
00052             const Packet &packet, const MAVAddress &address) const;
00053         virtual std::unique_ptr<Rule> clone() const;
00054         virtual bool operator==(const Rule &other) const;
00055         virtual bool operator!=(const Rule &other) const;
00056
00057     protected:
00058         virtual std::ostream &print_(std::ostream &os) const;
00059
00060     private:
00061         std::shared_ptr<Chain> chain_;
00062         std::optional<int> priority_;
00063     };
00064
00065
00066 #endif // GOTO_HPP_

```

16.69 If.cpp File Reference

```
#include <optional>
#include <stdexcept>
#include <string>
#include <utility>
#include "If.hpp"
#include "mavlink.hpp"
#include "MAVSubnet.hpp"
#include "Packet.hpp"
Include dependency graph for If.cpp:
```



Functions

- std::ostream & **operator<<** (std::ostream &os, const **If** &if_)

16.69.1 Function Documentation

16.69.1.1 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const If & if_ )
```

Print the **If** statement to the given output stream.

Some examples are:

- if PING from 1.0/8 to 255.0
- if HEARTBEAT from 255.0/8
- if SET_MODE to 255.0
- if from 255.0/8

Definition at line 224 of file [If.cpp](#).

References [mavlink::name\(\)](#).

Here is the call graph for this function:



16.70 If.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <optional>
00019 #include <stdexcept>
00020 #include <string>
00021 #include <utility>
00022
00023 #include "If.hpp"
00024 #include "mavlink.hpp"
00025 #include "MAVSubnet.hpp"
00026 #include "Packet.hpp"
00027
00028
00029 /** Construct an If statement.
00030 *
00031 * The default is to allow any type of packet from any address to any address.
00032 *
00033 * The \ref type, \ref to, and \ref from methods can be used to apply
00034 * conditions after construction. Some examples are:
00035 * - '%If().type("PING").from("1.0/8").to("255.0");'
00036 * - '%If().type("HEARTBEAT").from("255.0/8");'
00037 * - '%If().type("SET_MODE").to("255.0/8");'
00038 * - '%If().from("255.0/8");'
00039 */

```

```
00040 * \param id The packet ID to match. %If {} (std::nullopt) then any packet ID
00041 * will match. The default is {}.
00042 * \param source The subnet a source address must be in to match. %If {}
00043 * (std::nullopt) then any source address will match. The default is {}.
00044 * \param dest The subnet a destination address must be in to match. %If {}
00045 * (std::nullopt) then any destination address will match. The default is {}
00046 * {}.
00047 * \throws std::invalid_argument if the given \p id is not valid.
00048 */
00049 If::If(
00050     std::optional<unsigned long> id,
00051     std::optional<MAVSubnet> source,
00052     std::optional<MAVSubnet> dest)
00053     : source_(std::move(source)), dest_(std::move(dest))
00054 {
00055     if (id)
00056     {
00057         type(id.value());
00058     }
00059 }
00060
00061
00062 /** Set the packet type to match, by ID.
00063 *
00064 * \param id The packet ID to match.
00065 * \returns A reference to itself.
00066 * \throws std::invalid_argument if the given \p id is not valid.
00067 * \sa type(const std::string &name)
00068 */
00069 If &If::type(unsigned long id)
00070 {
00071     // Check packet ID, throws error if invalid.
00072     mavlink::name(id);
00073     id_ = id;
00074     return *this;
00075 }
00076
00077
00078 /** Set the packet type to match, by name.
00079 *
00080 * \param name The packet name to match.
00081 * \returns A reference to itself.
00082 * \throws std::invalid_argument if the given message \p name is not valid.
00083 * \sa type(unsigned long id)
00084 */
00085 If &If::type(const std::string &name)
00086 {
00087     id_ = mavlink::id(name);
00088     return *this;
00089 }
00090
00091
00092 /** Set subnet for source address matching using \ref MAVSubnet.
00093 *
00094 * \param subnet The subnet used for source address matching.
00095 * \returns A reference to itself.
00096 * \sa from(const std::string &subnet)
00097 */
00098 If &If::from(MAVSubnet subnet)
00099 {
00100     source_ = std::move(subnet);
00101     return *this;
00102 }
00103
00104
00105 /** Set subnet for source address matching by string.
00106 *
00107 * See \ref MAVSubnet::MAVSubnet(std::string address) for the acceptable
00108 * formats and possible errors.
00109 *
00110 * \param subnet The subnet used for source address matching.
00111 * \returns A reference to itself.
00112 * \sa from(MAVSubnet subnet)
00113 */
00114 If &If::from(const std::string &subnet)
00115 {
00116     source_ = MAVSubnet(subnet);
00117     return *this;
00118 }
00119
00120
```

```

00121 /** Set subnet for destination address matching using \ref MAVSubnet.
00122 *
00123 * \param subnet The subnet used for destination address matching.
00124 * \returns A reference to itself.
00125 * \sa to(const std::string &subnet)
00126 */
00127 If &If::to(MAVSubnet subnet)
00128 {
00129     dest_ = std::move(subnet);
00130     return *this;
00131 }
00132
00133
00134 /** Set subnet for destination address matching by string.
00135 *
00136 * See \ref MAVSubnet::MAVSubnet(std::string address) for the acceptable
00137 * formats and possible errors.
00138 *
00139 * \param subnet The subnet used for destination address matching.
00140 * \returns A reference to itself.
00141 * \sa to(MAVSubnet subnet)
00142 */
00143 If &If::to(const std::string &subnet)
00144 {
00145     dest_ = MAVSubnet(subnet);
00146     return *this;
00147 }
00148
00149
00150 /** Check whether a \ref Packet and \ref MAVAddress combination matches.
00151 *
00152 * \param packet The packet to check for a match.
00153 * \param address The address the packet is to be sent to.
00154 * \retval true %If the packet matches the type, source subnet (by \ref
00155 * MAVSubnet::contains), and destination subnet (by \ref
00156 * MAVSubnet::contains) of the if statement.
00157 * \retval false %If any of the packet type, source subnet, or destination
00158 * subnet does not match.
00159 */
00160 bool If::check(const Packet &packet, const MAVAddress &address) const
00161 {
00162     bool result = true;
00163
00164     // Check packet ID.
00165     if (id_)
00166     {
00167         result &= packet.id() == id_;
00168     }
00169
00170     // Check source address.
00171     if (source_)
00172     {
00173         result &= source_->contains(packet.source());
00174     }
00175
00176     // Check destination address.
00177     if (dest_)
00178     {
00179         result &= dest_->contains(address);
00180     }
00181
00182     return result;
00183 }
00184
00185
00186 /** Equality comparison.
00187 *
00188 * \relates If
00189 * \param lhs The left hand side if statement.
00190 * \param rhs The right hand side if statement.
00191 * \retval true if \p lhs and \p rhs are the same.
00192 * \retval false if \p lhs and \p rhs are not the same.
00193 */
00194 bool operator==(const If &lhs, const If &rhs)
00195 {
00196     return (lhs.id_ == rhs.id_) && (lhs.source_ == rhs.source_) &&
00197           (lhs.dest_ == rhs.dest_);
00198 }
00199
00200
00201 /** Inequality comparison.

```

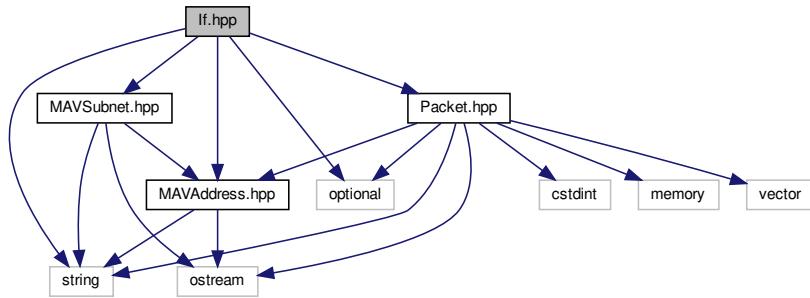
```

00202  *
00203  * \relates If
00204  * \param lhs The left hand side if statement.
00205  * \param rhs The right hand side if statement.
00206  * \retval true if \p lhs and \p rhs are not the same.
00207  * \retval false if \p lhs and \p rhs are the same.
00208 */
00209 bool operator!=(const If &lhs, const If &rhs)
00210 {
00211     return (lhs.id_ != rhs.id_) || (lhs.source_ != rhs.source_) ||
00212         (lhs.dest_ != rhs.dest_);
00213 }
00214
00215
00216 /** Print the \ref If statement to the given output stream.
00217 */
00218 * Some examples are:
00219 * - `if PING from 1.0/8 to 255.0`
00220 * - `if HEARTBEAT from 255.0/8`
00221 * - `if SET_MODE to 255.0`
00222 * - `if from 255.0/8`
00223 */
00224 std::ostream &operator<<(std::ostream &os, const If &if_)
00225 {
00226     // Handle the match any if_.
00227     os << "if";
00228
00229     if (!if_.id_ && !if_.source_ && !if_.dest_)
00230     {
00231         os << " any";
00232         return os;
00233     }
00234
00235     // Print packet name.
00236     if (if_.id_)
00237     {
00238         os << " " << mavlink::name(if_.id_.value());
00239     }
00240
00241     // Print source subnet.
00242     if (if_.source_)
00243     {
00244         os << " from " << if_.source_.value();
00245     }
00246
00247     // Print destination subnet.
00248     if (if_.dest_)
00249     {
00250         os << " to " << if_.dest_.value();
00251     }
00252
00253     return os;
00254 }
```

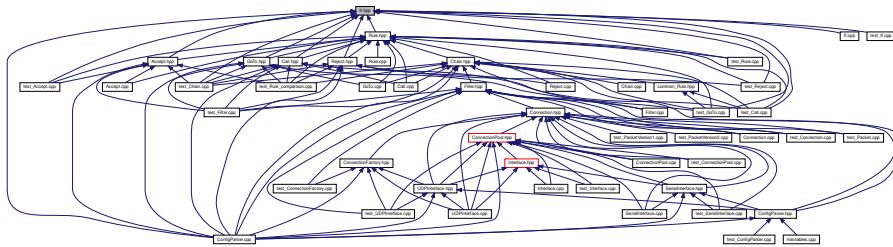
16.71 If.hpp File Reference

```
#include <optional>
#include <string>
#include "MAVAddress.hpp"
#include "MAVSubnet.hpp"
#include "Packet.hpp"
```

Include dependency graph for If.hpp:



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



Classes

- class If

Functions

- bool `operator==` (const `If` &lhs, const `If` &rhs)
 - bool `operator!=` (const `If` &lhs, const `If` &rhs)
 - std::ostream & `operator<<` (std::ostream &os, const `If` &if_)

16.71.1 Function Documentation

16.71.1.1 operator"!=()

```
bool operator!= ( const If & lhs, const If & rhs )
```

16.71.1.2 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const If & if_ )
```

Print the **If** statement to the given output stream.

Some examples are:

- if PING from 1.0/8 to 255.0
- if HEARTBEAT from 255.0/8
- if SET_MODE to 255.0
- if from 255.0/8

Definition at line 224 of file [If.cpp](#).

References [mavlink::name\(\)](#).

Here is the call graph for this function:



16.71.1.3 operator==()

```
bool operator== (
    const If & lhs,
    const If & rhs )
```

16.72 If.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef IF_HPP_
00019 #define IF_HPP_
00020
00021
00022 #include <optional>
00023 #include <string>
00024
00025 #include "MAVAddress.hpp"
00026 #include "MAVSubnet.hpp"
00027 #include "Packet.hpp"
00028
00029
00030 /** An if statement used to determine if a packet matches a rule.
00031 *
00032 * This uses the type, source, and destination of a packet to determine if it
00033 * matches.
00034 */
00035 class If
00036 {
00037     public:
00038         If(std::optional<unsigned long> id = {},
00039             std::optional<MAVSubnet> source = {},
00040             std::optional<MAVSubnet> dest = {});
00041         /** Copy constructor.
00042         *
00043         * \param other If to copy from.
00044         */
00045         If(const If &other) = default;
00046         /** Move constructor.
00047         *
00048         * \param other If to move from.
00049         */
00050         If(If &&other) = default;
00051         If &type(unsigned long id);
00052         If &type(const std::string &name);
00053         If &from(MAVSubnet subnet);
00054         If &from(const std::string &subnet);
00055         If &to(MAVSubnet subnet);
00056         If &to(const std::string &subnet);
00057         bool check(const Packet &packet, const MAVAddress &address) const;
00058         /** Assignment operator.
00059         *
00060         * \param other If to copy from.
00061         */
00062         If &operator=(const If &other) = default;
00063         /** Assignment operator (by move semantics).
00064         *
00065         * \param other If to move from.
00066         */
00067         If &operator=(If &&other) = default;
00068
00069         friend bool operator==(const If &lhs, const If &rhs);
00070         friend bool operator!=(const If &lhs, const If &rhs);
00071         friend std::ostream &operator<<(
00072             std::ostream &os, const If &if_);
00073
00074     private:
00075         std::optional<unsigned long> id_;
00076         std::optional<MAVSubnet> source_;
00077         std::optional<MAVSubnet> dest_;

```

```

00078 };
00079
00080
00081 bool operator==(const If &lhs, const If &rhs);
00082 bool operator!=(const If &lhs, const If &rhs);
00083 std::ostream &operator<<(std::ostream &os, const If &if_);
00084
00085
00086 #endif // IF_HPP_

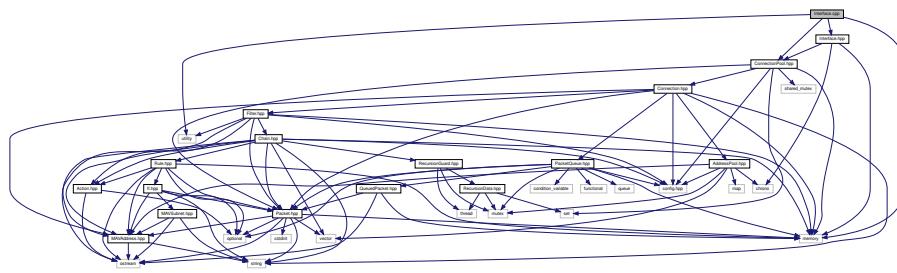
```

16.73 Interface.cpp File Reference

```

#include <memory>
#include <utility>
#include "ConnectionPool.hpp"
#include "Interface.hpp"
Include dependency graph for Interface.cpp:

```



16.74 Interface.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <memory>
00019 #include <utility>
00020
00021 #include "ConnectionPool.hpp"
00022 #include "Interface.hpp"
00023
00024
00025 // Placed here to avoid weak-vtables error.
00026 // LCOV_EXCL_START
00027 Interface::~Interface()
00028 {
00029 }
00030 // LCOV_EXCL_STOP

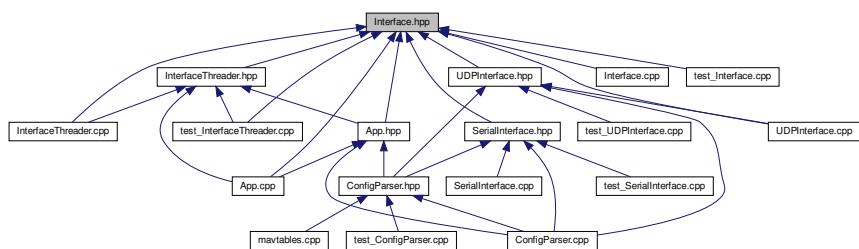
```

```
00031
00032
00033 /** Print the given \ref Interface to the given output stream.
00034 *
00035 * \note This is a polymorphic print. Therefore, it can print any derived
00036 * class as well.
00037 *
00038 * Some examples are:
00039 * ``
00040 * serial {
00041 *     device /dev/ttyUSB0;
00042 *     baudrate 115200;
00043 *     flow_control yes;
00044 * }
00045 * ``
00046 * ``
00047 * udp {
00048 *     port 14500;
00049 *     address 127.0.0.1;
00050 * }
00051 * ``
00052
00053 * \relates Interface
00054 * \param os The output stream to print to.
00055 * \param interface The interface (or any child of the Interface) to print.
00056 * \returns The output stream.
00057 */
00058 std::ostream &operator<<(std::ostream &os, const Interface &interface)
00059 {
00060     return interface.print_(os);
00061 }
```

16.75 Interface.hpp File Reference

```
#include <chrono>
#include <memory>
#include "ConnectionPool.hpp"
Include dependency graph for Interface.hpp
```

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



Classes

- class [Interface](#)

Functions

- std::ostream & [operator<<](#) (std::ostream &os, const [Interface](#) &interface)

16.75.1 Function Documentation

16.75.1.1 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const Interface & interface )
```

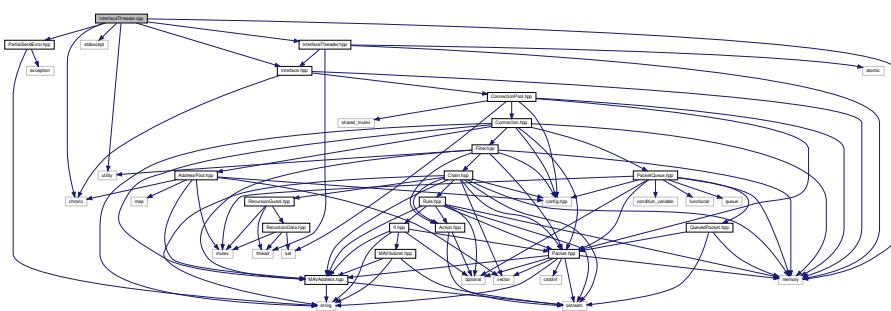
16.76 Interface.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef INTERFACE_HPP_
00019 #define INTERFACE_HPP_
00020
00021
00022 #include <chrono>
00023 #include <memory>
00024
00025 #include "ConnectionPool.hpp"
00026
00027
00028 /** The base class for all interfaces.
00029 *
00030 * Derived classes should add one or more connections to queue packets for
00031 * sending.
00032 */
00033 class Interface
00034 {
00035     public:
00036         virtual ~Interface();
00037         /** Send a packet from one of the interface's connections.
00038         *
00039         * \note Which connection to take this packet from is not defined but
00040         *       it must not starve any one of the \ref Interface's connections.
```

```
00041 * \param timeout The maximum amount of time to wait for a packet to be
00042 * available for sending.
00043 */
00044 virtual void send_packet(const std::chrono::nanoseconds &timeout) = 0;
00045 /** Receive a packet on the interface.
00046 */
00047 * \param timeout The maximum amount of time to wait for incoming data.
00048 */
00049 virtual void receive_packet(
00050     const std::chrono::nanoseconds &timeout) = 0;
00051
00052 friend std::ostream &operator<<(
00053     std::ostream &os, const Interface &interface);
00054
00055 protected:
00056     /** Print the interface to the given output stream.
00057     */
00058     * \param os The output stream to print to.
00059     * \returns The output stream.
00060 */
00061     virtual std::ostream &print_(std::ostream &os) const = 0;
00062
00063 };
00064
00065
00066 std::ostream &operator<<(std::ostream &os, const Interface &interface);
00067
00068
00069 #endif // INTERFACE_HPP_
```

16.77 InterfaceThreader.cpp File Reference

```
#include <chrono>
#include <memory>
#include <stdexcept>
#include <utility>
#include "Interface.hpp"
#include "InterfaceThreader.hpp"
#include "PartialSendError.hpp"
Include dependency graph for InterfaceThreader.cpp:
```



16.78 InterfaceThreader.cpp

```
00001 // MAVLink router and firewall.  
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>  
00003 //  
00004 // This program is free software; you can redistribute it and/or modify
```

```
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
0010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
0011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
0012 // GNU General Public License for more details.
0013 //
0014 // You should have received a copy of the GNU General Public License
0015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
0016
0017
0018 #include <chrono>
0019 #include <memory>
0020 #include <stdexcept>
0021 #include <utility>
0022
0023 #include "Interface.hpp"
0024 #include "InterfaceThreader.hpp"
0025 #include "PartialSendError.hpp"
0026
0027
0028 /** The transmitting thread runner.
0029 *
0030 * This handles all transmission related tasks in a separate thread.
0031 */
0032 void InterfaceThreader::tx_runner_()
0033 {
0034     while (running_.load())
0035     {
0036         try
0037         {
0038             interface_->send_packet(timeout_);
0039         }
0040         // Ignore partial write errors on shutdown.
0041         // TODO: A better way to handle this error might be in order. Not even
0042         // sure exactly why it happens on close.
0043         catch (const PartialSendError &)
0044         {
0045             if (running_.load())
0046             {
0047                 throw;
0048             }
0049         }
0050     }
0051 }
0052
0053
0054 /** The receiving thread runner.
0055 *
0056 * This handles all receiving related tasks in a separate thread.
0057 */
0058 void InterfaceThreader::rx_runner_()
0059 {
0060     while (running_.load())
0061     {
0062         interface_->receive_packet(timeout_);
0063     }
0064 }
0065
0066
0067 /** Construct and optionally start an interface threader.
0068 *
0069 * \param interface The \ref Interface to run in TX/RX threads. It's \ref
0070 * Interface::send_packet and \ref Interface::receive_packet methods will
0071 * be called repeatedly in two separate worker threads.
0072 * \param timeout The maximum amount of time to wait for incoming data or a
0073 * packet to transmit. The default value is 100000 us (100 ms).
0074 * \param start_threads Set to \ref InterfaceThreader::START (the default
0075 * value) to start the interface (including worker threads) on
0076 * construction. Set to \ref InterfaceThreader::DELAY_START to delay
0077 * starting the interface (and worker threads) until the \ref start method
0078 * is called.
0079 */
0080 InterfaceThreader::InterfaceThreader(
0081     std::unique_ptr<Interface> interface,
0082     std::chrono::microseconds timeout,
0083     Threads start_threads)
0084 : interface_(std::move(interface)),
0085     timeout_(std::move(timeout)),
```

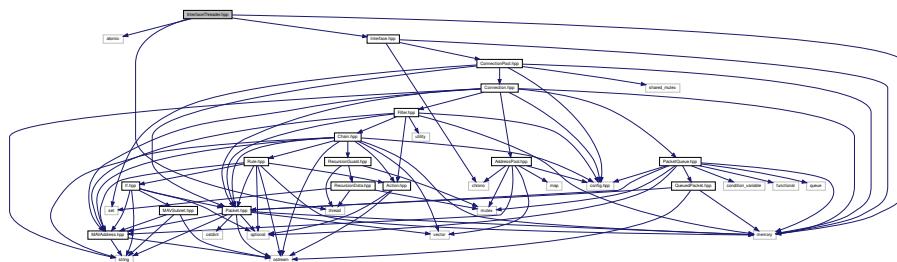
```

00086     running_(false)
00087 {
00088     if (start_threads == InterfaceThreader::START)
00089     {
00090         start();
00091     }
00092 }
00093
00094
00095 /** Shutdown the interface and its associated worker threads.
00096 */
00097 InterfaceThreader::~InterfaceThreader()
00098 {
00099     shutdown();
00100 }
00101
00102
00103 /** Start the worker threads for the interface.
00104 *
00105 * This starts the receiver and transmitter threads.
00106 */
00107 void InterfaceThreader::start()
00108 {
00109     running_.store(true);
00110     tx_thread_ = std::thread(&InterfaceThreader::tx_runner_, this);
00111     rx_thread_ = std::thread(&InterfaceThreader::rx_runner_, this);
00112 }
00113
00114
00115 /** Shutdown the interface and its associated worker threads.
00116 *
00117 * \note This will always be called by the interface's destructor.
00118 */
00119 void InterfaceThreader::shutdown()
00120 {
00121     if (running_.load())
00122     {
00123         running_.store(false);
00124         tx_thread_.join();
00125         rx_thread_.join();
00126     }
00127 }

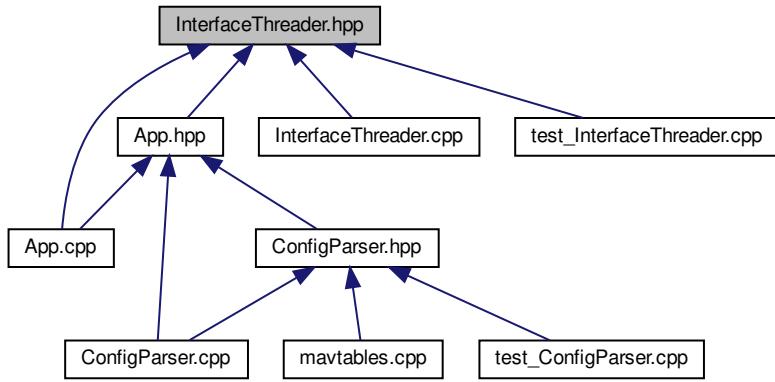
```

16.79 InterfaceThreader.hpp File Reference

```
#include <atomic>
#include <memory>
#include <thread>
#include "Interface.hpp"
Include dependency graph for InterfaceThreader.hpp:
```



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



Classes

- class [InterfaceThreader](#)

16.80 InterfaceThreader.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef INTERFACETHREADER_HPP_
00019 #define INTERFACETHREADER_HPP_
00020
00021
00022 #include <atomic>
00023 #include <memory>
00024 #include <thread>
00025
00026 #include "Interface.hpp"
00027
00028
00029 /** A multithreaded interface runner.
00030 *
00031 * This class runs a given interface. It does this by calling the \ref
00032 * Interface's \ref Interface::send_packet and \ref Interface::receive_packet
00033 * methods repeatedly until the runner is shutdown. This is a threaded runner
00034 * and thus it runs these methods in two separate threads which are managed by
00035 * the \ref InterfaceThreader class.
  
```

```

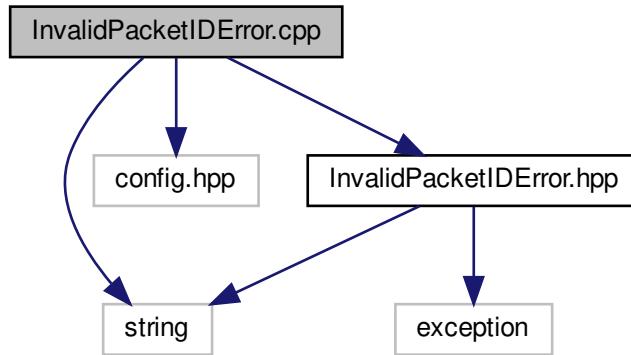
00036  *
00037  * %Call the \ref shutdown method to stop running the interface and cleanup the
00038  * threads.
00039  *
00040  * \note This is the only place where threading occurs in mavtables.
00041 */
00042 class InterfaceThreader
00043 {
00044     public:
00045         enum Threads
00046     {
00047             START, //!< Start the interface (and worker threads) immediately.
00048             DELAY_START //!< Delay starting, use \ref start to launch threads.
00049         };
00050     InterfaceThreader(
00051         std::unique_ptr<Interface> interface,
00052         std::chrono::microseconds = std::chrono::microseconds(100000),
00053         Threads start_threads = InterfaceThreader::START);
00054     InterfaceThreader(const InterfaceThreader &other) = delete;
00055     InterfaceThreader(InterfaceThreader &&other) = delete;
00056     ~InterfaceThreader();
00057     void start();
00058     void shutdown();
00059     InterfaceThreader &operator=(const
00060         InterfaceThreader &other) = delete;
00061     InterfaceThreader &operator=(InterfaceThreader &&other)
00062     = delete;
00063     private:
00064         // Variables
00065         std::unique_ptr<Interface> interface_;
00066         std::thread tx_thread_;
00067         std::thread rx_thread_;
00068         std::chrono::microseconds timeout_;
00069         std::atomic<bool> running_;
00070         // Methods
00071         void tx_runner_();
00072     };
00073
00074
00075 #endif // INTERFACETHREADER_HPP_

```

16.81 InvalidPacketIDError.cpp File Reference

```
#include <string>
#include "config.hpp"
#include "InvalidPacketIDError.hpp"
```

Include dependency graph for InvalidPacketIDError.cpp:



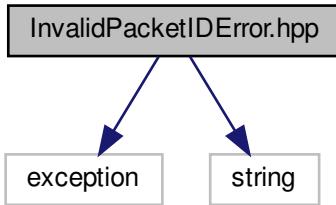
16.82 InvalidPacketIDError.cpp

```

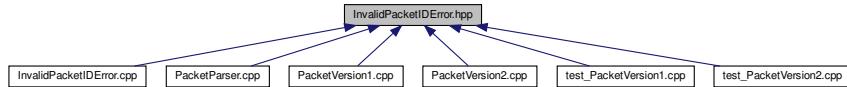
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <string>
00019
00020 #include "config.hpp"
00021 #include "InvalidPacketIDError.hpp"
00022
00023
00024 /** Construct a InvalidPacketIDError given a packet ID.
00025 *
00026 * \param id The packet ID.
00027 */
00028 InvalidPacketIDError::InvalidPacketIDError(unsigned long id)
00029 {
00030     message_ = "Packet ID (#" + std::to_string(id) +
00031                 ") is not part of the '" MAVLINK_DIALECT "' MAVLink dialect.";
00032 }
00033
00034
00035 /** Return error message string.
00036 *
00037 * \returns Error message string.
00038 */
00039 const char *InvalidPacketIDError::what() const noexcept
00040 {
00041     return message_.c_str();
00042 }
  
```

16.83 InvalidPacketIDError.hpp File Reference

```
#include <exception>
#include <string>
Include dependency graph for InvalidPacketIDError.hpp:
```



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



Classes

- class [InvalidPacketIDError](#)

16.84 InvalidPacketIDError.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef INVALIDPACKETIDERROR_HPP_
  
```

```

00019 #define INVALIDPACKETIDERROR_HPP_
00020
00021
00022 #include <exception>
00023 #include <string>
00024
00025
00026 /** Exception type emitted when parsing a packet with an invalid ID.
00027 */
00028 class InvalidPacketIDError : public std::exception
00029 {
00030     public:
00031         InvalidPacketIDError(unsigned long id);
00032         const char *what() const noexcept;
00033
00034     private:
00035         std::string message_;
00036 };
00037
00038
00039 #endif // INVALIDPACKETIDERROR_HPP_

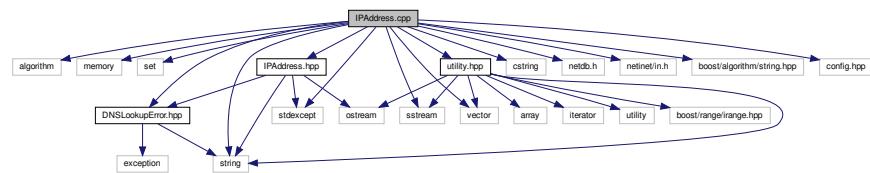
```

16.85 IPAddress.cpp File Reference

```

#include <algorithm>
#include <memory>
#include <set>
#include <sstream>
#include <stdexcept>
#include <string>
#include <vector>
#include <cstring>
#include <netdb.h>
#include <netinet/in.h>
#include <boost/algorithm/string.hpp>
#include "config.hpp"
#include "DNSLookupError.hpp"
#include "IPAddress.hpp"
#include "utility.hpp"
Include dependency graph for IPAddress.cpp:

```



Functions

- `std::ostream & operator<< (std::ostream &os, const IPAddress &ipaddress)`

16.85.1 Function Documentation

16.85.1.1 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const IPAddress & ipaddress )
```

Print the IP address to the given output stream.

The format is "<IP Address>" or "<IP Address>:<Port Number>" if the port number is nonzero and "<IP Address>" if the port is 0.

Some examples are:

- 127.0.0.1
- 127.0.0.1:14555
- 183.125.120.42:443

Note

The string constructor [IPAddress\(std::string\)](#) and the output stream operator are inverses and thus:

```
std::string addr = "127.0.0.1:14555"
str(IPAddress(addr)) == addr
```

Parameters

<i>os</i>	The output stream to print to.
<i>ipaddress</i>	The IP address to print.

Returns

The output stream.

Definition at line 340 of file [IPAddress.cpp](#).

References [to_bytes\(\)](#).

Here is the call graph for this function:



16.86 IPAddress.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <algorithm>
00019 #include <memory>
00020 #include <set>
00021 #include <sstream>
00022 #include <stdexcept>
00023 #include <string>
00024 #include <vector>
00025
00026 #include <cstring>
00027 #include <netdb.h>
00028 #include <netinet/in.h>
00029
00030 #include <boost/algorithm/string.hpp>
00031
00032 #include "config.hpp"
00033 #include "DNSLookupError.hpp"
00034 #include "IPAddress.hpp"
00035 #include "utility.hpp"
00036
00037
00038 // Private functions.
00039 #ifdef UNIX
00040 namespace
00041 {
00042     IPAddress unix_dnslookup(const std::string &url, unsigned int port);
00043 }
00044 #elif WINDOWS
00045 #endif
00046
00047
00048 /** Construct IP address from address and port number.
00049 *
00050 * \param address Numeric (32 bit) IP address.
00051 * \param port 16 bit port number.
00052 * \throws std::out_of_range if either the IP address or the port number is
00053 *     outside of the respectively allowed 32 or 16 bit ranges.
00054 */
00055 void IPAddress::construct_(unsigned long address, unsigned int port)
00056 {
00057     if (address > 0xFFFFFFFF)
00058     {
00059         std::stringstream ss;
00060         ss << "Address (0x"
00061             << std::uppercase << std::hex << address << std::nouppercase
00062             << ") is outside of the allowed range (0x00000000 - 0xFFFFFFFF).";
00063         throw std::out_of_range(ss.str());
00064     }
00065
00066     if (port > 65535)
00067     {
00068         throw std::out_of_range(
00069             "port number (" + std::to_string(port) +
00070             ") is outside of the allowed range (0 - 65535).");
00071     }
00072
00073     address_ = address;
00074     port_ = port;
00075 }
00076
00077
```

```
00078 /** Construct IP address from another IP address, changing the port number.
00079 *
00080 * Copy constructor that also changes the port.
00081 *
00082 * \param other IP address to copy from.
00083 * \param port Port number (0 - 65535). A port number of 0 has the special
00084 * meaning of no specific port.
00085 * \throws std::out_of_range if the port number is outside of the allowed 16
00086 * bit range.
00087 */
00088 IPAddress::IPAddress(const IPAddress &other, unsigned int port)
00089 {
00090     construct_(other.address_, port);
00091 }
00092
00093
00094 /** Construct IP address from address and port number.
00095 *
00096 * \param address 32-bit IP address in system byte order (0x00000000 -
00097 * 0xFFFFFFFF).
00098 * \param port Port number (0 - 65535). A port number of 0 has the special
00099 * meaning of no specific port.
00100 * \throws std::out_of_range if either the IP address or the port number is
00101 * outside of the respectively allowed 32 or 16 bit ranges.
00102 */
00103 IPAddress::IPAddress(unsigned long address, unsigned int port)
00104 {
00105     construct_(address, port);
00106 }
00107
00108
00109 /** Construct IP address from a string.
00110 *
00111 * Parse a string of the form "<IP Address>" or "<IP Address>:<Port Number>".
00112 *
00113 * Some examples are:
00114 * - "127.0.0.1"
00115 * - "127.0.0.1:8888"
00116 * - "183.125.120.42:443"
00117 *
00118 * %If no port is given the 0 port is used which represents no specific port.
00119 *
00120 * \param address String representing the IP address and optionally the port
00121 * number.
00122 * \throws std::invalid_argument if the string does not represent a valid IP
00123 * address.
00124 * \throws std::out_of_range if an address octet or the port number is out of
00125 * range.
00126 */
00127 IPAddress::IPAddress(std::string address)
00128 {
00129     // Separate port from address.
00130     unsigned int port = 0;
00131     std::vector<std::string> parts;
00132     boost::split(parts, address, [] (char c)
00133     {
00134         return c == ':';
00135     });
00136
00137     // Read port.
00138     if (parts.size() == 2)
00139     {
00140         std::istringstream(parts.back()) >> port;
00141         parts.pop_back();
00142     }
00143
00144     if (parts.size() != 1)
00145     {
00146         throw std::invalid_argument("Invalid IP address string.");
00147     }
00148
00149     address = parts.back();
00150
00151     // Check validity of address string.
00152     if (address.size() < 7 || !(isdigit(address.front())))
00153         || !isdigit(address.back()))
00154     {
00155         throw std::invalid_argument("Invalid IP address string.");
00156     }
00157
00158     for (auto c : address)
```

```
00159     {
00160         if (!(c == '.' || isdigit(c)))
00161         {
00162             throw std::invalid_argument("Invalid IP address string.");
00163         }
00164     }
00165
00166     // Read address.
00167     std::replace(address.begin(), address.end(), '.', ' ');
00168     std::vector<unsigned long> octets;
00169     std::istringstream ss(address);
00170     unsigned long octet;
00171
00172     while (ss >> octet)
00173     {
00174         octets.push_back(octet);
00175     }
00176
00177     // Ensure proper number of octets.
00178     if (octets.size() != 4)
00179     {
00180         throw std::invalid_argument("Invalid IP address string.");
00181     }
00182
00183     // Ensure octets are between 0 and 255.
00184     for (auto i : octets)
00185         if (i > 255)
00186         {
00187             throw std::out_of_range(
00188                 "Address octet (" + std::to_string(i) +
00189                 ") is outside of the allowed range (0 - 255).");
00190         }
00191     }
00192     {
00193     }
00194
00195     construct_((octets[0] << 8 * 3) | (octets[1] << 8 * 2) |
00196                 (octets[2] << 8) | octets[3], port);
00197 }
00198
00199
00200 /** Return the IP address.
00201 */
00202 * \returns The 32-bit IP address (in system byte order) as an integer
00203 * (0x00000000 - 0xFFFFFFFF).
00204 */
00205 unsigned long IPAddress::address() const
00206 {
00207     return address_;
00208 }
00209
00210
00211 /** Return the port.
00212 */
00213 * \returns The port number (0 - 65535).
00214 */
00215 unsigned int IPAddress::port() const
00216 {
00217     return port_;
00218 }
00219
00220
00221 /** Equality comparison.
00222 */
00223 * \note Compares address and port number.
00224 *
00225 * \relates IPAddress
00226 * \param lhs The left hand side IP address.
00227 * \param rhs The right hand side IP address.
00228 * \retval true if \p lhs and \p rhs have the same address and port.
00229 * \retval false if \p lhs and \p rhs do not have the same address and port.
00230 */
00231 bool operator==(const IPAddress &lhs, const IPAddress &rhs)
00232 {
00233     return (lhs.address() == rhs.address()) && (lhs.port() == rhs.
00234     port());
00235
00236
00237 /** Inequality comparison.
00238 */
00239
```

```

00239 * \note Compares address and port number.
00240 *
00241 * \relates IPAddress
00242 * \param lhs The left hand side IP address.
00243 * \param rhs The right hand side IP address.
00244 * \retval true if \p lhs and \p rhs do not have the same address and port.
00245 * \retval false if \p lhs and \p rhs have the same address and port.
00246 */
00247 bool operator!=(const IPAddress &lhs, const IPAddress &rhs)
00248 {
00249     return (lhs.address() != rhs.address()) || (lhs.port() != rhs.
00250         port());
00251
00252
00253 /** Less than comparison.
00254 *
00255 * \note The address is considered first followed by the port.
00256 *
00257 * \relates IPAddress
00258 * \param lhs The left hand side IP address.
00259 * \param rhs The right hand side IP address.
00260 * \retval true if \p lhs is less than \p rhs.
00261 * \retval false if \p lhs is not less than \p rhs.
00262 */
00263 bool operator<(const IPAddress &lhs, const IPAddress &rhs)
00264 {
00265     return (lhs.address() < rhs.address()) || ((lhs.address() == rhs.
00266         address())
00267             && (lhs.port() < rhs.port()));
00268
00269
00270 /** Greater than comparison.
00271 *
00272 * \note The address is considered first followed by the port.
00273 *
00274 * \relates IPAddress
00275 * \param lhs The left hand side IP address.
00276 * \param rhs The right hand side IP address.
00277 * \retval true if \p lhs is greater than \p rhs.
00278 * \retval false if \p lhs is not greater than \p rhs.
00279 */
00280 bool operator>(const IPAddress &lhs, const IPAddress &rhs)
00281 {
00282     return (lhs.address() > rhs.address()) || ((lhs.address() == rhs.
00283         address())
00284             && (lhs.port() > rhs.port()));
00285
00286
00287 /** Less than or equal comparison.
00288 *
00289 * \note The address is considered first followed by the port.
00290 *
00291 * \relates IPAddress
00292 * \param lhs The left hand side IP address.
00293 * \param rhs The right hand side IP address.
00294 * \retval true if \p lhs is less than or equal to \p rhs.
00295 * \retval false if \p lhs is greater than \p rhs.
00296 */
00297 bool operator<=(const IPAddress &lhs, const IPAddress &rhs)
00298 {
00299     return (lhs < rhs) || (lhs == rhs);
00300 }
00301
00302
00303 /** Greater than comparison.
00304 *
00305 * \note The address is considered first followed by the port.
00306 *
00307 * \relates IPAddress
00308 * \param lhs The left hand side IP address.
00309 * \param rhs The right hand side IP address.
00310 * \retval true if \p lhs is greater than or equal to \p rhs.
00311 * \retval false if \p lhs is less than \p rhs.
00312 */
00313 bool operator>=(const IPAddress &lhs, const IPAddress &rhs)
00314 {
00315     return (lhs > rhs) || (lhs == rhs);
00316 }

```

```
00317
00318
00319 /** Print the IP address to the given output stream.
00320 *
00321 * The format is "<IP Address>" or "<IP Address>:<Port Number>" if the port
00322 * number is nonzero an "<IP Address>" if the port is 0.
00323 *
00324 * Some examples are:
00325 * - '127.0.0.1'
00326 * - '127.0.0.1:14555'
00327 * - '183.125.120.42:443'
00328 *
00329 * \note The string constructor \ref IPAddress(std::string) and the output
00330 * stream operator are inverses and thus:
00331 *
00332 * std::string addr = "127.0.0.1:14555"
00333 * str(IPAddress(addr)) == addr
00334 * ``
00335 *
00336 * \param os The output stream to print to.
00337 * \param ipaddress The IP address to print.
00338 * \returns The output stream.
00339 */
00340 std::ostream &operator<<(std::ostream &os, const IPAddress &ipaddress)
00341 {
00342     const auto bytes = to_bytes(ipaddress.address_);
00343
00344     for (size_t i = 3; i > 0; --i)
00345     {
00346         os << static_cast<int>(bytes[i]) << ".";
00347     }
00348
00349     os << static_cast<int>(bytes[0]);
00350
00351     if (ipaddress.port_ != 0)
00352     {
00353         os << ":" << ipaddress.port_;
00354     }
00355
00356     return os;
00357 }
00358
00359
00360 /** Lookup an IP address based on a hostname.
00361 *
00362 * \warning Currently only supports IPv4.
00363 *
00364 * \note Currently only UNIX based operating system are supported.
00365 *
00366 * \relates IPAddress
00367 * \param url The URL to get an IP address for.
00368 * \returns IP addresses corresponding to the given URL.
00369 * \throws DNSLookupError if the address cannot be found.
00370 */
00371 IPAddress dnslookup(const std::string &url)
00372 {
00373     #ifdef UNIX
00374         return unix_dnslookup(url, 0);
00375     #elif WINDOWS
00376         return win32_dnslookup(url, 0);
00377     #endif
00378 }
00379
00380
00381 #ifdef UNIX
00382
00383 namespace
00384 {
00385
00386     /** Lookup an IP address based on a hostname.
00387     *
00388     * This version is UNIX only and will be called by \ref dnslookup on UNIX
00389     * systems.
00390     *
00391     * \relates IPAddress
00392     * \param url The URL to get an IP address for.
00393     * \returns IP addresses corresponding to the given URL.
00394     * \throws DNSLookupError if the address cannot be found.
00395     *
00396     */
00397     IPAddress unix_dnslookup(const std::string &url, unsigned int port)
```

```

00398     {
00399         std::set<IPAddress> addresses;
00400         // Setup hints.
00401         struct addrinfo hints;
00402         std::memset(&hints, 0, sizeof(hints));
00403         hints.ai_family = AF_INET; // IPv4 only (for now)
00404         hints.ai_protocol = static_cast<int>(port);
00405         // Get IP addresses.
00406         struct addrinfo *result_ptr = nullptr;
00407
00408         if (getaddrinfo(url.c_str(), nullptr, &hints, &result_ptr))
00409         {
00410             throw DNSLookupError(url);
00411         }
00412
00413         std::unique_ptr<struct addrinfo, void(*)(struct addrinfo *)>
00414         result(result_ptr, freeaddrinfo);
00415         result_ptr = nullptr;
00416
00417         for (result_ptr = result.get(); result_ptr != nullptr;
00418              result_ptr = result_ptr->ai_next)
00419         {
00420             struct sockaddr_in *address_ptr =
00421                 reinterpret_cast<struct sockaddr_in *>(result_ptr->ai_addr);
00422             unsigned long address = ntohl(address_ptr->sin_addr.s_addr);
00423             addresses.insert(IPAddress(address, port));
00424         }
00425
00426         // This should never be true but it's here just in case.
00427         if (addresses.empty())
00428         {
00429             // LCOV_EXCL_START
00430             throw DNSLookupError(url);
00431             // LCOV_EXCL_STOP
00432         }
00433
00434         return *(addresses.begin());
00435     }
00436
00437 }
00438
00439 #elif WINDOWS
00440
00441 // Place a windows implementation of windows_dnslookup here.
00442
00443 #endif

```

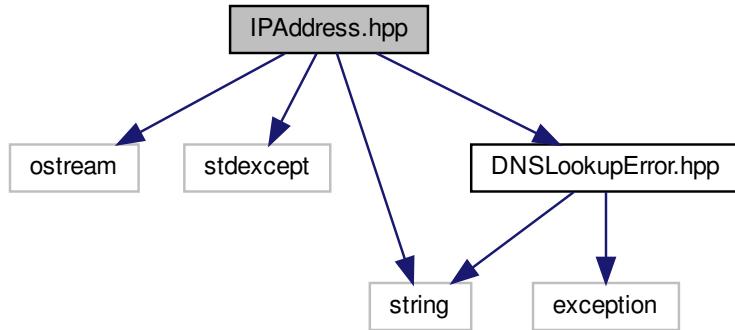
16.87 IPAddress.hpp File Reference

```

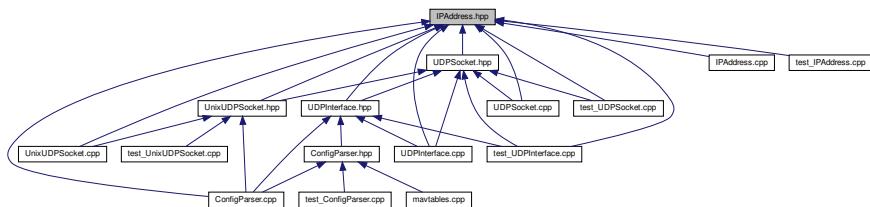
#include <iostream>
#include <stdexcept>
#include <string>
#include "DNSLookupError.hpp"

```

Include dependency graph for IPAddress.hpp:



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



Classes

- class `IPAddress`

Functions

- `bool operator==(const IPAddress &lhs, const IPAddress &rhs)`
 - `bool operator!=(const IPAddress &lhs, const IPAddress &rhs)`
 - `bool operator<(const IPAddress &lhs, const IPAddress &rhs)`
 - `bool operator>(const IPAddress &lhs, const IPAddress &rhs)`
 - `bool operator<=(const IPAddress &lhs, const IPAddress &rhs)`
 - `bool operator>=(const IPAddress &lhs, const IPAddress &rhs)`
 - `std::ostream &operator<< (std::ostream &os, const IPAddress &ipaddress)`
 - `IPAddress dnslookup (const std::string &url)`

16.87.1 Function Documentation

16.87.1.1 dnslookup()

```
IPAddress dnslookup (
    const std::string & url ) [related]
```

16.87.1.2 operator"!=()

```
bool operator!= (
    const IPAddress & lhs,
    const IPAddress & rhs ) [related]
```

16.87.1.3 operator<()

```
bool operator< (
    const IPAddress & lhs,
    const IPAddress & rhs ) [related]
```

16.87.1.4 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const IPAddress & ipaddress )
```

Print the IP address to the given output stream.

The format is "<IP Address>" or "<IP Address>:<Port Number>" if the port number is nonzero and "<IP Address>" if the port is 0.

Some examples are:

- 127.0.0.1
- 127.0.0.1:14555
- 183.125.120.42:443

Note

The string constructor `IPAddress(std::string)` and the output stream operator are inverses and thus:

```
std::string addr = "127.0.0.1:14555"
str(IPAddress(addr)) == addr
```

Parameters

<i>os</i>	The output stream to print to.
<i>ipaddress</i>	The IP address to print.

Returns

The output stream.

Definition at line 340 of file [IPAddress.cpp](#).

References [to_bytes\(\)](#).

Here is the call graph for this function:

**16.87.1.5 operator<=()**

```
bool operator<= (
    const IPAddress & lhs,
    const IPAddress & rhs ) [related]
```

16.87.1.6 operator==()

```
bool operator== (
    const IPAddress & lhs,
    const IPAddress & rhs ) [related]
```

16.87.1.7 operator>()

```
bool operator> (
    const IPAddress & lhs,
    const IPAddress & rhs ) [related]
```

16.87.1.8 operator>=()

```
bool operator>= (
    const IPAddress & lhs,
    const IPAddress & rhs ) [related]
```

16.88 IPAddress.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef IPADDRESS_HPP_
00019 #define IPADDRESS_HPP_
00020
00021
00022 #include <iostream>
00023 #include <stdexcept>
00024 #include <string>
00025
00026 #include "DNSLookupError.hpp"
00027
00028 /**
00029  * An IP address with optional port number.
00030 */
00031 class IPAddress
00032 {
00033     public:
00034         /**
00035          * \param other IP address to copy from.
00036          */
00037         IPAddress(const IPAddress &other) = default;
00038         /**
00039          * \param other IP address to move from.
00040          */
00041         IPAddress(IPAddress &&other) = default;
00042         /**
00043          * \param address IP address
00044          * \param port port number
00045          */
00046         IPAddress(unsigned long address, unsigned int port = 0);
00047         IPAddress(std::string address);
00048         unsigned long address() const;
00049         unsigned int port() const;
00050         /**
00051             * Assignment operator.
00052             */
00053 }
```

```

00051     * \param other IP address to copy from.
00052     */
00053     IPAddress &operator=(const IPAddress &other) = default;
00054     /** Assignment operator (by move semantics).
00055     *
00056     * \param other IP address to move from.
00057     */
00058     IPAddress &operator=(IPAddress &&other) = default;
00059
00060     friend std::ostream &operator<<( 
00061         std::ostream &os, const IPAddress &ipaddress);
00062
00063     private:
00064     unsigned long address_;
00065     unsigned int port_;
00066     void construct_(unsigned long address, unsigned int port);
00067 };
00068
00069
00070 bool operator==(const IPAddress &lhs, const IPAddress &rhs);
00071 bool operator!=(const IPAddress &lhs, const IPAddress &rhs);
00072 bool operator<(const IPAddress &lhs, const IPAddress &rhs);
00073 bool operator>(const IPAddress &lhs, const IPAddress &rhs);
00074 bool operator<=(const IPAddress &lhs, const IPAddress &rhs);
00075 bool operator>=(const IPAddress &lhs, const IPAddress &rhs);
00076 std::ostream &operator<<(std::ostream &os, const IPAddress &ipaddress);
00077
00078
00079 IPAddress dnslookup(const std::string &url);
00080
00081
00082 #endif // IPADDRESS_HPP_

```

16.89 LICENSE.md File Reference

16.90 LICENSE.md

```

00001 GNU General Public License {#license}
00002 =====
00003
00004 _Version 2, June 1991_
00005 _Copyright © 1989, 1991 Free Software Foundation, Inc.,_
00006 _51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA_
00007
00008 Everyone is permitted to copy and distribute verbatim copies
00009 of this license document, but changing it is not allowed.
00010
00011 ### Preamble
00012
00013 The licenses for most software are designed to take away your
00014 freedom to share and change it. By contrast, the GNU General Public
00015 License is intended to guarantee your freedom to share and change free
00016 software--to make sure the software is free for all its users. This
00017 General Public License applies to most of the Free Software
00018 Foundation's software and to any other program whose authors commit to
00019 using it. (Some other Free Software Foundation software is covered by
00020 the GNU Lesser General Public License instead.) You can apply it to
00021 your programs, too.
00022
00023 When we speak of free software, we are referring to freedom, not
00024 price. Our General Public Licenses are designed to make sure that you
00025 have the freedom to distribute copies of free software (and charge for
00026 this service if you wish), that you receive source code or can get it
00027 if you want it, that you can change the software or use pieces of it
00028 in new free programs; and that you know you can do these things.
00029
00030 To protect your rights, we need to make restrictions that forbid
00031 anyone to deny you these rights or to ask you to surrender the rights.
00032 These restrictions translate to certain responsibilities for you if you
00033 distribute copies of the software, or if you modify it.
00034
00035 For example, if you distribute copies of such a program, whether

```

00036 gratis or for a fee, you must give the recipients all the rights that
00037 you have. You must make sure that they, too, receive or can get the
00038 source code. And you must show them these terms so they know their
00039 rights.
00040
00041 We protect your rights with two steps: ****(1)**** copyright the software, and
00042 ****(2)**** offer you this license which gives you legal permission to copy,
00043 distribute and/or modify the software.
00044
00045 Also, for each author's protection and ours, we want to make certain
00046 that everyone understands that there is no warranty for this free
00047 software. If the software is modified by someone else and passed on, we
00048 want its recipients to know that what they have is not the original, so
00049 that any problems introduced by others will not reflect on the original
00050 authors' reputations.
00051
00052 Finally, any free program is threatened constantly by software
00053 patents. We wish to avoid the danger that redistributors of a free
00054 program will individually obtain patent licenses, in effect making the
00055 program proprietary. To prevent this, we have made it clear that any
00056 patent must be licensed for everyone's free use or not licensed at all.
00057
00058 The precise terms and conditions for copying, distribution and
00059 modification follow.
00060
00061 **### TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**
00062
00063 ****0.**** This License applies to any program or other work which contains
00064 a notice placed by the copyright holder saying it may be distributed
00065 under the terms of this General Public License. The "Program", below,
00066 refers to any such program or work, and a "work based on the Program"
00067 means either the Program or any derivative work under copyright law:
00068 that is to say, a work containing the Program or a portion of it,
00069 either verbatim or with modifications and/or translated into another
00070 language. (Hereinafter, translation is included without limitation in
00071 the term "modification".) Each licensee is addressed as "you".
00072
00073 Activities other than copying, distribution and modification are not
00074 covered by this License; they are outside its scope. The act of
00075 running the Program is not restricted, and the output from the Program
00076 is covered only if its contents constitute a work based on the
00077 Program (independent of having been made by running the Program).
00078 Whether that is true depends on what the Program does.
00079
00080 ****1.**** You may copy and distribute verbatim copies of the Program's
00081 source code as you receive it, in any medium, provided that you
00082 conspicuously and appropriately publish on each copy an appropriate
00083 copyright notice and disclaimer of warranty; keep intact all the
00084 notices that refer to this License and to the absence of any warranty;
00085 and give any other recipients of the Program a copy of this License
00086 along with the Program.
00087
00088 You may charge a fee for the physical act of transferring a copy, and
00089 you may at your option offer warranty protection in exchange for a fee.
00090
00091 ****2.**** You may modify your copy or copies of the Program or any portion
00092 of it, thus forming a work based on the Program, and copy and
00093 distribute such modifications or work under the terms of Section 1
00094 above, provided that you also meet all of these conditions:
00095
00096 * ****a)**** You must cause the modified files to carry prominent notices
00097 stating that you changed the files and the date of any change.
00098 * ****b)**** You must cause any work that you distribute or publish, that in
00099 whole or in part contains or is derived from the Program or any
00100 part thereof, to be licensed as a whole at no charge to all third
00101 parties under the terms of this License.
00102 * ****c)**** If the modified program normally reads commands interactively
00103 when run, you must cause it, when started running for such
00104 interactive use in the most ordinary way, to print or display an
00105 announcement including an appropriate copyright notice and a
00106 notice that there is no warranty (or else, saying that you provide
00107 a warranty) and that users may redistribute the program under
00108 these conditions, and telling the user how to view a copy of this
00109 License. (Exception: if the Program itself is interactive but
00110 does not normally print such an announcement, your work based on
00111 the Program is not required to print an announcement.)
00112
00113 These requirements apply to the modified work as a whole. If
00114 identifiable sections of that work are not derived from the Program,
00115 and can be reasonably considered independent and separate works in
00116 themselves, then this License, and its terms, do not apply to those

00117 sections when you distribute them as separate works. But when you
00118 distribute the same sections as part of a whole which is a work based
00119 on the Program, the distribution of the whole must be on the terms of
00120 this License, whose permissions for other licensees extend to the
00121 entire whole, and thus to each and every part regardless of who wrote it.
00122
00123 Thus, it is not the intent of this section to claim rights or contest
00124 your rights to work written entirely by you; rather, the intent is to
00125 exercise the right to control the distribution of derivative or
00126 collective works based on the Program.
00127
00128 In addition, mere aggregation of another work not based on the Program
00129 with the Program (or with a work based on the Program) on a volume of
00130 a storage or distribution medium does not bring the other work under
00131 the scope of this License.
00132
00133 ****3.**** You may copy and distribute the Program (or a work based on it,
00134 under Section 2) in object code or executable form under the terms of
00135 Sections 1 and 2 above provided that you also do one of the following:
00136
00137 * ****a)**** Accompany it with the complete corresponding machine-readable
00138 source code, which must be distributed under the terms of Sections
00139 1 and 2 above on a medium customarily used for software interchange; or,
00140 * ****b)**** Accompany it with a written offer, valid for at least three
00141 years, to give any third party, for a charge no more than your
00142 cost of physically performing source distribution, a complete
00143 machine-readable copy of the corresponding source code, to be
00144 distributed under the terms of Sections 1 and 2 above on a medium
00145 customarily used for software interchange; or,
00146 * ****c)**** Accompany it with the information you received as to the offer
00147 to distribute corresponding source code. (This alternative is
00148 allowed only for noncommercial distribution and only if you
00149 received the program in object code or executable form with such
00150 an offer, in accord with Subsection b above.)
00151
00152 The source code for a work means the preferred form of the work for
00153 making modifications to it. For an executable work, complete source
00154 code means all the source code for all modules it contains, plus any
00155 associated interface definition files, plus the scripts used to
00156 control compilation and installation of the executable. However, as a
00157 special exception, the source code distributed need not include
00158 anything that is normally distributed (in either source or binary
00159 form) with the major components (compiler, kernel, and so on) of the
00160 operating system on which the executable runs, unless that component
00161 itself accompanies the executable.
00162
00163 If distribution of executable or object code is made by offering
00164 access to copy from a designated place, then offering equivalent
00165 access to copy the source code from the same place counts as
00166 distribution of the source code, even though third parties are not
00167 compelled to copy the source along with the object code.
00168
00169 ****4.**** You may not copy, modify, sublicense, or distribute the Program
00170 except as expressly provided under this License. Any attempt
00171 otherwise to copy, modify, sublicense or distribute the Program is
00172 void, and will automatically terminate your rights under this License.
00173 However, parties who have received copies, or rights, from you under
00174 this License will not have their licenses terminated so long as such
00175 parties remain in full compliance.
00176
00177 ****5.**** You are not required to accept this License, since you have not
00178 signed it. However, nothing else grants you permission to modify or
00179 distribute the Program or its derivative works. These actions are
00180 prohibited by law if you do not accept this License. Therefore, by
00181 modifying or distributing the Program (or any work based on the
00182 Program), you indicate your acceptance of this License to do so, and
00183 all its terms and conditions for copying, distributing or modifying
00184 the Program or works based on it.
00185
00186 ****6.**** Each time you redistribute the Program (or any work based on the
00187 Program), the recipient automatically receives a license from the
00188 original licensor to copy, distribute or modify the Program subject to
00189 these terms and conditions. You may not impose any further
00190 restrictions on the recipients' exercise of the rights granted herein.
00191 You are not responsible for enforcing compliance by third parties to
00192 this License.
00193
00194 ****7.**** If, as a consequence of a court judgment or allegation of patent
00195 infringement or for any other reason (not limited to patent issues),
00196 conditions are imposed on you (whether by court order, agreement or
00197 otherwise) that contradict the conditions of this License, they do not

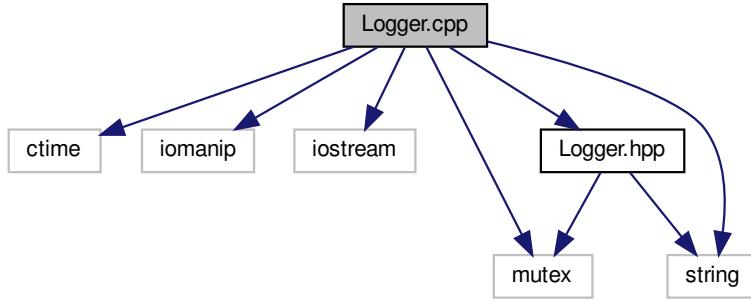
00198 excuse you from the conditions of this License. If you cannot
00199 distribute so as to satisfy simultaneously your obligations under this
00200 License and any other pertinent obligations, then as a consequence you
00201 may not distribute the Program at all. For example, if a patent
00202 license would not permit royalty-free redistribution of the Program by
00203 all those who receive copies directly or indirectly through you, then
00204 the only way you could satisfy both it and this License would be to
00205 refrain entirely from distribution of the Program.
00206
00207 If any portion of this section is held invalid or unenforceable under
00208 any particular circumstance, the balance of the section is intended to
00209 apply and the section as a whole is intended to apply in other
00210 circumstances.
00211
00212 It is not the purpose of this section to induce you to infringe any
00213 patents or other property right claims or to contest validity of any
00214 such claims; this section has the sole purpose of protecting the
00215 integrity of the free software distribution system, which is
00216 implemented by public license practices. Many people have made
00217 generous contributions to the wide range of software distributed
00218 through that system in reliance on consistent application of that
00219 system; it is up to the author/donor to decide if he or she is willing
00220 to distribute software through any other system and a licensee cannot
00221 impose that choice.
00222
00223 This section is intended to make thoroughly clear what is believed to
00224 be a consequence of the rest of this License.
00225
00226 ****8.**** If the distribution and/or use of the Program is restricted in
00227 certain countries either by patents or by copyrighted interfaces, the
00228 original copyright holder who places the Program under this License
00229 may add an explicit geographical distribution limitation excluding
00230 those countries, so that distribution is permitted only in or among
00231 countries not thus excluded. In such case, this License incorporates
00232 the limitation as if written in the body of this License.
00233
00234 ****9.**** The Free Software Foundation may publish revised and/or new versions
00235 of the General Public License from time to time. Such new versions will
00236 be similar in spirit to the present version, but may differ in detail to
00237 address new problems or concerns.
00238
00239 Each version is given a distinguishing version number. If the Program
00240 specifies a version number of this License which applies to it and "any
00241 later version", you have the option of following the terms and conditions
00242 either of that version or of any later version published by the Free
00243 Software Foundation. If the Program does not specify a version number of
00244 this License, you may choose any version ever published by the Free Software
00245 Foundation.
00246
00247 ****10.**** If you wish to incorporate parts of the Program into other free
00248 programs whose distribution conditions are different, write to the author
00249 to ask for permission. For software which is copyrighted by the Free
00250 Software Foundation, write to the Free Software Foundation; we sometimes
00251 make exceptions for this. Our decision will be guided by the two goals
00252 of preserving the free status of all derivatives of our free software and
00253 of promoting the sharing and reuse of software generally.
00254
00255 **## NO WARRANTY**
00256
00257 ****11.** BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY**
00258 **FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.** EXCEPT WHEN
00259 **OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES**
00260 **PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED**
00261 **OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF**
00262 **MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.** THE ENTIRE RISK AS
00263 **TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.** SHOULD THE
00264 **PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,**
00265 **REPAIR OR CORRECTION.**
00266
00267 ****12.** IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING**
00268 **WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR**
00269 **REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,**
00270 **INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING**
00271 **OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED**
00272 **TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY**
00273 **YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER**
00274 **PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE**
00275 **POSSIBILITY OF SUCH DAMAGES.**
00276
00277 **END OF TERMS AND CONDITIONS**
00278

```
00279 ### How to Apply These Terms to Your New Programs
00280
00281 If you develop a new program, and you want it to be of the greatest
00282 possible use to the public, the best way to achieve this is to make it
00283 free software which everyone can redistribute and change under these terms.
00284
00285 To do so, attach the following notices to the program. It is safest
00286 to attach them to the start of each source file to most effectively
00287 convey the exclusion of warranty; and each file should have at least
00288 the "copyright" line and a pointer to where the full notice is found.
00289
00290 <one line to give the program's name and a brief idea of what it does.>
00291 Copyright (C) <year> <name of author>
00292
00293 This program is free software; you can redistribute it and/or modify
00294 it under the terms of the GNU General Public License as published by
00295 the Free Software Foundation; either version 2 of the License, or
00296 (at your option) any later version.
00297
00298 This program is distributed in the hope that it will be useful,
00299 but WITHOUT ANY WARRANTY; without even the implied warranty of
00300 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00301 GNU General Public License for more details.
00302
00303 You should have received a copy of the GNU General Public License along
00304 with this program; if not, write to the Free Software Foundation, Inc.,
00305 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
00306
00307 Also add information on how to contact you by electronic and paper mail.
00308
00309 If the program is interactive, make it output a short notice like this
00310 when it starts in an interactive mode:
00311
00312 Gnomovision version 69, Copyright (C) year name of author
00313 Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
00314 This is free software, and you are welcome to redistribute it
00315 under certain conditions; type 'show c' for details.
00316
00317 The hypothetical commands 'show w' and 'show c' should show the appropriate
00318 parts of the General Public License. Of course, the commands you use may
00319 be called something other than 'show w' and 'show c'; they could even be
00320 mouse-clicks or menu items--whatever suits your program.
00321
00322 You should also get your employer (if you work as a programmer) or your
00323 school, if any, to sign a "copyright disclaimer" for the program, if
00324 necessary. Here is a sample; alter the names:
00325
00326 Yoyodyne, Inc., hereby disclaims all copyright interest in the program
00327 'Gnomovision' (which makes passes at compilers) written by James Hacker.
00328
00329 <signature of Ty Coon>, 1 April 1989
00330 Ty Coon, President of Vice
00331
00332 This General Public License does not permit incorporating your program into
00333 proprietary programs. If your program is a subroutine library, you may
00334 consider it more useful to permit linking proprietary applications with the
00335 library. If this is what you want to do, use the GNU Lesser General
00336 Public License instead of this License.
```

16.91 Logger.cpp File Reference

```
#include <ctime>
#include <iomanip>
#include <iostream>
#include <mutex>
#include <string>
#include "Logger.hpp"
```

Include dependency graph for Logger.cpp:



16.92 Logger.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <ctime>
00019 #include <iomanip>
00020 #include <iostream>
00021 #include <mutex>
00022 #include <string>
00023
00024 #include "Logger.hpp"
00025
00026
00027 /** Log a message with timestamp (at level 1).
00028 *
00029 * This will log a message with the current date and time as the timestamp if
00030 * the loglevel is set to at least 1.
00031 *
00032 * \note This is the only method (along with \ref log(unsigned int,
00033 * std::string)) that is threadsafe.
00034 *
00035 * \param message The message to log.
00036 */
00037 void Logger::log(std::string message)
00038 {
00039     Logger::log(1, std::move(message));
00040 }
00041
00042
00043 /** Log a message with timestamp at the given level.
00044 *
00045 * This will log a message with the current date and time as the timestamp if
00046 * the loglevel is at least \p level.
```

```

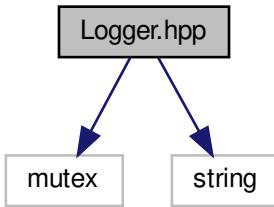
00047  *
00048  * \note This is the only method (along with \ref log(std::string)) that is
00049  *      threadsafe.
00050  *
00051  * \param level The level to log the \p message at. If the logger's level is
00052  *      lower than this, the message will be discarded. The valid range is 1 to
00053  *      65535, a value of 0 will be corrected to 1.
00054  * \param message The message to log.
00055  * \remarks
00056  *      Threadsafe (locking).
00057  */
00058 void Logger::log(unsigned int level, std::string message)
00059 {
00060     if (level < 1)
00061     {
00062         level = 1;
00063     }
00064
00065     if (level_ >= level)
00066     {
00067         auto t = std::time(nullptr);
00068         auto tm = *std::localtime(&t);
00069         std::lock_guard<std::mutex> lock(mutex_);
00070         std::cout << std::put_time(&tm, "%Y-%m-%d %H:%M:%S") << " "
00071             << message << std::endl;
00072     }
00073 }
00074
00075
00076 /** Set the logging level.
00077 */
00078 * A higher level indicates a higher verbosity of logging. A level of 0 will
00079 * completely disable logging.
00080 *
00081 * \param level The new logging level, valid values are 0 to 65535.
00082 */
00083 void Logger::level(unsigned int level)
00084 {
00085     level_ = level;
00086 }
00087
00088
00089 /** Get the logging level.
00090 */
00091 * \note It is recommended to check the level before constructing a log message
00092 *      if the message is expensive to construct.
00093 *
00094 * \returns The current logging level.
00095 */
00096 unsigned int Logger::level()
00097 {
00098     return level_;
00099 }
00100
00101
00102 unsigned int Logger::level_ = 0;
00103
00104
00105 #ifdef __clang__
00106     #pragma clang diagnostic push
00107     #pragma clang diagnostic ignored "-Wglobal-constructors"
00108     #pragma clang diagnostic ignored "-Wexit-time-destructors"
00109 #endif
00110
00111 std::mutex Logger::mutex_;
00112
00113 #ifdef __clang__
00114     #pragma clang diagnostic pop
00115 #endif

```

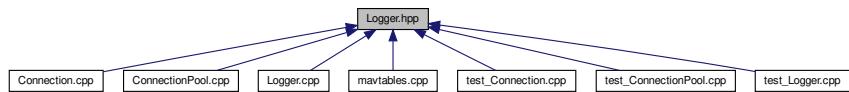
16.93 Logger.hpp File Reference

```
#include <mutex>
#include <string>
```

Include dependency graph for Logger.hpp:



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



Classes

- class [Logger](#)

16.94 Logger.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef LOGGER_HPP_
00019 #define LOGGER_HPP_
00020
00021
00022 #include <mutex>
00023 #include <string>
00024
00025
  
```

```

00026 /** A global static logger for use by all of mavtables.
00027  * 
00028  * \note Only supports writing to stdout.
00029  */
00030 class Logger
00031 {
00032     public:
00033         static void log(std::string message);
00034         static void log(unsigned int level, std::string message);
00035         static void level(unsigned int level);
00036         static unsigned int level();
00037
00038     private:
00039         Logger(const Logger &logger) = delete;
00040         void operator=(const Logger &logger) = delete;
00041         Logger() = default;
00042         static unsigned int level_;
00043         static std::mutex mutex_;
00044     };
00045
00046
00047 #endif // LOGGER_HPP_

```

16.95 logger.py File Reference

Namespaces

- `logger`

Functions

- def `logger.signal_handler` (signal, frame)
- def `logger.parse_args` ()
- def `logger.heartbeat` (mav)
- def `logger.start_heartbeats` (mav)
- def `logger.main` ()

16.96 logger.py

```

00001 #!/usr/bin/env python
00002
00003 from __future__ import (absolute_import, division,
00004                         print_function, unicode_literals)
00005 from builtins import *
00006
00007 import os
00008 import sys
00009 import signal
00010 from time import sleep
00011 from argparse import ArgumentParser
00012 from threading import Thread
00013 from pymavlink import mavutil
00014
00015
00016 def signal_handler(signal, frame):
00017     sys.exit(0)
00018
00019
00020 def parse_args():
00021     parser = ArgumentParser(description='Log incoming MAVLink packets.')
00022     parser.add_argument('system', type=int, help='system ID')
00023     parser.add_argument('component', type=int, help='component ID')

```

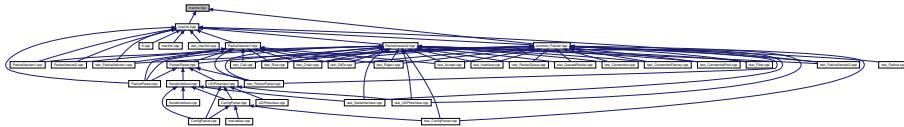
```

00024     parser.add_argument(
00025         '--verbose', '-v', action='store_true', help='enable verbosity')
00026     parser.add_argument(
00027         '--noheartbeat', action='store_true', help='disable heartbeat')
00028     parser.add_argument(
00029         '--udp', action='store', help='UDP address:port to connect to')
00030     parser.add_argument(
00031         '--serial', action='store', help='serial port device string')
00032     args = vars(parser.parse_args())
00033     if not args['udp'] and not args['serial']:
00034         print('expected --udp or --serial option')
00035         sys.exit()
00036     if args['udp'] and args['serial']:
00037         print('expected --udp or --serial option, but not both')
00038         sys.exit()
00039     return args
00040
00041
00042 def heartbeat(mav):
00043     while True:
00044         mav.mav.heartbeat_send(0, 0, 0, 0, 0)
00045         sleep(60)
00046
00047
00048 def start_heartbeats(mav):
00049     hbthread = Thread(target=heartbeat, args=(mav,))
00050     hbthread.daemon = True
00051     hbthread.start()
00052
00053
00054 def main():
00055     signal.signal(signal.SIGINT, signal_handler)
00056     os.environ['MAVLINK20'] = '1'
00057     mavutil.set_dialect('common')
00058     args = parse_args()
00059     if args['udp']:
00060         mav = mavutil.mavlink_connection('udpout:' + args['udp'],
00061                                         source_system=args['system'], source_component=args['component'])
00062     elif args['serial']:
00063         mav = mavutil.mavlink_connection(args['serial'], baud=57600,
00064                                         source_system=args['system'], source_component=args['component'])
00065     else:
00066         sys.exit(1)
00067     if not args['noheartbeat']:
00068         start_heartbeats(mav)
00069     while True:
00070         msg = mav.recv_match(blocking=True)
00071         if (args['verbose']):
00072             msg_string = "{:s} from {:d}.{:d}".format(
00073                 msg.get_type(), msg.get_srcSystem(), msg.get_srcComponent())
00074             try:
00075                 dest_string = " to {:d}.{:d}".format(
00076                     msg.target_system, msg.target_component)
00077                 msg_string += dest_string
00078             except:
00079                 pass
00080             print(msg_string)
00081         else:
00082             print(msg.get_type())
00083         sys.stdout.flush()
00084
00085
00086 if __name__ == '__main__':
00087     main()

```

16.97 macros.hpp File Reference

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



Macros

- `#define PACKED __attribute__((packed))`

16.98 macros.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 /** \defgroup macros Macros
00019 *
00020 * Macros used elsewhere.
00021 */
00022
00023
00024 /** Enforce a packed structure.
00025 *
00026 * A packed structure will not have any padding regardless of the typical
00027 * alignment restrictions.
00028 *
00029 * Example:
00030 * ``
00031 * struct PACKED a_packed_structure
00032 * {
00033 *     uint8_t a;
00034 *     uint16_t b;
00035 *     uint32_t c;
00036 * }
00037 * ``
00038 *
00039 * \ingroup macros
00040 */
00041 #define PACKED __attribute__((packed))

```

16.99 mainpage.md File Reference

16.100 mainpage.md

```

00001 mavtables {#mainpage}
00002 =====
00003
00004 * [Build Status](https://travis-ci.org/shamuproject/mavtables) ! [Build
    Status](https://travis-ci.org/shamuproject/mavtables.svg?branch=master)
00005 * [Coverage Status](https://coveralls.io/github/shamuproject/mavtables?branch=master) ! [Coverage
    Status](https://coveralls.io/repos/github/shamuproject/mavtables/badge.svg?branch=master)
00006
00007 ## Introduction
00008
00009 A MAVLink router and firewall. It can connect 2 or more MAVLink endpoints such
00010 as autopilots, ground control software, loggers, image capture systems, etc over
00011 serial and UDP. MAVLink packets will be routed to specific components when they
00012 have a destination address. Any packet, targeted or broadcasted, can be
00013 filtered based on source system/component, destination system/component and
00014 message type. The filter can also apply a priority to packets allowing more
00015 important packets to take priority over lower priority packets when an endpoint
00016 is choked.
00017
00018 * [User Manual] (\ref user_manual)
00019 * [Configuration] (\ref configuration)
00020 * [README] (\ref readme)
00021 * [LICENSE] (\ref license)
00022 * [CONTRIBUTING] (\ref contributing)
00023 * [Makefile Targets] (\ref makefile_targets)
00024
00025
00026 ## Links
00027
00028 * [HTML Documentation](https://shamuproject.github.io/mavtables)
00029 * [PDF Documentation](mavtables.pdf)
00030 * [Download](https://github.com/shamuproject/mavtables/archive/master.zip)
00031 * [GitHub](https://github.com/shamuproject/mavtables)
00032 * [Theory of MAVLink Routing](http://ardupilot.org/dev/docs/mavlink-routing-in-ardupilot.html)
00033
00034
00035 ## Install
00036
00037 In order to compile you will need the following packages:
00038
00039 * GCC 7+ or Clang 5+ (needed for C++17 support)
00040 * [CMake v3.3+](https://cmake.org/)
00041 * [Boost v1.54+](https://www.boost.org/)
00042
00043 'mavtables' can be easily installed using the standard procedure of
00044 ``
00045 $ make
00046 # make install
00047 ``
00048 The installation prefix is '/usr/local' by default but can be changed with
00049 ``
00050 $ make
00051 # make PREFIX=/desired/install/path install
00052 ``
00053
00054 See the [README] (\ref readme) for further documentation.

```

16.101 make_targets.md File Reference

16.102 make_targets.md

```

00001 Makefile Targets {#makefile_targets}
00002 =====

```

```
00003
00004 While mavtables uses [CMake] (https://cmake.org/) for building a simple Makefile
00005 (which calls 'cmake') is included for convenience. The available targets are
00006 listed below.
00007
00008
00009 ## release (default)
00010
00011 Build 'mavtables' using the _Release_ option which generates an optimized (for
00012 speed) executable without debug symbols. 'mavtables' is placed at
00013 'build/mavtables'.
00014
00015 Unit tests are built as well under 'build/unit_tests' and are built with
00016 optimizations.
00017
00018
00019 ## debug
00020
00021 Build 'mavtables' and unit tests using the _Debug_ option which will enable
00022 debug flags and turn off optimizations. The executables are located at
00023 'build/mavtables' and 'build/unit_tests'.
00024
00025
00026 ## test
00027
00028 Build and run all tests. Runs both 'test/unit_tests/run_tests.sh' and
00029 'test/integration_tests/run_tests.sh'. The latter requires Python with the
00030 packages located in 'test/integration_tests/requirements.txt'.
00031
00032
00033 ## unit_tests
00034
00035 Build and run unit tests only.
00036
00037
00038 ## integration_tests
00039
00040 Build and run integration tests only. Requires Python with the packages located
00041 in 'test/integration_tests/requirements.txt'.
00042
00043
00044 ## coverage
00045
00046 Build and run all tests and compute test coverage. This option requires
00047 [lcov] (http://ltp.sourceforge.net/coverage/lcov.php) and
00048 [gcovr] (http://gcovr.com/) to be installed and will only work when the compiler
00049 is 'g++'.
00050
00051 The coverage per file will be printed to the terminal and a detailed, line by
00052 line, report generated with [lcov] (http://ltp.sourceforge.net/coverage/lcov.php)
00053 at 'build/lcov/html/selected_targets/index.html'.
00054
00055
00056 ## linecheck
00057
00058 Prints all lines exceeding 80 characters.
00059
00060
00061 ## style
00062
00063 Fix the style of C++ source code and header files. The original files are
00064 backed up with a '.orig' extension. [Artistic
00065 Style] (http://astyle.sourceforge.net/) is required to use this target. This
00066 will also call the linecheck target.
00067
00068
00069 ## html
00070
00071 Generate html documentation with
00072 [Doxygen] (http://www.stack.nl/~dimitri/doxygen/) and place it at
00073 'build/doc/html/index.html'.
00074
00075
00076 ## doc
00077
00078 Generate html and pdf documentation with
00079 [Doxygen] (http://www.stack.nl/~dimitri/doxygen/) and place it at
00080 'build/doc/html/index.html' and 'build/doc/html/mavtables.pdf' respectively.
00081
00082
00083 ## gh-pages
```

```

00084
00085 Generate html and pdf documentation and publish to
00086 [https://shamuproject.github.io/mavtables](https://shamuproject.github.io/mavtables).
00087
00088
00089 ## clean
00090
00091 Clean up the project directory. This does not remove git submodules.
00092
00093
00094 ## remove-subs
00095
00096 Remove all git submodules. They will be re-downloaded if needed.

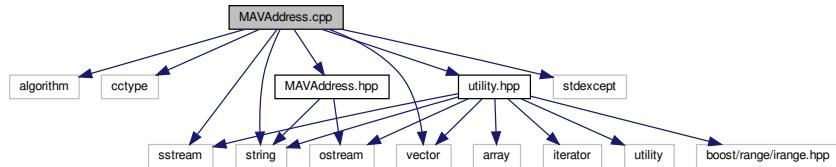
```

16.103 MAVAddress.cpp File Reference

```

#include <algorithm>
#include <cctype>
#include <sstream>
#include <stdexcept>
#include <string>
#include <vector>
#include "MAVAddress.hpp"
#include "utility.hpp"
Include dependency graph for MAVAddress.cpp:

```



16.104 MAVAddress.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <algorithm>
00019 #include <cctype>
00020 #include <sstream>
00021 #include <stdexcept>
00022 #include <string>

```

```
00023 #include <vector>
00024
00025 #include "MAVAddress.hpp"
00026 #include "utility.hpp"
00027
00028
00029 /** Construct a MAVLink address from an address in numeric representation.
00030 *
00031 * The numeric representation of a MAVLink address is two bytes, the MSB
00032 * contains the system ID and the LSB contains the component ID.
00033 *
00034 * \param address Address (0 - 65535) with system ID encoded in MSB and
00035 * component ID encoded in LSB.
00036 * \throws std::out_of_range if the address is not between 0 and 65535.
00037 */
00038 MAVAddress::MAVAddress(unsigned int address)
00039 {
00040     if (address > 65535)
00041     {
00042         throw std::out_of_range(
00043             "Address (" + std::to_string(address) +
00044             ") is outside of the allowed range (0 - 65535).");
00045     }
00046
00047     address_ = address;
00048 }
00049
00050
00051 /** Construct a MAVLink address from system and component ID's.
00052 *
00053 * \param system System ID (0 - 255).
00054 * \param component Component ID (0 - 255).
00055 * \throws std::out_of_range if either the \p system ID or the \p component ID
00056 * is out of range.
00057 */
00058 void MAVAddress::construct_(unsigned int system, unsigned int component)
00059 {
00060     if (system > 255)
00061     {
00062         throw std::out_of_range(
00063             "System ID (" + std::to_string(system) +
00064             ") is outside of the allowed range (0 - 255).");
00065     }
00066
00067     if (component > 255)
00068     {
00069         throw std::out_of_range(
00070             "Component ID (" + std::to_string(component) +
00071             ") is outside of the allowed range (0 - 255).");
00072     }
00073
00074     address_ = ((system << 8) & 0xFF00) | (component & 0x00FF);
00075 }
00076
00077
00078 /** Construct a MAVLink address from the system ID and component ID.
00079 *
00080 * \note component=0 and system=0 is the broadcast address.
00081 *
00082 * \param system System ID (0 - 255).
00083 * \param component Component ID (0 - 255).
00084 * \throws std::out_of_range if either the \p system ID or the \p component ID
00085 * is out of range.
00086 */
00087 MAVAddress::MAVAddress(unsigned int system, unsigned int component)
00088 {
00089     construct_(system, component);
00090 }
00091
00092
00093 /** Construct MAVLink address from a string.
00094 *
00095 * Parse a string of the form "<System ID>.<Component ID>".
00096 *
00097 * Some examples are:
00098 * - "0.0"
00099 * - "16.8"
00100 * - "128.4"
00101 *
00102 * \param address String representing the MAVLink address.
00103 * \throws std::invalid_argument if the address string does not represent a
```

```

00104 *      valid MAVLink address.
00105 * \throws std::out_of_range if either the \p system ID or the \p component ID
00106 *      is out of range.
00107 */
00108 MAVAddress::MAVAddress(std::string address)
00109 {
00110     // Check validity of address string.
00111     if (address.size() < 3 || !(isdigit(address.front())))
00112         || !isdigit(address.back()))
00113     {
00114         throw std::invalid_argument("Invalid MAVLink address string.");
00115     }
00116
00117     for (auto c : address)
00118     {
00119         if (!(c == '.' || isdigit(c)))
00120         {
00121             throw std::invalid_argument("Invalid MAVLink address string.");
00122         }
00123     }
00124
00125     // Read address string.
00126     std::replace(address.begin(), address.end(), '.', ' ');
00127     std::vector<unsigned int> octets;
00128     std::istringstream ss(address);
00129     unsigned int i;
00130
00131     while (ss >> i)
00132     {
00133         octets.push_back(i);
00134     }
00135
00136     // Check for correct number of octets.
00137     if (octets.size() != 2 || !ss.eof())
00138     {
00139         throw std::invalid_argument("Invalid MAVLink address string.");
00140     }
00141
00142     construct_(octets[0], octets[1]);
00143 }
00144
00145
00146 /** Return the MAVLink address in numeric form.
00147 *
00148 * \returns The MAVLink address as a two byte number with the system ID encoded
00149 *          in the MSB and the component ID in the LSB.
00150 */
00151 unsigned int MAVAddress::address() const
00152 {
00153     return address_;
00154 }
00155
00156
00157 /** Return the System ID.
00158 *
00159 * \returns The system ID (0 - 255).
00160 */
00161 unsigned int MAVAddress::system() const
00162 {
00163     return (address_ >> 8) & 0x00FF;
00164 }
00165
00166
00167 /** Return the Component ID.
00168 *
00169 * \returns The component ID (0 - 255).
00170 */
00171 unsigned int MAVAddress::component() const
00172 {
00173     return address_ & 0x00FF;
00174 }
00175
00176
00177 /** Equality comparison.
00178 *
00179 * \relates MAVAddress
00180 * \param lhs The left hand side MAVLink address.
00181 * \param rhs The right hand side MAVLink address.
00182 * \retval true if \p lhs and \p rhs have the same system and component ID's.
00183 * \retval false if \p lhs and \p rhs do not have the same system and component
00184 *          ID's.

```

```
00185  */
00186 bool operator==(const MAVAddress &lhs, const MAVAddress &rhs)
00187 {
00188     return lhs.address() == rhs.address();
00189 }
00190
00191
00192 /** Inequality comparison.
00193 *
00194 * \relates MAVAddress
00195 * \param lhs The left hand side MAVLink address.
00196 * \param rhs The right hand side MAVLink address.
00197 * \retval true if \p lhs and \p rhs do not have the same system and component
00198 *         ID's
00199 * \retval false if \p lhs and \p rhs have the same system and component ID's.
00200 */
00201 bool operator!=(const MAVAddress &lhs, const MAVAddress &rhs)
00202 {
00203     return lhs.address() != rhs.address();
00204 }
00205
00206
00207 /** Less than comparison.
00208 *
00209 * \note The System ID is considered first followed by the Component ID.
00210 *
00211 * \relates MAVAddress
00212 * \param lhs The left hand side MAVLink address.
00213 * \param rhs The right hand side MAVLink address.
00214 * \retval true if \p lhs is less than \p rhs.
00215 * \retval false if \p lhs is not less than \p rhs.
00216 */
00217 bool operator<(const MAVAddress &lhs, const MAVAddress &rhs)
00218 {
00219     return lhs.address() < rhs.address();
00220 }
00221
00222
00223 /** Greater than comparison.
00224 *
00225 * \note The System ID is considered first followed by the Component ID.
00226 *
00227 * \relates MAVAddress
00228 * \param lhs The left hand side MAVLink address.
00229 * \param rhs The right hand side IP address.
00230 * \retval true if \p lhs is greater than \p rhs.
00231 * \retval false if \p lhs is not greater than \p rhs.
00232 */
00233 bool operator>(const MAVAddress &lhs, const MAVAddress &rhs)
00234 {
00235     return lhs.address() > rhs.address();
00236 }
00237
00238
00239 /** Less than or equal comparison.
00240 *
00241 * \note The System ID is considered first followed by the Component ID.
00242 *
00243 * \relates MAVAddress
00244 * \param lhs The left hand side IP address.
00245 * \param rhs The right hand side IP address.
00246 * \retval true if \p lhs is less than or equal to \p rhs.
00247 * \retval false if \p lhs is greater than \p rhs.
00248 */
00249 bool operator<=(const MAVAddress &lhs, const MAVAddress &rhs)
00250 {
00251     return lhs.address() <= rhs.address();
00252 }
00253
00254
00255 /** Greater than comparison.
00256 *
00257 * \note The System ID is considered first followed by the Component ID.
00258 *
00259 * \relates MAVAddress
00260 * \param lhs The left hand side IP address.
00261 * \param rhs The right hand side IP address.
00262 * \retval true if \p lhs is greater than or equal to \p rhs.
00263 * \retval false if \p lhs is less than \p rhs.
00264 */
00265 bool operator>=(const MAVAddress &lhs, const MAVAddress &rhs)
```

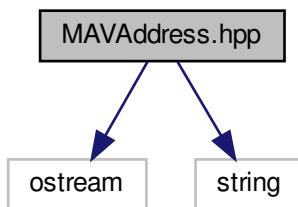
```

00266 {
00267     return lhs.address() >= rhs.address();
00268 }
00269
00270
00271 /** Print the MAVLink address to the given output stream.
00272 *
00273 * The format is "<System ID>.<Component ID>".
00274 *
00275 * Some examples are:
00276 *   - '0.0'
00277 *   - '16.8'
00278 *   - '128.4'
00279 *
00280 * \note The string constructor \ref MAVAddress(std::string) and the output
00281 * stream operator are inverses and thus:
00282 *
00283 * std::string addr = "192.168"
00284 * str(MAVAddress(addr)) == addr
00285 *
00286 *
00287 * \relates MAVAddress
00288 * \param os The output stream to print to.
00289 * \param mavaddress The MAVLink address to print.
00290 * \returns The output stream.
00291 */
00292 std::ostream &operator<<(std::ostream &os, const MAVAddress &mavaddress)
00293 {
00294     os << mavaddress.system() << "." << mavaddress.component();
00295     return os;
00296 }

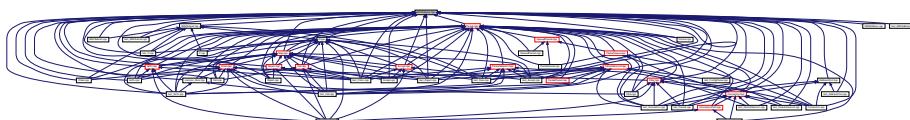
```

16.105 MAVAddress.hpp File Reference

```
#include <iostream>
#include <string>
Include dependency graph for MAVAddress.hpp:
```



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



Classes

- class [MAVAddress](#)

16.105.1 Function Documentation

16.105.1.1 operator"!=()

```
bool operator!= (
    const MAVAddress & lhs,
    const MAVAddress & rhs ) [related]
```

16.105.1.2 operator<()

```
bool operator< (
    const MAVAddress & lhs,
    const MAVAddress & rhs ) [related]
```

16.105.1.3 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const MAVAddress & mavaddress ) [related]
```

16.105.1.4 operator<=()

```
bool operator<= (
    const MAVAddress & lhs,
    const MAVAddress & rhs ) [related]
```

16.105.1.5 operator==()

```
bool operator== (
    const MAVAddress & lhs,
    const MAVAddress & rhs ) [related]
```

16.105.1.6 operator>()

```
bool operator> (
    const MAVAddress & lhs,
    const MAVAddress & rhs ) [related]
```

16.105.1.7 operator>=()

```
bool operator>= (
    const MAVAddress & lhs,
    const MAVAddress & rhs ) [related]
```

16.106 MAVAddress.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef MAVADDRESS_HPP_
00019 #define MAVADDRESS_HPP_
00020
00021
00022 #include <iostream>
00023 #include <string>
00024
00025
00026 /** A MAVLink address.
00027 *
00028 * MAVLink addresses consist of a system and component and can be represented
00029 * as two octets in the form:
00030 * ``
00031 * system.component
00032 * ``
00033 * Therefore, a system ID of 16 and a component ID of 8 can be represented as
00034 * '16.8'.
00035 *
```

```

00036     * '0.0' is reserved as the broadcast address.
00037     */
00038 class MAVAddress
00039 {
00040     public:
00041         /** Copy constructor.
00042         *
00043         * \param other MAVLink address to copy from.
00044         */
00045     MAVAddress(const MAVAddress &other) = default;
00046     /** Move constructor.
00047     *
00048     * \param other MAVLink address to move from.
00049     */
00050     MAVAddress(MAVAddress &&other) = default;
00051     MAVAddress(unsigned int address);
00052     MAVAddress(unsigned int system, unsigned int component);
00053     MAVAddress(std::string address);
00054     unsigned int address() const;
00055     unsigned int system() const;
00056     unsigned int component() const;
00057     /** Assignment operator.
00058     *
00059     * \param other MAVLink address to copy from.
00060     */
00061     MAVAddress &operator=(const MAVAddress &other) = default;
00062     /** Assignment operator (by move semantics).
00063     *
00064     * \param other MAVLink address to move from.
00065     */
00066     MAVAddress &operator=(MAVAddress &&other) = default;
00067
00068     private:
00069         unsigned int address_;
00070         void construct_(unsigned int system, unsigned int component);
00071 };
00072
00073
00074 bool operator==(const MAVAddress &lhs, const MAVAddress &rhs);
00075 bool operator!=(const MAVAddress &lhs, const MAVAddress &rhs);
00076 bool operator<(const MAVAddress &lhs, const MAVAddress &rhs);
00077 bool operator>(const MAVAddress &lhs, const MAVAddress &rhs);
00078 bool operator<=(const MAVAddress &lhs, const MAVAddress &rhs);
00079 bool operator>=(const MAVAddress &lhs, const MAVAddress &rhs);
00080 std::ostream &operator<<(std::ostream &os, const MAVAddress &mavaddress);
00081
00082
00083 #endif // MAVADDRESS_HPP_

```

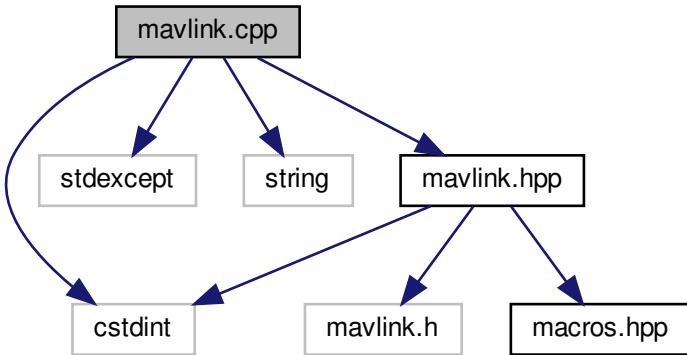
16.107 mavlink.cpp File Reference

```

#include <cstdint>
#include <stdexcept>
#include <string>
#include "mavlink.hpp"

```

Include dependency graph for mavlink.cpp:



Namespaces

- `mavlink`

Functions

- `std::string mavlink::name (unsigned long id)`
- `unsigned long mavlink::id (std::string name)`

16.108 mavlink.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <cstdint>
00019 #include <stdexcept>
00020 #include <string>
00021
00022 #include "mavlink.hpp"
00023
00024

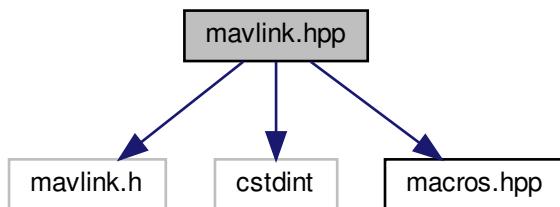
```

```

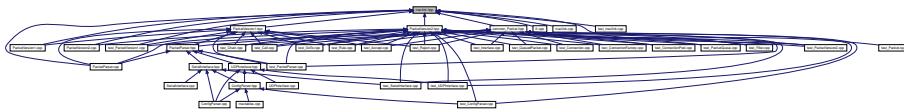
00025 namespace mavlink
00026 {
00027
00028     /** Get message name from numeric ID.
00029     *
00030     * \ingroup mavlink
00031     * \param id The ID of the MAVLink message to get the name of.
00032     * \returns The name of the message.
00033     * \throws std::invalid_argument if the given \p id is not valid.
00034     */
00035     std::string name(unsigned long id)
00036     {
00037         const mavlink_message_info_t *info =
00038             mavlink_get_message_info_by_id(static_cast<uint32_t>(id));
00039
00040         if (info)
00041         {
00042             return std::string(info->name);
00043         }
00044
00045         throw std::invalid_argument(
00046             "Invalid packet ID (" + std::to_string(id) + ")");
00047     }
00048
00049
00050     /** Get message ID from message name.
00051     *
00052     * \ingroup mavlink
00053     * \param name The name of the MAVLink message to get the numeric ID of.
00054     * \returns The numeric ID of the message.
00055     * \throws std::invalid_argument if the given message \p name is not valid.
00056     */
00057     unsigned long id(std::string name)
00058     {
00059         const mavlink_message_info_t *info =
00060             mavlink_get_message_info_by_name(name.c_str());
00061
00062         if (info)
00063         {
00064             return info->msgid;
00065         }
00066
00067         throw std::invalid_argument("Invalid packet name (" + name + ")");
00068     }
00069 }
00070 }
```

16.109 mavlink.hpp File Reference

```
#include "mavlink.h"
#include <cstdint>
#include "macros.hpp"
Include dependency graph for mavlink.hpp:
```



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



Classes

- struct [mavlink::v1_header](#)
- struct [mavlink::v2_header](#)

Namespaces

- [mavlink](#)

Macros

- `#define MAVLINK_USE_MESSAGE_INFO`

Functions

- std::string [mavlink::name](#) (unsigned long id)
- unsigned long [mavlink::id](#) (std::string name)

16.109.1 Macro Definition Documentation

16.109.1.1 MAVLINK_USE_MESSAGE_INFO

```
#define MAVLINK_USE_MESSAGE_INFO
```

Definition at line [25](#) of file [mavlink.hpp](#).

16.110 mavlink.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef MAVLINK_H_
00019 #define MAVLINK_H_
00020
00021
00022 extern "C"
00023 {
00024
00025 #define MAVLINK_USE_MESSAGE_INFO
00026 #include "mavlink.h"
00027
00028 }
00029
00030 #include <cstdint>
00031 #include "macros.hpp"
00032
00033 namespace mavlink
00034 {
00035
00036     /** \defgroup mavlink MAVLink Library and Helpers
00037     *
00038     * MAVLink utility macros, defines, types and functions.
00039     *
00040     * Including this file also includes the main MAVLink C library. This will
00041     * be the reference implementation using the ArduPilot dialect by default.
00042     *
00043     * The dialect can be changed by setting the 'DIALECT' environment
00044     * variable. The default is 'ardupilotmega'.
00045     *
00046     * The MAVLink library can be changed with the 'MDIR' environement variable
00047     * which should point to the directory where the 'DIALECT' directory can be
00048     * found.
00049     */
00050
00051
00052     /** MAVLink packet v1.0 header.
00053     *
00054     * \ingroup mavlink
00055     */
00056     struct PACKED v1_header
00057     {
00058         uint8_t magic;    //!< Protocol magic marker (0xFE).
00059         uint8_t len;      //!< Length of payload.
00060         uint8_t seq;      //!< Sequence of packet.
00061         uint8_t sysid;   //!< ID of message sender system/aircraft.
00062         uint8_t compid;  //!< ID of the message sneder component.
00063         uint8_t msgid;   //!< ID of message in payload.
00064     };
00065
00066
00067     /** MAVLink packet v2.0 header.
00068     *
00069     * \ingroup mavlink
00070     */
00071     struct PACKED v2_header
00072     {
00073         uint8_t magic;      //!< Protocol magic marker (0xFD).
00074         uint8_t len;        //!< Length of payload.
00075         uint8_t incompat_flags; //!< Flags that must be understood.
00076         uint8_t compat_flags; //!< Flags that can be ignored if not known.
00077         uint8_t seq;        //!< Sequence of packet.
```

```

00078     uint8_t sysid;           //!< ID of message sender system/aircraft.
00079     uint8_t compid;          //!< ID of the message sender component.
00080     uint32_t msgid : 24;    //!< ID of message in payload (3 bytes).
00081 };
00082
00083
00084     std::string name(unsigned long id);
00085     unsigned long id(std::string name);
00086
00087 }
00088
00089 #endif // MAVLINK_H_

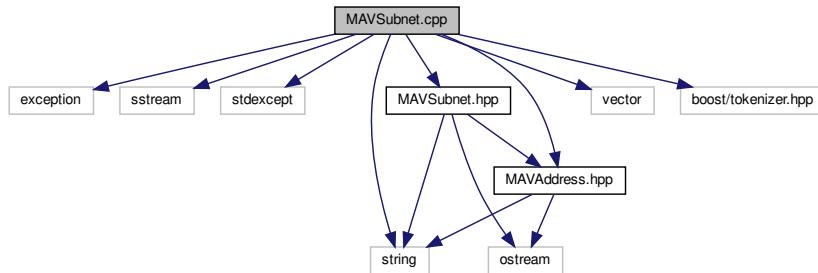
```

16.111 MAVSubnet.cpp File Reference

```

#include <exception>
#include <sstream>
#include <stdexcept>
#include <string>
#include <vector>
#include <boost/tokenizer.hpp>
#include "MAVAddress.hpp"
#include "MAVSubnet.hpp"
Include dependency graph for MAVSubnet.cpp:

```



16.112 MAVSubnet.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017

```

```
00018 #include <exception>
00019 #include <sstream>
00020 #include <stdexcept>
00021 #include <string>
00022 #include <vector>
00023
00024 #include <boost/tokenizer.hpp>
00025
00026 #include "MAVAddress.hpp"
00027 #include "MAVSubnet.hpp"
00028
00029
00030 /** Construct a MAVLink subnet from a MAVLink address and mask.
00031 *
00032 * \param address MAVLink address of subnet.
00033 * \param mask Two byte subnet mask, where the system mask is in the MSB and
00034 *     the component mask is in the LSB.
00035 * \throws std::out_of_range if the mask is not between 0x0000 and 0xFFFF.
00036 * \sa Check if a \ref MAVAddress is within the subnet with \ref contains.
00037 */
00038 MAVSubnet::MAVSubnet(const MAVAddress &address, unsigned int mask)
00039     : address_(address)
00040 {
00041     // Ensure mask is withing range.
00042     if (mask > 0xFFFF)
00043     {
00044         std::ostringstream ss;
00045         ss << "mask (0x"
00046             << std::uppercase << std::hex << mask << std::nouppercase
00047             << ") is outside of the allowed range (0x0000 - 0xFFFF).";
00048         throw std::out_of_range(ss.str());
00049     }
00050
00051     mask_ = mask;
00052 }
00053
00054
00055 /** Construct MAVLink subnet from an address, system mask, and component mask.
00056 *
00057 * \param address MAVLink address of subnet.
00058 * \param system_mask One byte subnet system mask with the bits set that must
00059 *     match the subnet address for a MAVLink address to be contained within
00060 *     the subnet. Valid range is 0 to 255.
00061 * \param component_mask One byte subnet component mask with the bits set that
00062 *     must match the subnet address for a MAVLink address to be contained
00063 *     within the subnet. Valid range is 0 to 255.
00064 * \throws std::out_of_range if the system and component masks are not each
00065 *     between 0x00 and 0xFF.
00066 * \sa Check if a \ref MAVAddress is within the subnet with \ref contains.
00067 */
00068 MAVSubnet::MAVSubnet(const MAVAddress &address, unsigned int system_mask,
00069                         unsigned int component_mask)
00070     : address_(address)
00071 {
00072     // Ensure system mask is withing range.
00073     if (system_mask > 0xFF)
00074     {
00075         std::ostringstream ss;
00076         ss << "System mask (0x"
00077             << std::uppercase << std::hex << system_mask << std::nouppercase
00078             << ") is outside of the allowed range (0x00 - 0xFF).";
00079         throw std::out_of_range(ss.str());
00080     }
00081
00082     // Ensure component mask is withing range.
00083     if (component_mask > 0xFF)
00084     {
00085         std::ostringstream ss;
00086         ss << "Component mask (0x" << std::hex << component_mask <<
00087             " ) is outside of the allowed range (0x00 - 0xFF).";
00088         throw std::out_of_range(ss.str());
00089     }
00090
00091     mask_ = ((system_mask << 8) & 0xFF00) | (component_mask & 0x00FF);
00092 }
00093
00094
00095 /** Construct a MAVLink subnet from a string.
00096 *
00097 * There are four string forms of MAVLink subnets:
00098 *
```

```

00099 *      1. "<System ID>.<Component ID>:<System ID mask>.<Component ID mask>"  
00100 *      2. "<System ID>.<Component ID>/<bits>"  
00101 *      3. "<System ID>.<Component ID>\<bits>"  
00102 *      4. "<System ID>.<Component ID>"  
00103 *  
00104 * The first form is self explanatory, but the 2nd and 3rd are not as simple.  
00105 * In the 2nd case the number of bits (0 - 16) is the number of bits from the  
00106 * left that must match for an address to be in the subnet. The 3rd form is  
00107 * like the 2nd, but does not require the system ID (first octet) to match and  
00108 * starts with the number of bits of the component ID (0 - 8) that must match  
00109 * from the left for an address to be in the subnet. The last form is  
00110 * shorthand for "<System ID>.<Component ID>/16", that is an exact match.  
00111 *  
00112 * Below is a table relating the slash postfix to the subnet mask in \<System  
00113 * mask\>.\<Component mask\> notation.  
00114 *  
00115 * | Mask with '\` | Mask with '\` | Postfix (\#bits) |  
00116 * | -----:| -----:| -----:|  
00117 * | 255.255 | out of range | 16 |  
00118 * | 255.254 | out of range | 15 |  
00119 * | 255.252 | out of range | 14 |  
00120 * | 255.248 | out of range | 13 |  
00121 * | 255.240 | out of range | 12 |  
00122 * | 255.224 | out of range | 11 |  
00123 * | 255.192 | out of range | 10 |  
00124 * | 255.128 | out of range | 9 |  
00125 * | 255.0 | 0.255 | 8 |  
00126 * | 254.0 | 0.254 | 7 |  
00127 * | 252.0 | 0.252 | 6 |  
00128 * | 248.0 | 0.248 | 5 |  
00129 * | 240.0 | 0.240 | 4 |  
00130 * | 224.0 | 0.224 | 3 |  
00131 * | 192.0 | 0.192 | 2 |  
00132 * | 128.0 | 0.128 | 1 |  
00133 * | 0.0 | 0 | 0 |  
00134 *  
00135 * Some examples are:  
00136 *  
00137 * - "128.0/8" - Matches addresses with system ID 128 and any component ID.  
00138 * - "128.0/9" - Matches addresses with system ID 128 and components ID's of  
00139 * 127 or less.  
00140 * - "128.255/9" - Matches addresses with system ID 128 and components ID's  
00141 * of 128 or more.  
00142 * - "128.0\1" - Matches addresses any system ID and components ID's of 127  
00143 * or less.  
00144 * - "128.255\1" - Matches addresses any system ID and components ID's of 128  
00145 * or more.  
00146 * - "255.0:128.240" - Matches system ID's 128 or greater and component ID's  
00147 * from 0 to 15.  
00148 * - "255.16:128.240" - Matches system ID's 128 or greater and component ID's  
00149 * from 16 to 31.  
00150 * - "255.16" - Matches only the address with system ID 255 and component ID  
00151 * 16.  
00152 *  
00153 * \param subnet String representing the MAVLink subnet.  
00154 * \throws std::out_of_range if either the system ID or the component ID is out  
00155 * of range.  
00156 * \throws std::out_of_range if the mask or slash bits are out of range.  
00157 * \throws std::invalid_argument if the subnet string does not represent a  
00158 * valid MAVLink subnet.  
00159 * \sa Check if a \ref MAVAddress is within the subnet with \ref contains.  
00160 */  
00161 MAVSubnet::MAVSubnet(std::string subnet)  
00162     : address_(0)  
00163 {  
00164     // If only an address was given (exact match subnet).  
00165     try  
00166     {  
00167         address_ = MAVAddress(subnet);  
00168         mask_ = 0xFFFF;  
00169         return;  
00170     }  
00171     catch (std::exception)  
00172     {  
00173         // Continue on parsing normally.  
00174     }  
00175  
00176     // Extract parts of subnet string.  
00177     std::vector<std::string> parts;  
00178     boost::char_separator<char> sep("", ":/\\"");  
00179     boost::tokenizer<boost::char_separator<char>> tokens(subnet, sep);

```

```
00180
00181     for (auto i : tokens)
00182     {
00183         parts.push_back(i);
00184     }
00185
00186     // Ensure proper format.
00187     if (parts.size() != 3)
00188     {
00189         throw std::invalid_argument(
00190             "Invalid MAVLink subnet: \\" + subnet + "\\".");
00191     }
00192
00193     address_ = MAVAddress(parts[0]);
00194     // Determine format of subnet string.
00195     int slashmask;
00196
00197     // If a regular subnet was given.
00198     switch (parts[1].at(0))
00199     {
00200         // Mask based subnet.
00201         case ':':
00202             try
00203             {
00204                 mask_ = MAVAddress(parts[2]).address();
00205             }
00206             catch (std::invalid_argument)
00207             {
00208                 throw std::invalid_argument(
00209                     "Invalid MAVLink subnet: \\" + subnet + "\\".");
00210             }
00211
00212             break;
00213
00214         // Forward slash based subnet (bits from left).
00215         case '/':
00216             std::istringstream(parts.at(2)) >> slashmask;
00217
00218             if (slashmask < 0 || slashmask > 16)
00219             {
00220                 throw std::out_of_range(
00221                     "Forward slash mask (" + std::to_string(slashmask)
00222                     + ") is outside of allowed range (0 - 16).");
00223             }
00224
00225             mask_ = (0xFFFF << (16 - slashmask)) & 0xFFFF;
00226             break;
00227
00228         // Backward slash based subnet (bits from left of component).
00229         case '\\':
00230             std::istringstream(parts.at(2)) >> slashmask;
00231
00232             if (slashmask < 0 || slashmask > 8)
00233             {
00234                 throw std::out_of_range(
00235                     "Backslash mask (" + std::to_string(slashmask)
00236                     + ") is outside of allowed range (0 - 8).");
00237             }
00238
00239             mask_ = (0xFFFF << (8 - slashmask)) & 0x00FF;
00240             break;
00241     }
00242 }
00243
00244
00245 /** Determine whether or not the subnet contains a given MAVLink address.
00246 */
00247 * \param address The MAVLink address to test.
00248 * \retval true if \p address is part of the subnet.
00249 */
00250 bool MAVSubnet::contains(const MAVAddress &address) const
00251 {
00252     return (address.address() & mask_) == (address_.address() & mask_);
00253 }
00254
00255
00256 /** Equality comparison.
00257 */
00258 * Compares both address and mask.
00259 *
00260 * \relates MAVSubnet
```

```

00261 * \param lhs The left hand side MAVLink subnet.
00262 * \param rhs The right hand side MAVLink subnet.
00263 * \retval true if \p lhs and \p rhs are the same.
00264 * \retval false if \p lhs and \p rhs are not the same.
00265 */
00266 bool operator==(const MAVSubnet &lhs, const MAVSubnet &rhs)
00267 {
00268     return (lhs.address_ == rhs.address_) && (lhs.mask_ == rhs.mask_);
00269 }
00270
00271
00272 /** Inequality comparison.
00273 *
00274 * Compares both address and mask.
00275 *
00276 * \relates MAVSubnet
00277 * \param lhs The left hand side MAVLink subnet.
00278 * \param rhs The right hand side MAVLink subnet.
00279 * \retval true if \p lhs and \p rhs are not the same.
00280 * \retval false if \p lhs and \p rhs are the same.
00281 */
00282 bool operator!=(const MAVSubnet &lhs, const MAVSubnet &rhs)
00283 {
00284     return (lhs.address_ != rhs.address_) || (lhs.mask_ != rhs.mask_);
00285 }
00286
00287
00288 /** Print the MAVLink subnet to the given output stream.
00289 *
00290 * There are three string forms of MAVLink subnets.
00291 *
00292 * 1. "<System ID>.<Component ID>:<System ID mask>.<Component ID mask>"
00293 * 2. "<System ID>.<Component ID>/<bits>"
00294 * 3. "<System ID>.<Component ID>\<bits>"
00295 * 4. "<System ID>.<Component ID>"
00296 *
00297 * The slash notation is preferred. The last form is used when the mask
00298 * requires all bits of a subnet to match an address.
00299 *
00300 * See \ref MAVSubnet::MAVSubnet(std::string address) for more information on
00301 * the string format.
00302 *
00303 * \note The string constructor \ref MAVSubnet(std::string) and the output
00304 * stream operator are not inverses because of form preferences:
00305 *
00306 * \relates MAVSubnet
00307 * \param os The output stream to print to.
00308 * \param mavsubnet The MAVLink subnet to print.
00309 * \returns The output stream.
00310 */
00311 std::ostream &operator<<(std::ostream &os, const MAVSubnet &mavsubnet)
00312 {
00313     os << mavsubnet.address_;
00314
00315     switch (mavsubnet.mask_)
00316     {
00317         case 0b1111111111111111:
00318             // return os << "/16"
00319             // Represent exact masks as an address.
00320             return os;
00321
00322         case 0b1111111111111110:
00323             return os << "/15";
00324
00325         case 0b1111111111111100:
00326             return os << "/14";
00327
00328         case 0b1111111111110000:
00329             return os << "/13";
00330
00331         case 0b1111111111100000:
00332             return os << "/12";
00333
00334         case 0b1111111111000000:
00335             return os << "/11";
00336
00337         case 0b1111111110000000:
00338             return os << "/10";
00339
00340         case 0b1111111100000000:
00341             return os << "/9";

```

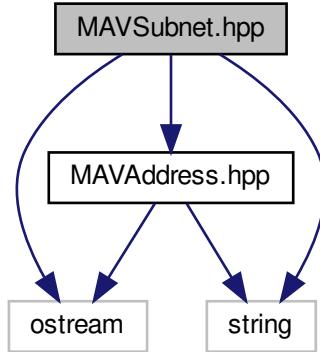
```

00342
00343     case 0b1111111000000000:
00344         return os << "/8";
00345
00346     case 0b1111110000000000:
00347         return os << "/7";
00348
00349     case 0b1111110000000000:
00350         return os << "/6";
00351
00352     case 0b1111100000000000:
00353         return os << "/5";
00354
00355     case 0b1111000000000000:
00356         return os << "/4";
00357
00358     case 0b1110000000000000:
00359         return os << "/3";
00360
00361     case 0b1100000000000000:
00362         return os << "/2";
00363
00364     case 0b1000000000000000:
00365         return os << "/1";
00366
00367     case 0b0000000000000000:
00368         return os << "/0";
00369
00370     case 0b0000000011111111:
00371         return os << "\\8";
00372
00373     case 0b0000000011111110:
00374         return os << "\\7";
00375
00376     case 0b0000000011111100:
00377         return os << "\\6";
00378
00379     case 0b0000000011111000:
00380         return os << "\\5";
00381
00382     case 0b0000000011110000:
00383         return os << "\\4";
00384
00385     case 0b0000000011100000:
00386         return os << "\\3";
00387
00388     case 0b0000000011000000:
00389         return os << "\\2";
00390
00391     case 0b0000000010000000:
00392         return os << "\\1";
00393
00394     default:
00395         return os << ":" << MAVAddress(mavsubnet.mask_);
00396     }
00397 }
```

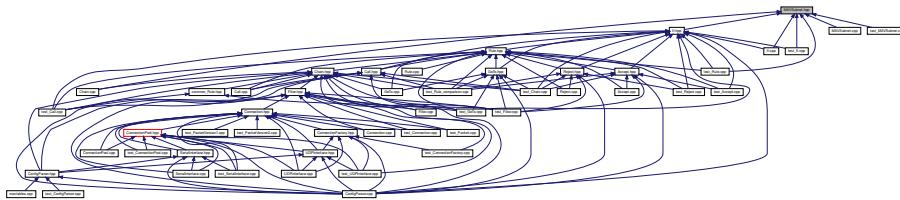
16.113 MAVSubnet.hpp File Reference

```
#include <iostream>
#include <string>
#include "MAVAddress.hpp"
```

Include dependency graph for MAVSubnet.hpp:



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



Classes

- class [MAVSubnet](#)

Functions

- bool [operator==](#) (const [MAVSubnet](#) &lhs, const [MAVSubnet](#) &rhs)
- bool [operator!=](#) (const [MAVSubnet](#) &lhs, const [MAVSubnet](#) &rhs)
- std::ostream & [operator<<](#) (std::ostream &os, const [MAVSubnet](#) &mavsubnet)

16.113.1 Function Documentation

16.113.1.1 operator"!=()

```
bool operator!= (
    const MAVSubnet & lhs,
    const MAVSubnet & rhs )
```

16.113.1.2 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const MAVSubnet & mavsubnet )
```

16.113.1.3 operator==()

```
bool operator== (
    const MAVSubnet & lhs,
    const MAVSubnet & rhs )
```

16.114 MAVSubnet.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef MAVSUBNET_HPP_
00019 #define MAVSUBNET_HPP_
00020
00021
00022 #include <iostream>
00023 #include <string>
00024
00025 #include "MAVAddress.hpp"
00026
00027
00028 /** A MAVLink subnet.
00029 *
00030 * Mavlink subnets work the same as IP subnets and allow the definition of a
00031 * range of addresses. This is used to allow a single firewall rule to match
00032 * multiple addresses.
00033 *
00034 * \sa MAVAddress
00035 */
```

```

00036 class MAVSubnet
00037 {
00038     public:
00039         /** Copy constructor.
00040         *
00041         * \param other MAVLink subnet to copy from.
00042         */
00043     MAVSubnet(const MAVSubnet &other) = default;
00044     /** Move constructor.
00045     *
00046     * \param other MAVLink subnet to move from.
00047     */
00048     MAVSubnet(MAVSubnet &&other) = default;
00049     MAVSubnet(const MAVAddress &address, unsigned int mask = 0xFFFF);
00050     MAVSubnet(const MAVAddress &address, unsigned int system_mask,
00051                 unsigned int component_mask);
00052     MAVSubnet(std::string address);
00053     bool contains(const MAVAddress &address) const;
00054     /** Assignment operator.
00055     *
00056     * \param other MAVLink subnet to copy from.
00057     */
00058     MAVSubnet &operator=(const MAVSubnet &other) = default;
00059     /** Assignment operator (by move semantics).
00060     *
00061     * \param other MAVLink subnet to move from.
00062     */
00063     MAVSubnet &operator=(MAVSubnet &&other) = default;
00064
00065     friend bool operator==(const MAVSubnet &lhs, const
00066                             MAVSubnet &rhs);
00066     friend bool operator!=(const MAVSubnet &lhs, const
00067                             MAVSubnet &rhs);
00067     friend std::ostream &operator<<(std::ostream &os,
00068                                         const MAVSubnet &mavsubnet);
00069
00070     private:
00071         MAVAddress address_;
00072         unsigned int mask_;
00073     };
00074
00075
00076     bool operator==(const MAVSubnet &lhs, const MAVSubnet &rhs);
00077     bool operator!=(const MAVSubnet &lhs, const MAVSubnet &rhs);
00078     std::ostream &operator<<(std::ostream &os, const MAVSubnet &mavsubnet);
00079
00080
00081 #endif // MAVSUBNET_HPP_

```

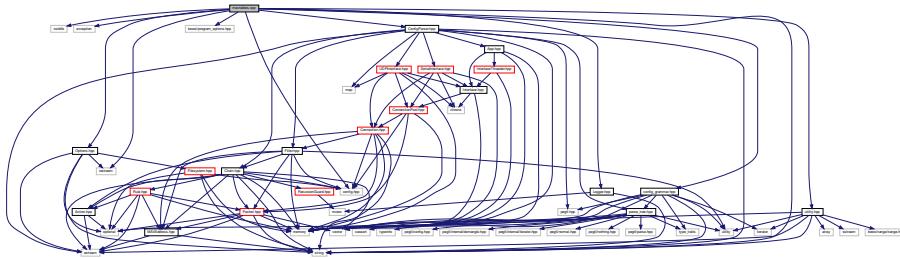
16.115 mavtables.cpp File Reference

```

#include <cstdlib>
#include <exception>
#include <iostream>
#include <string>
#include <boost/program_options.hpp>
#include "config.hpp"
#include "config_grammar.hpp"
#include "ConfigParser.hpp"
#include "Logger.hpp"
#include "Options.hpp"
#include "utility.hpp"

```

Include dependency graph for mavtables.cpp:



Functions

- int [main](#) (int argc, const char *argv[])

16.115.1 Function Documentation

16.115.1.1 [main\(\)](#)

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

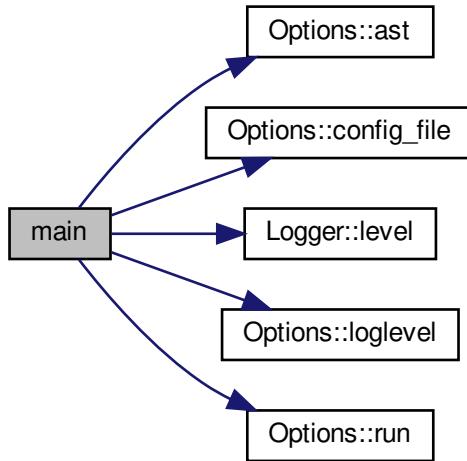
mavtables: A MAVLink router and firewall.

See the README for further documentation.

Definition at line [37](#) of file [mavtables.cpp](#).

References [Options::ast\(\)](#), [Options::config_file\(\)](#), [Logger::level\(\)](#), [Options::loglevel\(\)](#), and [Options::run\(\)](#).

Here is the call graph for this function:



16.116 mavtables.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <cstdlib>
00019 #include <exception>
00020 #include <iostream>
00021 #include <string>
00022
00023 #include <boost/program_options.hpp>
00024
00025 #include "config.hpp"
00026 #include "config_grammar.hpp"
00027 #include "ConfigParser.hpp"
00028 #include "Logger.hpp"
00029 #include "Options.hpp"
00030 #include "utility.hpp"
00031
00032
00033 /** mavtables: A MAVLink router and firewall.
00034 *
00035 * See the README for further documentation.
00036 */
00037 int main(int argc, const char *argv[])
  
```

```

00038 {
00039     try
00040     {
00041         Options options(argc, argv);
00042
00043         if (options)
00044         {
00045             auto config = std::make_unique<ConfigParser>(options.
00046             config_file());
00047
00048             if (options.ast())
00049             {
00050                 std::cout << "===== " << options.config_file()
00051                     << " =====" << std::endl;
00052                 std::cout << *config;
00053
00054             if (options.run())
00055             {
00056                 Logger::level(options.loglevel());
00057                 auto app = config->make_app();
00058                 app->run();
00059             }
00060         }
00061     }
00062     catch (const std::exception &e)
00063     {
00064         std::cerr << "error: " << e.what() << std::endl;
00065         return EXIT_FAILURE;
00066     }
00067
00068     return EXIT_SUCCESS;
00069 }

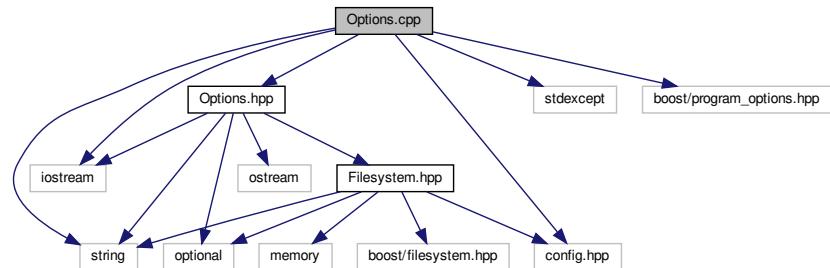
```

16.117 Options.cpp File Reference

```

#include <iostream>
#include <stdexcept>
#include <string>
#include <boost/program_options.hpp>
#include "config.hpp"
#include "Options.hpp"
Include dependency graph for Options.cpp:

```



16.118 Options.cpp

```
00001 // MAVLink router and firewall.
```

```
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <iostream>
00019 #include <stdexcept>
00020 #include <string>
00021
00022 #include <boost/program_options.hpp>
00023
00024 #include "config.hpp"
00025 #include "Options.hpp"
00026
00027
00028 namespace po = boost::program_options;
00029
00030
00031 /** Construct an options object.
00032 *
00033 * This will parse the command line arguments and construct an object for
00034 * passing the result of these arguments to the application.
00035 *
00036 * The first two arguments are designed to be taken directly from the inputs to
00037 * the standard 'main' function.
00038 *
00039 * \param argc The number of command line arguments given.
00040 * \param argv The command line arguments, as given in the arguments to the
00041 * main function.
00042 * \param filesystem A filesystem instance. The default is to construct an
00043 * instance. This exists for testing purposes.
00044 * \throws std::runtime_error if no configuration file can be found.
00045 */
00046 Options::Options(
00047     int argc, const char *argv[], const Filesystem &filesystem)
00048     : continue_(true), loglevel_(0)
00049 {
00050     // Command line options.
00051     po::options_description options(
00052         "usage: " + std::string(argv[0]));
00053     options.add_options()
00054         ("help,h", "print this message")
00055         ("config", po::value<std::string>(), "specify configuration file")
00056         ("ast", "print AST of configuration file (do not run)")
00057         ("version", "print version and license information")
00058         ("loglevel", po::value<unsigned int>(),
00059             "level of logging, between 0 and 3");
00060     po::variables_map vm;
00061     po::store(po::parse_command_line(argc, argv, options), vm);
00062     po::notify(vm);

00063     // Print help.
00064     if (vm.count("help"))
00065     {
00066         std::cout << options << std::endl;
00067         continue_ = false;
00068         return;
00069     }
00070
00071     // Print version information.
00072     if (vm.count("version"))
00073     {
00074         std::cout << "mavtables (SHAMU Project) ";
00075         std::cout << "v" << std::to_string(VERSION_MAJOR);
00076         std::cout << "." << std::to_string(VERSION_MINOR);
00077         std::cout << "." << std::to_string(VERSION_PATCH) << "\n";
00078         std::cout << "Copyright (C) 2018 Michael R. Shannon\n";
00079         std::cout << "\n";
00080         std::cout << "License: GPL v2.0 or any later version.\n";
00081         std::cout << "This is free software; see the source for copying "
```

```
00083         "conditions.  ";
00084         std::cout << "There is NO\\nwarranty; not even for MERCHANTABILITY or "
00085             "FITNESS ";
00086         std::cout << "FOR A PARTICULAR PURPOSE." << std::endl;
00087         continue_ = false;
00088         return;
00089     }
00090
00091     // Find configuration file.
00092     if (vm.count("config"))
00093     {
00094         if (filesystem.exists(vm["config"].as<std::string>()))
00095         {
00096             config_file_ = vm["config"].as<std::string>();
00097         }
00098         else
00099         {
00100             continue_ = false;
00101             throw std::runtime_error(
00102                 "mavtables could not locate a configuration file");
00103         }
00104     }
00105     else
00106     {
00107         if (auto config_file = find_config(filesystem))
00108         {
00109             config_file_ = config_file.value();
00110         }
00111         else
00112         {
00113             continue_ = false;
00114             throw std::runtime_error(
00115                 "mavtables could not locate a configuration file");
00116         }
00117     }
00118
00119     if (vm.count("loglevel"))
00120     {
00121         loglevel_ = vm["loglevel"].as<unsigned int>();
00122     }
00123
00124     // Determine actions.
00125     print_ast_ = vm.count("ast");
00126     run_firewall_ = !print_ast_;
00127 }
00128
00129
00130 /** Determine whether to print the configuration file's AST or not.
00131 */
00132 * \retval true Print abstract syntax tree of configuration file.
00133 * \retval false Don't print abstract syntax tree of configuration file.
00134 */
00135 bool Options::ast()
00136 {
00137     return print_ast_;
00138 }
00139
00140
00141 /** Get path to an existing configuration file.
00142 */
00143 * \returns The path to an existing, but not necessarily valid configuration
00144 *         file.
00145 */
00146 std::string Options::config_file()
00147 {
00148     return config_file_;
00149 }
00150
00151
00152 /** Get the log level.
00153 */
00154 * \returns The level to log at, between 0 and 3.
00155 */
00156 unsigned int Options::loglevel()
00157 {
00158     return loglevel_;
00159 }
00160
00161
00162 /** Determine whether to run the firewall/router or not.
00163 */

```

```

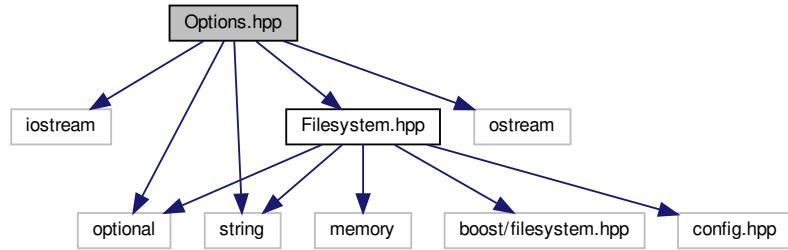
00164 * \retval true Run the firewall/router.
00165 * \retval false Don't run the firewall/router.
00166 */
00167 bool Options::run()
00168 {
00169     return run_firewall_;
00170 }
00171
00172
00173 /** Determine if the \ref Options object is valid.
00174 *
00175 * \retval true %If the options object successfully parsed the command line
00176 * arguments.
00177 * \retval false %If the program should exit immediately.
00178 */
00179 Options::operator bool() const
00180 {
00181     return continue_;
00182 }
00183
00184
00185 /** Find the configuration file.
00186 *
00187 * Find the first configuration file in the list below:
00188 * 1. The target of the 'MAVTABLES_CONFIG_PATH' environment variable.
00189 * 2. '.mavtablesrc' in the current directory.
00190 * 3. '.mavtablesrc' at '$HOME/.mavtablesrc'.
00191 * 4. The main configuration file at 'PREFIX/etc/mavtables.conf'.
00192 *
00193 * \relates Options
00194 * \param filesystem A filesystem instance. The default is to construct an
00195 * instance. This exists for testing purposes.
00196 * \returns The path to the first configuration file found. {} if no
00197 * configuration file could be found.
00198 */
00199 std::optional<std::string> find_config(const Filesystem &filesystem)
00200 {
00201     // Check MAVTABLES_CONFIG_PATH.
00202     if (auto config_path = std::getenv("MAVTABLES_CONFIG_PATH"))
00203     {
00204         if (filesystem.exists(Filesystem::path(config_path)))
00205         {
00206             return config_path;
00207         }
00208     }
00209
00210     // Check for .mavtablesrc in current directory.
00211     if (filesystem.exists(".mavtablesrc"))
00212     {
00213         return ".mavtablesrc";
00214     }
00215
00216     // Check for .mavtablesrc in home directory.
00217     if (auto home = std::getenv("HOME"))
00218     {
00219         Filesystem::path config_path(home);
00220         config_path /= ".mavtablesrc";
00221
00222         if (filesystem.exists(config_path))
00223         {
00224             return config_path.string();
00225         }
00226     }
00227
00228     // Check for PREFIX/etc/mavtables.conf.
00229     if (filesystem.exists(PREFIX "/etc/mavtables.conf"))
00230     {
00231         return PREFIX "/etc/mavtables.conf";
00232     }
00233
00234     return {};
00235 }

```

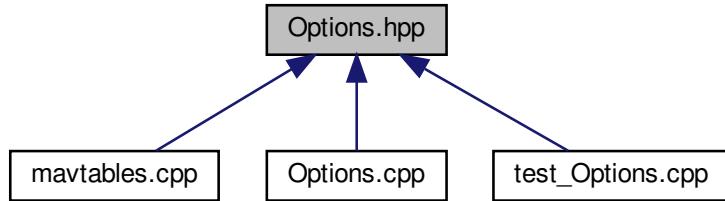
16.119 Options.hpp File Reference

```
#include <iostream>
```

```
#include <optional>
#include <iostream>
#include <string>
#include "Filesystem.hpp"
Include dependency graph for Options.hpp:
```



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



Classes

- class [Options](#)

16.119.1 Function Documentation

16.119.1.1 [find_config\(\)](#)

```
std::optional<std::string> find_config (
    const Filesystem & filesystem = Filesystem\(\) ) [related]
```

16.120 Options.hpp

```

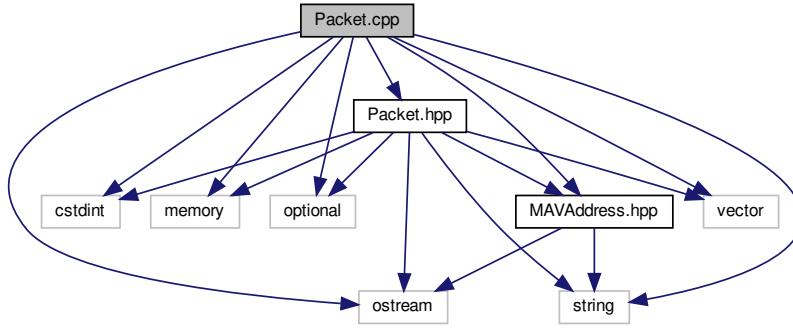
00001 // MAVLINK router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef OPTIONS_HPP_
00019 #define OPTIONS_HPP_
00020
00021
00022 #include <iostream>
00023 #include <optional>
00024 #include <ostream>
00025 #include <string>
00026
00027 #include "Filesystem.hpp"
00028
00029
00030 /** An options class which is used to parse the command line arguments.
00031 *
00032 * This class is what provides the command line help for mavtables.
00033 * ``
00034 * usage: mavtables:
00035 * -h [ --help ]           print this message
00036 * --config arg            specify configuration file
00037 * --ast                   print AST of configuration file (do not run)
00038 * --version               print version and license information
00039 * --loglevel arg          level of logging, between 0 and 3
00040 * ``
00041 */
00042 class Options
00043 {
00044     public:
00045         Options(
00046             int argc, const char *argv[],
00047             const Filesystem &filesystem = Filesystem());
00048         bool ast();
00049         unsigned int loglevel();
00050         std::string config_file();
00051         bool run();
00052         explicit operator bool() const;
00053
00054     private:
00055         bool continue_;
00056         unsigned int loglevel_;
00057         std::string config_file_;
00058         bool print_ast_;
00059         bool run_firewall_;
00060     };
00061
00062
00063 std::optional<std::string> find_config(
00064     const Filesystem &filesystem = Filesystem());
00065
00066
00067 #endif // OPTIONS_HPP_

```

16.121 Packet.cpp File Reference

```
#include <cstdint>
#include <memory>
```

```
#include <optional>
#include <iostream>
#include <string>
#include <vector>
#include "MAVAddress.hpp"
#include "Packet.hpp"
Include dependency graph for Packet.cpp:
```



16.122 Packet.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <cstdint>
00019 #include <memory>
00020 #include <optional>
00021 #include <iostream>
00022 #include <string>
00023 #include <vector>
00024
00025 #include "MAVAddress.hpp"
00026 #include "Packet.hpp"
00027
00028 /** Construct a packet.
00029 *
00030 * \param data Raw packet data.
00031 */
00032
00033 Packet::Packet(std::vector<uint8_t> data)
00034     : data_(std::move(data))
00035 {
00036 }
00037
00038
```

```

00039 // GCC generates a seemingly uncallable destructor for pure virtual classes.
00040 // Therefore, it must be excluded from test coverage.
00041 // LCOV_EXCL_START
00042 Packet::Packet()
00043 {
00044 }
00045 // LCOV_EXCL_STOP
00046
00047
00048 /** Return the packet data.
00049 *
00050 * \returns The packet data as a vector of bytes.
00051 */
00052 const std::vector<uint8_t> &Packet::data() const
00053 {
00054     return data_;
00055 }
00056
00057
00058 /** Set the source connection of the packet.
00059 *
00060 * \param connection The source connection.
00061 * \sa connection()
00062 */
00063 void Packet::connection(std::weak_ptr<Connection> connection)
00064 {
00065     connection_ = connection;
00066 }
00067
00068
00069 /** Get the source connection of the packet.
00070 *
00071 * \returns The source connection if set and it still exists, otherwise
00072 * nullptr.
00073 * \sa connection(std::weak_ptr<Connection>)
00074 */
00075 const std::shared_ptr<Connection> Packet::connection() const
00076 {
00077     return connection_.lock();
00078 }
00079
00080
00081 /** Equality comparison.
00082 *
00083 * Compares the raw packet data.
00084 *
00085 * \relates Packet
00086 * \param lhs The left hand side packet.
00087 * \param rhs The right hand side packet.
00088 * \retval true if \p lhs and \p rhs have the same packet data.
00089 * \retval false if \p lhs and \p rhs do not have the same packet data.
00090 */
00091 bool operator==(const Packet &lhs, const Packet &rhs)
00092 {
00093     return lhs.data() == rhs.data();
00094 }
00095
00096
00097 /** Inequality comparison.
00098 *
00099 * Compares the raw packet data.
00100 *
00101 * \relates Packet
00102 * \param lhs The left hand side packet.
00103 * \param rhs The right hand side packet.
00104 * \retval true if \p lhs and \p rhs do not have the same packet data.
00105 * \retval false if \p lhs and \p rhs have the same packet data.
00106 */
00107 bool operator!=(const Packet &lhs, const Packet &rhs)
00108 {
00109     return lhs.data() != rhs.data();
00110 }
00111
00112
00113 /** Print the packet to the given output stream.
00114 *
00115 * The format is "<Message Name> (#<Message ID>) from <Source Address> to
00116 * <Destination Address> (v<Packet Version>)". However, "to <Destination
00117 * Address>" is optional and will not be printed if the destination is the
00118 * broadcast address 0.0.
00119 */

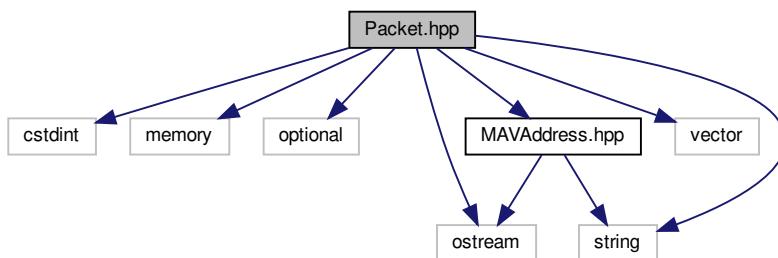
```

```

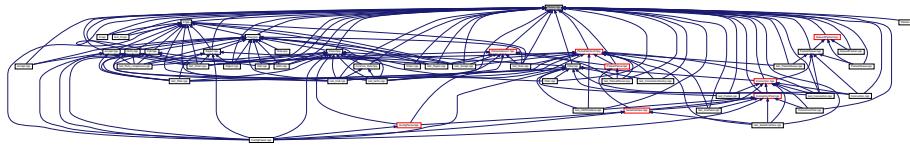
00120 * Some examples are:
00121 * - 'HEARTBEAT (#1) from 16.8 (v1.0)'
00122 * - 'PING (#4) from 128.4 to 16.8 (v2.0)'
00123 * - 'DATA_TRANSMISSION_HANDSHAKE (#130) from 16.8 (v2.0)'
00124 * - 'ENCAPSULATED_DATA (#131) from 128.4 (v2.0)'
00125 *
00126 * \relates Packet
00127 * \param os The output stream to print to.
00128 * \param packet The MAVLink packet to print.
00129 * \returns The output stream.
00130 */
00131 std::ostream &operator<<(std::ostream &os, const Packet &packet)
00132 {
00133     // ID, name, and source.
00134     os << packet.name() << " (" << packet.id() << ")";
00135     os << " from " << packet.source();
00136
00137     // Destination.
00138     if (auto dest = packet.dest())
00139     {
00140         os << " to " << dest.value();
00141     }
00142
00143     // Version.
00144     os << " (v" << ((packet.version() & 0xFF00) >> 8) << "."
00145         << (packet.version() & 0x00FF) << ")";
00146
00147     return os;
00147 }
```

16.123 Packet.hpp File Reference

```
#include <cstdint>
#include <memory>
#include <optional>
#include <iostream>
#include <string>
#include <vector>
#include "MAVAddress.hpp"
Include dependency graph for Packet.hpp:
```



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



Classes

- class [Packet](#)

16.123.1 Function Documentation

16.123.1.1 operator"!=()

```
bool operator!= (
    const Packet & lhs,
    const Packet & rhs ) [related]
```

16.123.1.2 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const Packet & packet ) [related]
```

16.123.1.3 operator==()

```
bool operator== (
    const Packet & lhs,
    const Packet & rhs ) [related]
```

16.124 Packet.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef PACKET_HPP_
00019 #define PACKET_HPP_
00020
00021
00022 #include <cstdint>
00023 #include <memory>
00024 #include <optional>
00025 #include <ostream>
00026 #include <string>
00027 #include <vector>
00028
00029 #include "MAVAddress.hpp"
00030
00031
00032 class Connection;
00033
00034
00035 /** A MAVLink packet.
00036 *
00037 * This is an abstract class, it is meant to be overridden by classes
00038 * implementing either version 1 or version 2 of the MAVLink packet wire
00039 * protocol.
00040 *
00041 * \internal The \ref Version enum must be changed if another packet version is
00042 * added.
00043 */
00044 class Packet
00045 {
00046     public:
00047         enum Version
00048         {
00049             V1 = 0x0100, //!< MAVLink Version 1.0
00050             V2 = 0x0200 //!< MAVLink Version 2.0
00051         };
00052
00053     /** Copy constructor.
00054     *
00055     * \param other Packet to copy from.
00056     */
00057     Packet(const Packet &other) = default;
00058     /** Move constructor.
00059     *
00060     * \param other Packet to move from.
00061     */
00062     Packet(Packet &&other) = default;
00063     Packet(std::vector<uint8_t> data);
00064     virtual ~Packet(); // Clang does not like pure virtual destructors.
00065     /** Return packet version.
00066     *
00067     * \returns Two byte Packet version with major version in MSB and minor
00068     * version in LSB.
00069     */
00070     virtual Version version() const = 0;
00071     /** Return MAVLink message ID.
00072     *
00073     * \returns The numeric message ID of the MAVLink packet (0 to 255).
00074     */
00075     virtual unsigned long id() const = 0;
00076     /** Return MAVLink message name.
00077     *
```

```

00078     * \returns The message name of the MAVLink packet.
00079     */
00080     virtual std::string name() const = 0;
00081     /** Return source address.
00082     *
00083     * Where the packet came from.
00084     *
00085     * \returns The source MAVLink address of the packet.
00086     */
00087     virtual MAVAddress source() const = 0;
00088     /** Return destination address.
00089     *
00090     * Where the packet is sent to. This is optional because not all
00091     * packets have a destination. %If a system is specified, but not a
00092     * component, a component ID of 0 will be used (the broadcast ID).
00093     *
00094     * \returns The destination MAVLink address of the packet if not a
00095     * broadcast packet. %If the packet does not have a destination
00096     * address then {} will be returned.
00097     */
00098     virtual std::optional<MAVAddress> dest() const = 0;
00099     void connection(std::weak_ptr<Connection> connection);
00100     const std::shared_ptr<Connection> connection() const;
00101     const std::vector<uint8_t> &data() const;
00102     /** Assignment operator.
00103     *
00104     * \param other Packet to copy from.
00105     */
00106     Packet &operator=(const Packet &other) = default;
00107     /** Assignment operator (by move semantics).
00108     *
00109     * \param other Packet to move from.
00110     */
00111     Packet &operator=(Packet &&other) = default;
00112
00113     private:
00114         std::vector<uint8_t> data_;
00115         std::weak_ptr<Connection> connection_;
00116     };
00117
00118
00119     bool operator==(const Packet &lhs, const Packet &rhs);
00120     bool operator!=(const Packet &lhs, const Packet &rhs);
00121     std::ostream &operator<<(std::ostream &os, const Packet &packet);
00122
00123
00124 #endif // PACKET_HPP_

```

16.125 packet_scripter.py File Reference

Namespaces

- [packet_scripter](#)

Functions

- def [packet_scripter.send_packet](#) (mav, packet, system=0, component=0)
- def [packet_scripter.parse_line](#) (line)
- def [packet_scripter.parse_file](#) (filename)
- def [packet_scripter.start_connection](#) (args)
- def [packet_scripter.parse_args](#) ()
- def [packet_scripter.main](#) ()

16.126 packet_scripter.py

```

00001 #!/usr/bin/env python
00002
00003 from __future__ import (absolute_import, division,
00004                            print_function, unicode_literals)
00005 from builtins import *
00006
00007 import os
00008 import sys
00009 from time import sleep
00010 from argparse import ArgumentParser
00011 from threading import Thread
00012 from pymavlink import mavutil
00013 from datetime import datetime, timedelta
00014
00015
00016 def send_packet(mav, packet, system=0, component=0):
00017     def a(n):
00018         return tuple(range(0, n))
00019     def b(n):
00020         return bytes(range(0, n))
00021     if packet == 'SYS_STATUS': # 1
00022         mav.mav.sys_status_send(*a(13))
00023     elif packet == 'SYSTEM_TIME': # 2
00024         mav.mav.system_time_send(*a(2))
00025     elif packet == 'PING': # 4
00026         args = a(2) + (system, component)
00027         mav.mav.ping_send(*args)
00028     elif packet == 'CHANGE_OPERATOR_CONTROL': # 5
00029         args = (system,) + a(2) + (b(25),)
00030         mav.mav.change_operator_control_send(*args)
00031     elif packet == 'CHANGE_OPERATOR_CONTROL_ACK': # 6
00032         mav.mav.change_operator_control_ack_send(*a(3))
00033     elif packet == 'AUTH_KEY': # 7
00034         mav.mav.auth_key_send(b(32))
00035     elif packet == 'PARAM_REQUEST_READ': # 20
00036         mav.mav.param_request_read_send(system, component, b(16), 1)
00037     elif packet == 'PARAM_REQUEST_LIST': # 21
00038         mav.mav.param_request_list_send(system, component)
00039     elif packet == 'PARAM_VALUE': # 22
00040         mav.mav.param_value_send(b(16), *a(4))
00041     elif packet == 'PARAM_SET': # 23
00042         mav.mav.param_set_send(system, component, b(16), *a(2))
00043     elif packet == 'GPS_RAW_INT': # 24
00044         if mav.mavlink20():
00045             mav.mav.gps_raw_int_send(*a(15))
00046         else:
00047             mav.mav.gps_raw_int_send(*a(10))
00048     elif packet == 'GPS_STATUS': # 25
00049         mav.mav.gps_status_send(1, b(20), b(20), b(20), b(20))
00050     elif packet == 'SCALDED_IMU': # 26
00051         mav.mav.scaled_imu_send(*a(10))
00052     elif packet == 'RAW_IMU': # 27
00053         mav.mav.raw_imu_send(*a(10))
00054     elif packet == 'RAW_PRESSURE': # 28
00055         mav.mav.raw_pressure_send(*a(5))
00056     elif packet == 'SCALDED_PRESSURE': # 29
00057         mav.mav.scaled_pressure_send(*a(4))
00058     elif packet == 'ATTITUDE': # 30
00059         mav.mav.attitude_send(*a(7))
00060     elif packet == 'ATTITUDE_QUATERNION': # 31
00061         mav.mav.attitude_quaternion_send(*a(8))
00062     elif packet == 'LOCAL_POSITION_NED': # 32
00063         mav.mav.local_position_ned_send(*a(7))
00064     elif packet == 'GLOBAL_POSITION_INT': # 33
00065         mav.mav.global_position_int_send(*a(9))
00066     elif packet == 'RC_CHANNELS_SCALED': # 34
00067         mav.mav.rc_channels_scaled_send(*a(11))
00068     elif packet == 'RC_CHANNELS_RAW': # 35
00069         mav.mav.rc_channels_raw_send(*a(11))
00070     elif packet == 'SERVO_OUTPUT_RAW': # 36
00071         if mav.mavlink20():
00072             mav.mav.servo_output_raw_send(*a(18))
00073         else:
00074             mav.mav.servo_output_raw_send(*a(10))
00075     elif packet == 'MISSION_REQUEST_PARTIAL_LIST': # 37
00076         if mav.mavlink20():
00077             mav.mav.mission_request_partial_list_send(system, *a(3))

```

```

00078     else:
00079         mav.mav.mission_request_partial_list_send(system, component, *a(2))
00080     elif packet == 'MISSION_WRITE_PARTIAL_LIST': # 38
00081         if mav.mavlink20():
00082             mav.mav.mission_write_partial_list_send(system, component, *a(3))
00083         else:
00084             mav.mav.mission_write_partial_list_send(system, component, *a(2))
00085     elif packet == 'MISSION_ITEM': # 39
00086         if mav.mavlink20():
00087             mav.mav.mission_item_send(system, component, *a(13))
00088         else:
00089             mav.mav.mission_item_send(system, component, *a(12))
00090     elif packet == 'MISSION_REQUEST': # 40
00091         if mav.mavlink20():
00092             mav.mav.mission_request_send(system, component, *a(2))
00093         else:
00094             mav.mav.mission_request_send(system, component, 1)
00095     elif packet == 'MISSION_SET_CURRENT': # 41
00096         mav.mav.mission_set_current_send(system, component, 1)
00097     elif packet == 'MISSION_CURRENT': # 42
00098         mav.mav.mission_current_send(1)
00099     elif packet == 'MISSION_REQUEST_LIST': # 43
00100         if mav.mavlink20():
00101             mav.mav.mission_request_list_send(system, component, 1)
00102         else:
00103             mav.mav.mission_request_list_send(system, component)
00104     elif packet == 'MISSION_COUNT': # 44
00105         if mav.mavlink20():
00106             mav.mav.mission_count_send(system, component, *a(2))
00107         else:
00108             mav.mav.mission_count_send(system, component, 1)
00109     elif packet == 'MISSION_CLEAR_ALL': # 45
00110         if mav.mavlink20():
00111             mav.mav.mission_clear_all_send(system, component, 1)
00112         else:
00113             mav.mav.mission_clear_all_send(system, component)
00114     elif packet == 'MISSION_ITEM_REACHED': # 46
00115         mav.mav.mission_item_reached_send(1)
00116     elif packet == 'MISSION_ACK': # 47
00117         if mav.mavlink20():
00118             mav.mav.mission_ack_send(system, component, *a(2))
00119         else:
00120             mav.mav.mission_ack_send(system, component, 1)
00121     elif packet == 'SET_GPS_GLOBAL_ORIGIN': # 48
00122         if mav.mavlink20():
00123             mav.mav.set_gps_global_origin_send(system, *a(4))
00124         else:
00125             mav.mav.set_gps_global_origin_send(system, *a(3))
00126     elif packet == 'GPS_GLOBAL_ORIGIN': # 49
00127         if mav.mavlink20():
00128             mav.mav.gps_global_origin_send(*a(4))
00129         else:
00130             mav.mav.gps_global_origin_send(*a(3))
00131     elif packet == 'PARAM_MAP_RC': # 50
00132         mav.mav.param_map_rc_send(system, component, b(16), *a(6))
00133     elif packet == 'MISSION_REQUEST_INT': # 51
00134         if mav.mavlink20():
00135             mav.mav.mission_request_int_send(system, component, *a(2))
00136         else:
00137             mav.mav.mission_request_int_send(system, component, 1)
00138     elif packet == 'SAFETY_SET_ALLOWED_AREA': # 54
00139         mav.mav.safety_set_allowed_area_send(system, component, *a(7))
00140     elif packet == 'SAFETY_ALLOWED_AREA': # 55
00141         mav.mav.safety_allowed_area_send(*a(7))
00142     elif packet == 'ATTITUDE_QUATERNION_COV': # 61
00143         args = (1, b(4)) + a(3) + (b(9),)
00144         mav.mav.attitude_quaternion_cov_send(*args)
00145     elif packet == 'NAV_CONTROLLER_OUTPUT': # 62
00146         mav.mav.nav_controller_output_send(*a(8))
00147     elif packet == 'GLOBAL_POSITION_INT_COV': # 63
00148         args = a(9) + (b(36),)
00149         mav.mav.global_position_int_cov_send(*args)
00150     elif packet == 'LOCAL_POSITION_NED_COV': # 64
00151         args = a(11) + (b(45),)
00152         mav.mav.local_position_ned_cov_send(*args)
00153     elif packet == 'RC_CHANNELS': # 65
00154         mav.mav.rc_channels_send(*a(21))
00155     elif packet == 'MANUAL_CONTROL': # 69
00156         mav.mav.manual_control_send(system, *a(5))
00157     elif packet == 'RC_CHANNELS_OVERRIDE': # 70
00158         mav.mav.rc_channels_override_send(system, component, *a(8))

```

```

00159     elif packet == 'MISSION_ITEM_INT': # 73
00160         if mav.mavlink20():
00161             mav.mav.mission_item_int_send(system, component, *a(13))
00162         else:
00163             mav.mav.mission_item_int_send(system, component, *a(12))
00164     elif packet == 'VFR_HUD': # 74
00165         mav.mav.vfr_hud_send(*a(6))
00166     elif packet == 'COMMAND_INT': # 75
00167         mav.mav.command_int_send(system, component, *a(11))
00168     elif packet == 'COMMAND_LONG': # 76
00169         mav.mav.command_long_send(system, component, *a(9))
00170     elif packet == 'COMMAND_ACK': # 77
00171         mav.mav.command_ack_send(*a(2))
00172     elif packet == 'MANUAL_SETPOINT': # 81
00173         mav.mav.manual_setpoint_send(*a(7))
00174     elif packet == 'SET_ATTITUDE_TARGET': # 82
00175         mav.mav.set_attitude_target_send(1, system, component, 1, b(4), *a(4))
00176     elif packet == 'ATTITUDE_TARGET': # 83
00177         args = a(2) + (b(4), ) + a(4)
00178         mav.mav.attitude_target_send(*args)
00179     elif packet == 'SET_POSITION_TARGET_LOCAL_NED': # 84
00180         mav.mav.set_position_target_local_ned_send(
00181             1, system, component, *a(13))
00182     elif packet == 'POSITION_TARGET_LOCAL_NED': # 85
00183         mav.mav.position_target_local_ned_send(*a(14))
00184     elif packet == 'SET_POSITION_TARGET_GLOBAL_INT': # 86
00185         mav.mav.set_position_target_global_int_send(
00186             1, system, component, *a(13))
00187     elif packet == 'POSITION_TARGET_GLOBAL_INT': # 87
00188         mav.mav.position_target_global_int_send(*a(14))
00189     elif packet == 'LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET': # 89
00190         mav.mav.local_position_ned_system_global_offset_send(*a(7))
00191     elif packet == 'HIL_CONTROLS': # 91
00192         mav.mav.hil_controls_send(*a(11))
00193     elif packet == 'HIL_RC_INPUTS_RAW': # 92
00194         mav.mav.hil_rc_inputs_raw_send(*a(14))
00195     elif packet == 'HIL_ACTUATOR_CONTROLS': # 93
00196         mav.mav.hil_actuator_controls_send(1, b(16), *a(2))
00197     elif packet == 'OPTICAL_FLOW': # 100
00198         if mav.mavlink20():
00199             mav.mav.optical_flow_send(*a(10))
00200         else:
00201             mav.mav.optical_flow_send(*a(8))
00202     elif packet == 'GLOBAL_VISION_POSITION_ESTIMATE': # 101
00203         if mav.mavlink20():
00204             args = a(7) + (b(21),)
00205             mav.mav.global_vision_position_estimate_send(*args)
00206         else:
00207             mav.mav.global_vision_position_estimate_send(*a(7))
00208     elif packet == 'VISION_POSITION_ESTIMATE': # 102
00209         if mav.mavlink20():
00210             args = a(7) + (b(21),)
00211             mav.mav.vision_position_estimate_send(*args)
00212         else:
00213             mav.mav.vision_position_estimate_send(*a(7))
00214     elif packet == 'VISION_SPEED_ESTIMATE': # 103
00215         if mav.mavlink20():
00216             args = a(4) + (b(9),)
00217             mav.mav.vision_speed_estimate_send(*args)
00218         else:
00219             mav.mav.vision_speed_estimate_send(*a(4))
00220     elif packet == 'VICON_POSITION_ESTIMATE': # 104
00221         if mav.mavlink20():
00222             args = a(7) + (b(21),)
00223             mav.mav.vicon_position_estimate_send(*args)
00224         else:
00225             mav.mav.vicon_position_estimate_send(*a(7))
00226     elif packet == 'HIGHRES_IMU': # 105
00227         mav.mav.highres_imu_send(*a(15))
00228     elif packet == 'OPTICAL_FLOW_RAD': # 106
00229         mav.mav.optical_flow_rad_send(*a(12))
00230     elif packet == 'HIL_SENSOR': # 107
00231         mav.mav.hil_sensor_send(*a(15))
00232     elif packet == 'SIM_STATE': # 108
00233         mav.mav.sim_state_send(*a(21))
00234     elif packet == 'RADIO_STATUS': # 109
00235         mav.mav.radio_status_send(*a(7))
00236     elif packet == 'FILE_TRANSFER_PROTOCOL': # 110
00237         mav.mav.file_transfer_protocol_send(0, system, component, b(251))
00238     elif packet == 'TIMESYNC': # 111
00239         mav.mav.timesync_send(*a(2))

```

```

00240     elif packet == 'CAMERA_TRIGGER': # 112
00241         mav.mav.camera_trigger_send(*a(2))
00242     elif packet == 'HIL_GPS': # 113
00243         mav.mav.hil_gps_send(*a(13))
00244     elif packet == 'HIL_OPTICAL_FLOW': # 114
00245         mav.mav.hil_optical_flow_send(*a(12))
00246     elif packet == 'HIL_STATE_QUATERNION': # 115
00247         mav.mav.hil_state_quaternion_send(1, b(4), *a(14))
00248     elif packet == 'SCALED_IMU2': # 116
00249         mav.mav.scaled_imu2_send(*a(10))
00250     elif packet == 'LOG_REQUEST_LIST': # 117
00251         mav.mav.log_request_list_send(system, component, *a(2))
00252     elif packet == 'LOG_ENTRY': # 118
00253         mav.mav.log_entry_send(*a(5))
00254     elif packet == 'LOG_REQUEST_DATA': # 119
00255         mav.mav.log_request_data_send(system, component, *a(3))
00256     elif packet == 'LOG_DATA': # 120
00257         args = a(3) + (b(90),)
00258         mav.mav.log_data_send(*args)
00259     elif packet == 'LOG_ERASE': # 121
00260         mav.mav.log_erase_send(system, component)
00261     elif packet == 'LOG_REQUEST_END': # 122
00262         mav.mav.log_request_end_send(system, component)
00263     elif packet == 'GPS_INJECT_DATA': # 123
00264         mav.mav.gps_inject_data_send(system, component, 110, b(110))
00265     elif packet == 'GPS2_RAW': # 124
00266         mav.mav.gps2_raw_send(*a(12))
00267     elif packet == 'POWER_STATUS': # 125
00268         mav.mav.power_status_send(*a(3))
00269     elif packet == 'SERIAL_CONTROL': # 126
00270         args = a(5) + (b(70),)
00271         mav.mav.serial_control_send(*args)
00272     elif packet == 'GPS_RTK': # 127
00273         mav.mav.gps_rtk_send(*a(13))
00274     elif packet == 'GPS2_RTK': # 128
00275         mav.mav.gps2_rtk_send(*a(13))
00276     elif packet == 'SCALED_IMU3': # 129
00277         mav.mav.scaled_imu3_send(*a(10))
00278     elif packet == 'DATA_TRANSMISSION_HANDSHAKE': # 130
00279         mav.mav.data_transmission_handshake_send(*a(7))
00280     elif packet == 'ENCAPSULATED_DATA': # 131
00281         mav.mav.encapsulated_data_send(1, b(253))
00282     elif packet == 'DISTANCE_SENSOR': # 132
00283         mav.mav.distance_sensor_send(*a(8))
00284     elif packet == 'TERRAIN_REQUEST': # 133
00285         mav.mav.terrain_request_send(*a(4))
00286     elif packet == 'TERRAIN_DATA': # 134
00287         args = a(4) + (b(16),)
00288         mav.mav.terrain_data_send(*args)
00289     elif packet == 'TERRAIN_CHECK': # 135
00290         mav.mav.terrain_check_send(*a(2))
00291     elif packet == 'TERRAIN_REPORT': # 136
00292         mav.mav.terrain_report_send(*a(7))
00293     elif packet == 'SCALED_PRESSURE2': # 137
00294         mav.mav.scaled_pressure2_send(*a(4))
00295     elif packet == 'ATT_POS_MOCAP': # 138
00296         if mav.mavlink20():
00297             args = (1, b(4)) + a(3) + (b(21),)
00298             mav.mav.att_pos_mocap_send(*args)
00299         else:
00300             mav.mav.att_pos_mocap_send(1, b(4), *a(3))
00301     elif packet == 'SET_ACTUATOR_CONTROL_TARGET': # 139
00302         args = a(2) + (system, component, b(8))
00303         mav.mav.set_actuator_control_target_send(*args)
00304     elif packet == 'ACTUATOR_CONTROL_TARGET': # 140
00305         args = a(2) + (b(8),)
00306         mav.mav.actuator_control_target_send(*args)
00307     elif packet == 'ALTITUDE': # 141
00308         mav.mav.altitude_send(*a(7))
00309     elif packet == 'RESOURCE_REQUEST': # 142
00310         args = a(2) + (b(120), 1, b(120))
00311         mav.mav.resource_request_send(*args)
00312     elif packet == 'SCALED_PRESSURE3': # 143
00313         mav.mav.scaled_pressure3_send(*a(4))
00314     elif packet == 'FOLLOW_TARGET': # 144
00315         args = a(5) + (b(3), b(3), b(4), b(3), b(3), 1)
00316         mav.mav.follow_target_send(*args)
00317     elif packet == 'CONTROL_SYSTEM_STATE': # 146
00318         args = a(11) + (b(3), b(3), b(4)) + a(3)
00319         mav.mav.control_system_state_send(*args)
00320     elif packet == 'BATTERY_STATUS': # 147

```

```

00321     args = a(4) + (b(10),) + a(4)
00322     mav.mav.battery_status_send(*args)
00323 elif packet == 'AUTOPILOT_VERSION': # 148
00324     if mav.mavlink20():
00325         args = a(5) + (b(8), b(8), b(8)) + a(3) + (b(18),)
00326         mav.mav.autopilot_version_send(*args)
00327     else:
00328         args = a(5) + (b(8), b(8), b(8)) + a(3)
00329         mav.mav.autopilot_version_send(*args)
00330 elif packet == 'LANDING_TARGET': # 149
00331     if mav.mavlink20():
00332         args = a(11) + (b(4),) + a(2)
00333         mav.mav.landing_target_send(*args)
00334     else:
00335         mav.mav.landing_target_send(*a(8))
00336 elif packet == 'ESTIMATOR_STATUS': # 230
00337     mav.mav.estimator_status_send(*a(10))
00338 elif packet == 'WIND_COV': # 231
00339     mav.mav.wind_cov_send(*a(9))
00340 elif packet == 'GPS_INPUT': # 232
00341     mav.mav.gps_input_send(*a(18))
00342 elif packet == 'GPS_RTCM_DATA': # 233
00343     args = a(2) + (b(180),)
00344     mav.mav.gps_rtcm_data_send(*args)
00345 elif packet == 'HIGH_LATENCY': # 234
00346     mav.mav.high_latency_send(*a(24))
00347 elif packet == 'VIBRATION': # 241
00348     mav.mav.vibration_send(*a(7))
00349 elif packet == 'HOME_POSITION': # 242
00350     if mav.mavlink20():
00351         args = a(6) + (b(4),) + a(4)
00352         mav.mav.home_position_send(*args)
00353     else:
00354         args = a(6) + (b(4),) + a(3)
00355         mav.mav.home_position_send(*args)
00356 elif packet == 'SET_HOME_POSITION': # 243
00357     if mav.mavlink20():
00358         args = (system,) + a(6) + (b(4),) + a(4)
00359         mav.mav.set_home_position_send(*args)
00360     else:
00361         args = (system,) + a(6) + (b(4),) + a(3)
00362         mav.mav.set_home_position_send(*args)
00363 elif packet == 'MESSAGE_INTERVAL': # 244
00364     mav.mav.message_interval_send(*a(2))
00365 elif packet == 'EXTENDED_SYS_STATE': # 245
00366     mav.mav.extended_sys_state_send(*a(2))
00367 elif packet == 'ADSB_VEHICLE': # 246
00368     args = a(8) + (b(9),) + a(4)
00369     mav.mav.adsb_vehicle_send(*args)
00370 elif packet == 'COLLISION': # 247
00371     mav.mav.collision_send(*a(7))
00372 elif packet == 'V2_EXTENSION': # 248
00373     mav.mav.v2_extension_send(1, system, component, 1, a(249))
00374 elif packet == 'MEMORY_VECT': # 249
00375     args = a(3) + (b(32),)
00376     mav.mav.memory_vect_send(*args)
00377 elif packet == 'DEBUG_VECT': # 250
00378     mav.mav.debug_vect_send(b(10), *a(4))
00379 elif packet == 'NAMED_VALUE_FLOAT': # 251
00380     mav.mav.named_value_float_send(1, b(10), 1)
00381 elif packet == 'NAMED_VALUE_INT': # 252
00382     mav.mav.named_value_int_send(1, b(10), 1)
00383 elif packet == 'STATUSTEXT': # 253
00384     mav.mav.statustext_send(1, b(50))
00385 elif packet == 'DEBUG': # 254
00386     mav.mav.debug_send(*a(3))
00387 elif packet == 'SETUP_SIGNING': # 256
00388     mav.mav.setup_signing_send(system, component, b(32), 1)
00389 elif packet == 'BUTTON_CHANGE': # 257
00390     mav.mav.button_change_send(*a(3))
00391 elif packet == 'PLAY_TUNE': # 258
00392     mav.mav.play_tune_send(system, component, b(30))
00393 elif packet == 'CAMERA_IMAGE_CAPTURED': # 263
00394     args = a(7) + (b(4),) + a(2) + (b(205),)
00395     mav.mav.camera_image_captured_send(*args)
00396 elif packet == 'FLIGHT_INFORMATION': # 264
00397     mav.mav.flight_information_send(*a(4))
00398 elif packet == 'MOUNT_ORIENTATION': # 265
00399     mav.mav.mount_orientation_send(*a(4))
00400 elif packet == 'LOGGING_DATA': # 266
00401     args = (system, component) + a(3) + (b(249),)

```

```

00402     mav.mav.logging_data_send(*args)
00403 elif packet == 'LOGGING_DATA_ACKED': # 267
00404     args = (system, component) + a(3) + (b(249),)
00405     mav.mav.logging_data_acked_send(*args)
00406 elif packet == 'LOGGING_ACK': # 268
00407     mav.mav.logging_ack_send(system, component, 1)
00408 elif packet == 'WIFI_CONFIG_AP': # 299
00409     mav.mav.wifi_config_ap_send(b(32), b(64))
00410 elif packet == 'PROTOCOL_VERSION': # 300
00411     args = a(3) + (b(8), b(8))
00412     mav.mav.protocol_version_send(*args)
00413 elif packet == 'UAVCAN_NODE_STATUS': # 310
00414     mav.mav.uavcan_node_status_send(*a(6))
00415 elif packet == 'UAVCAN_NODE_INFO': # 311
00416     args = a(2) + (b(80),) + a(2) + (b(16),) + a(3)
00417     mav.mav.uavcan_node_info_send(*args)
00418 elif packet == 'OBSTACLE_DISTANCE': # 330
00419     args = a(2) + (b(72),) + a(3)
00420     mav.mav.obstacle_distance_send(*args)
00421 else:
00422     print('unknown packet type {:s}'.format(packet))
00423     sys.exit(1)
00424
00425
00426 def parse_line(line):
00427     parts = line.split(' to ')
00428     packet = parts[0]
00429     try:
00430         system, component = parts[1].split('.')
00431     except:
00432         system = 0
00433         component = 0
00434     return packet, int(system), int(component)
00435
00436
00437 def parse_file(filename):
00438     with open(filename) as f:
00439         content = f.readlines()
00440     return [parse_line(x.strip()) for x in content]
00441
00442
00443 def start_connection(args):
00444     if not args['mavlink1']:
00445         os.environ['MAVLINK20'] = '1'
00446     mavutil.set_dialect('common')
00447     if args['udp']:
00448         mav = mavutil.mavlink_connection('udpout:' + args['udp'],
00449                                         source_system=args['system'], source_component=args['component'])
00450     elif args['serial']:
00451         mav = mavutil.mavlink_connection(args['serial'], baud=57600,
00452                                         source_system=args['system'], source_component=args['component'])
00453     else:
00454         sys.exit()
00455     return mav
00456
00457
00458 def parse_args():
00459     parser = ArgumentParser(description='Send mavlink packets from file.')
00460     parser.add_argument('system', type=int, help='system ID')
00461     parser.add_argument('component', type=int, help='component ID')
00462     parser.add_argument('script', help='script file to run')
00463     parser.add_argument(
00464         '--mavlink1', action='store_true',
00465         help='force MAVLink v1.0 instead of v2.0')
00466     parser.add_argument(
00467         '--udp', action='store', help='UDP address:port to connect to')
00468     parser.add_argument(
00469         '--serial', action='store', help='serial port device string')
00470     args = vars(parser.parse_args())
00471     if not args['udp'] and not args['serial']:
00472         print('expected --udp or --serial option')
00473         sys.exit()
00474     if args['udp'] and args['serial']:
00475         print('expected --udp or --serial option, but not both')
00476         sys.exit()
00477     return args
00478
00479
00480 def main():
00481     args = parse_args()
00482     packets = parse_file(args['script'])

```

```

00483     mav = start_connection(args)
00484     last_heartbeat = datetime.now() - timedelta(seconds=180)
00485     for packet, system, component in packets:
00486         if ((datetime.now() - last_heartbeat) > timedelta(seconds=120)):
00487             last_heartbeat = datetime.now()
00488             mav.mav.heartbeat_send(0, 0, 0, 0, 0)
00489             send_packet(mav, packet, system, component)
00490             sleep(0.0003)
00491
00492
00493 if __name__ == '__main__':
00494     main()

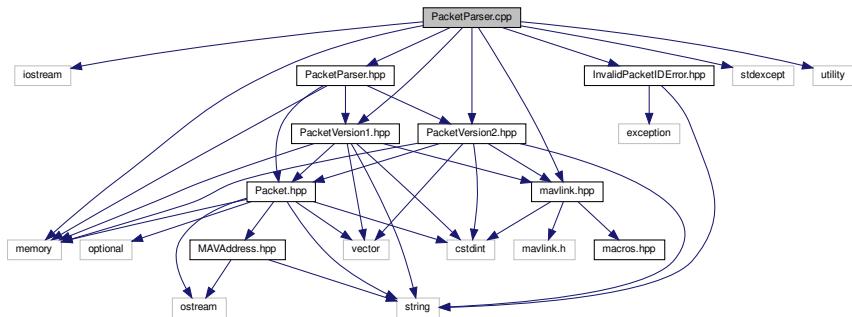
```

16.127 PacketParser.cpp File Reference

```

#include <iostream>
#include <memory>
#include <stdexcept>
#include <utility>
#include "InvalidPacketIDError.hpp"
#include "mavlink.hpp"
#include "PacketParser.hpp"
#include "PacketVersion1.hpp"
#include "PacketVersion2.hpp"
Include dependency graph for PacketParser.cpp:

```



16.128 PacketParser.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.

```

```
00016
00017
00018 #include <iostream>
00019 #include <memory>
00020 #include <stdexcept>
00021 #include <utility>
00022
00023 #include "InvalidPacketIDError.hpp"
00024 #include "mavlink.hpp"
00025 #include "PacketParser.hpp"
00026 #include "PacketVersion1.hpp"
00027 #include "PacketVersion2.hpp"
00028
00029
00030 /** Construct a \ref PacketParser.
00031 */
00032 PacketParser::PacketParser()
00033     : state_(WAITING_FOR_START_BYTE)
00034 {
00035     clear();
00036 }
00037
00038
00039 /** Return the number of bytes parsed on the current packet.
00040 */
00041 * \returns The number of bytes parsed on the current packet, 0 if no packet is
00042 *         currently being parsed.
00043 */
00044 size_t PacketParser::bytes_parsed() const
00045 {
00046     return buffer_.size();
00047 }
00048
00049
00050 /** Reset packet parser so it can parse another packet.
00051 */
00052 * %If called while parsing a packet, that packet will be lost.
00053 */
00054 void PacketParser::clear()
00055 {
00056     buffer_.clear();
00057     buffer_.reserve(MAVLINK_MAX_PACKET_LEN);
00058     state_ = WAITING_FOR_START_BYTE;
00059     version_ = Packet::V2;
00060     bytes_remaining_ = 0;
00061 }
00062
00063
00064 /** Parse a MAVLink wire protocol byte, v1.0 or v2.0.
00065 */
00066 * When a packet is completed it will be returned and the parser reset so it
00067 * can be used to continue parsing.
00068 *
00069 * \param byte A byte from the MAVLink wire protocol.
00070 * \returns A complete v1.0 or v2.0 packet. %If the parser has not yet parsed
00071 *         a complete packet, nullptr is returned.
00072 */
00073 std::unique_ptr<Packet> PacketParser::parse_byte(uint8_t byte)
00074 {
00075     std::unique_ptr<Packet> packet;
00076
00077     switch (state_)
00078     {
00079         case WAITING_FOR_START_BYTE:
00080             waiting_for_start_byte_(byte);
00081             break;
00082
00083         case WAITING_FOR_HEADER:
00084             waiting_for_header_(byte);
00085             break;
00086
00087         case WAITING_FOR_PACKET:
00088             packet = waiting_for_packet_(byte);
00089     }
00090
00091     return packet;
00092 }
00093
00094
00095 /** Check for start of packet.
00096 */
```

```
00097 * Start packet parsing if the given \p byte is a start byte for either v1.0 or
00098 * v2.0 packets. Next state will be 'WAITING_FOR_HEADER' if this is the case.
00099 *
00100 * \param byte The byte to attempt parsing.
00101 */
00102 void PacketParser::waiting_for_start_byte_(uint8_t byte)
00103 {
00104     if (byte == packet_v1::START_BYTE)
00105     {
00106         // Store start byte and begin receiving header.
00107         buffer_.push_back(byte);
00108         state_ = WAITING_FOR_HEADER;
00109         version_ = packet_v1::VERSION;
00110     }
00111     else if (byte == packet_v2::START_BYTE)
00112     {
00113         // Store start byte and begin receiving header.
00114         buffer_.push_back(byte);
00115         state_ = WAITING_FOR_HEADER;
00116         version_ = packet_v2::VERSION;
00117     }
00118 }
00119
00120
00121 /** Parser header byte.
00122 *
00123 * Next state will be 'WAITING_FOR_PACKET' if the given \p byte completes the
00124 * header.
00125 *
00126 * \param byte The byte to attempt parsing.
00127 */
00128 void PacketParser::waiting_for_header_(uint8_t byte)
00129 {
00130     buffer_.push_back(byte);
00131
00132     switch (version_)
00133     {
00134         case packet_v1::VERSION:
00135             if (packet_v1::header_complete(buffer_))
00136             {
00137                 // Set number of expected bytes and start waiting for
00138                 // remainder of packet.
00139                 bytes_remaining_ = packet_v1::header(buffer_)-
00140                             len +
00141                             packet_v1::CHECKSUM_LENGTH;
00142                 state_ = WAITING_FOR_PACKET;
00143             }
00144             break;
00145
00146         case packet_v2::VERSION:
00147             if (packet_v2::header_complete(buffer_))
00148             {
00149                 // Set number of expected bytes and start waiting for
00150                 // remainder of packet.
00151                 bytes_remaining_ = packet_v2::header(buffer_)-
00152                             len +
00153                             packet_v2::CHECKSUM_LENGTH;
00154
00155                 if (packet_v2::is_signed(buffer_))
00156                 {
00157                     bytes_remaining_ += packet_v2::SIGNATURE_LENGTH;
00158                 }
00159
00160                 state_ = WAITING_FOR_PACKET;
00161             }
00162             break;
00163     }
00164 }
00165
00166
00167 /** Parse packet bytes.
00168 *
00169 * %If this \p byte completes the packet a \ref std::unique_ptr to the packet
00170 * is returned and the parser is reset to begin parsing another packet (next
00171 * state WAITING_FOR_START_BYTE). Otherwise a nullptr is returned.
00172 *
00173 * \param byte The byte to attempt parsing.
00174 * \returns A unique pointer to the complete packet. %If the packet is not
00175 * complete a nullptr is returned.
```

```

00176  */
00177 std::unique_ptr<Packet> PacketParser::waiting_for_packet_(uint8_t byte)
00178 {
00179     buffer_.push_back(byte);
00180     --bytes_remaining_;
00181
00182     if (bytes_remaining_ == 0)
00183     {
00184         std::unique_ptr<Packet> packet;
00185
00186         try
00187         {
00188             switch (version_)
00189             {
00190                 case packet_v1::VERSION:
00191                     packet = std::make_unique<packet_v1::Packet>(
00192                         std::move(buffer_));
00193                     break;
00194
00195                 case packet_v2::VERSION:
00196                     packet = std::make_unique<packet_v2::Packet>(
00197                         std::move(buffer_));
00198                     break;
00199             }
00200         }
00201         catch (const InvalidPacketIDError &err)
00202         {
00203             packet = nullptr;
00204             std::cerr << err.what() << std::endl;
00205         }
00206
00207         clear();
00208         return packet;
00209     }
00210
00211     return nullptr;
00212 }

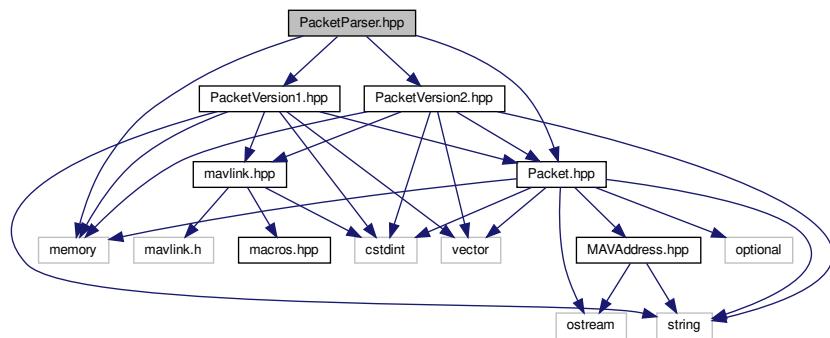
```

16.129 PacketParser.hpp File Reference

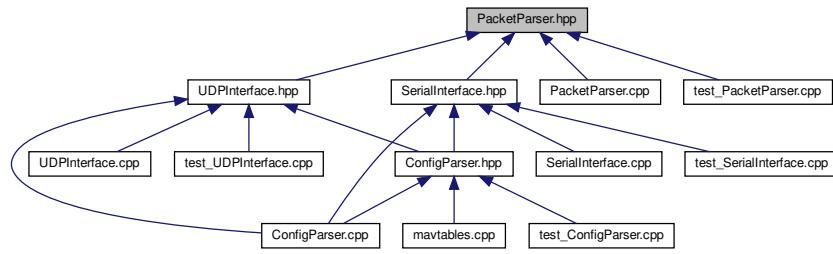
```

#include <memory>
#include "Packet.hpp"
#include "PacketVersion1.hpp"
#include "PacketVersion2.hpp"
Include dependency graph for PacketParser.hpp:

```



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



Classes

- class [PacketParser](#)

16.130 PacketParser.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef PACKETPARSER_HPP_
00019 #define PACKETPARSER_HPP_
00020
00021
00022 #include <memory>
00023
00024 #include "Packet.hpp"
00025 #include "PacketVersion1.hpp"
00026 #include "PacketVersion2.hpp"
00027
00028
00029 /** A MAVLink packet parser.
00030 *
00031 * Parses wire protocol bytes into a MAVLink \ref Packet.
00032 */
00033 class PacketParser
00034 {
00035     public:
00036         PacketParser();
00037         PacketParser(const PacketParser &other) = delete;
00038         PacketParser(PacketParser &&other) = delete;
00039         size_t bytes_parsed() const;
00040         void clear();
00041         std::unique_ptr<Packet> parse_byte(uint8_t byte);
00042         PacketParser &operator=(const PacketParser &other) = delete;
00043         PacketParser &operator=(PacketParser &&other) = delete;
00044
  
```

```

00045     private:
00046         // Types
00047         /** Packet parser states.
00048             */
00049         enum State
00050         {
00051             WAITING_FOR_START_BYTE,    //!< Waiting for a magic start byte.
00052             WAITING_FOR_HEADER,        //!< Waiting for complete header.
00053             WAITING_FOR_PACKET         //!< Waitinf for complete packet.
00054         };
00055         // Variables
00056         std::vector<uint8_t> buffer_;
00057         PacketParser::State state_;
00058         Packet::Version version_;
00059         size_t bytes_remaining_;
00060         // Methods
00061         void waiting_for_start_byte_(uint8_t byte);
00062         void waiting_for_header_(uint8_t byte);
00063         std::unique_ptr<Packet> waiting_for_packet_(uint8_t byte);
00064     };
00065
00066
00067 #endif // PACKETPARSER_HPP_

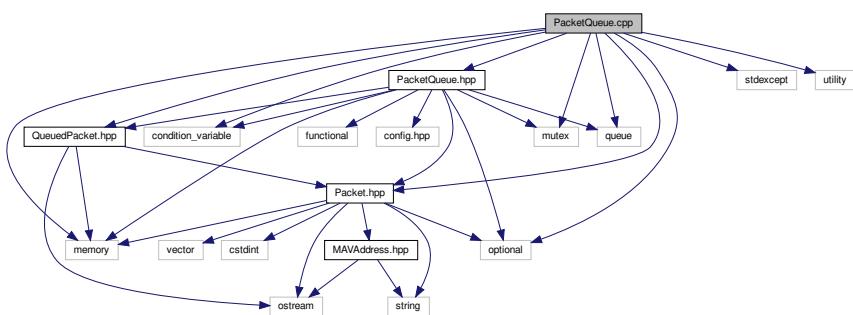
```

16.131 PacketQueue.cpp File Reference

```

#include <condition_variable>
#include <memory>
#include <mutex>
#include <optional>
#include <queue>
#include <stdexcept>
#include <utility>
#include "Packet.hpp"
#include "PacketQueue.hpp"
#include "QueuedPacket.hpp"
Include dependency graph for PacketQueue.cpp:

```



16.132 PacketQueue.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //

```

```
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
0010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
0011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
0012 // GNU General Public License for more details.
0013 //
0014 // You should have received a copy of the GNU General Public License
0015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
0016
0017
0018 #include <condition_variable>
0019 #include <memory>
0020 #include <mutex>
0021 #include <optional>
0022 #include <queue>
0023 #include <stdexcept>
0024 #include <utility>
0025
0026 #include "Packet.hpp"
0027 #include "PacketQueue.hpp"
0028 #include "QueuedPacket.hpp"
0029
0030
0031 /** Get packet from queue.
0032 *
0033 * \note This is an internal method and thus the internal mutex must be locked
0034 * before calling.
0035 *
0036 * \returns The next packet or nullptr if the queue has been closed or is
0037 * empty.
0038 */
0039 std::shared_ptr<const Packet> PacketQueue::get_packet_()
0040 {
0041     if (running_ && !queue_.empty())
0042     {
0043         std::shared_ptr<const Packet> packet = queue_.top().packet();
0044         queue_.pop();
0045         return packet;
0046     }
0047
0048     return nullptr;
0049 }
0050
0051
0052 /** Construct a packet queue.
0053 *
0054 * \param callback A function to call whenever a new packet is added to the
0055 * queue. This allows the queue to signal when it has become non empty.
0056 * The default is no callback {}.
0057 */
0058 PacketQueue::PacketQueue(std::optional<std::function<void(void)>> callback)
0059 : callback_(std::move(callback)), ticket_(0), running_(true)
0060 {
0061 }
0062
0063
0064 /** Close the queue.
0065 *
0066 * This will release any blocking calls to \ref pop.
0067 * \remarks
0068 * Threadsafe (locking).
0069 */
0070 void PacketQueue::close()
0071 {
0072     {
0073         std::lock_guard<std::mutex> lock(mutex_);
0074         running_ = false;
0075     }
0076     cv_.notify_all();
0077 }
0078
0079
0080 /** Determine if the packet queue is empty or not.
0081 *
0082 * \retval true There are no packets in the queue.
0083 * \retval false There is at least one packet in the queue.
0084 * \remarks
```

```

00085     *      Threadsafe (locking).
00086     */
00087     bool PacketQueue::empty()
00088 {
00089     std::lock_guard<std::mutex> lock(mutex_);
00090     return queue_.empty();
00091 }
00092
00093
00094 /** Remove and return the packet at the front of the queue.
00095 *
00096 * This version will block on an empty queue and will not return until the
00097 * queue becomes non empty or is closed with \ref close.
00098 *
00099 * \returns The packet that was at the front of the queue, or nullptr if the
00100 * queue was closed.
00101 * \remarks
00102 * Threadsafe (locking).
00103 * \sa pop(const std::chrono::nanoseconds &)
00104 */
00105 std::shared_ptr<const Packet> PacketQueue::pop()
00106 {
00107     std::unique_lock<std::mutex> lock(mutex_);
00108     // Wait for available packet.
00109     cv_.wait(lock, [this]()
00110     {
00111         return !running_ || !queue_.empty();
00112     });
00113     // Return the packet if the queue is running and is not empty.
00114     return get_packet_();
00115 }
00116
00117
00118 /** Remove and return the packet at the front of the queue.
00119 *
00120 * This version will block on an empty queue and will not return until the
00121 * queue becomes non empty, is closed with \ref close, or the \p timeout has
00122 * expired.
00123 *
00124 * \param timeout How long to block waiting for an empty queue. Set to 0s for
00125 * non blocking.
00126 * \returns The packet that was at the front of the queue, or nullptr if the
00127 * queue was closed or the timeout expired.
00128 * \remarks
00129 * Threadsafe (locking).
00130 * \sa pop()
00131 */
00132 std::shared_ptr<const Packet> PacketQueue::pop(
00133     const std::chrono::nanoseconds &timeout)
00134 {
00135     std::unique_lock<std::mutex> lock(mutex_);
00136
00137     if (timeout > std::chrono::nanoseconds::zero())
00138     {
00139         // Wait for available packet (or the queue to be closed).
00140         cv_.wait_for(lock, timeout, [this]()
00141         {
00142             return !running_ || !queue_.empty();
00143         });
00144     }
00145
00146     // Return the packet if the queue is running and is not empty.
00147     return get_packet_();
00148 }
00149
00150
00151 /** Add a new packet to the queue, with a priority.
00152 *
00153 * A higher \p priority will result in the \p packet being pushed to the front
00154 * of the queue. When priorities are equal the order in which the packets were
00155 * added to the queue is maintained.
00156 *
00157 * \param packet The packet to add to the queue. It must not be nullptr.
00158 * \param priority The priority to use when adding it to the queue. The
00159 * default is 0.
00160 * \throws std::invalid_argument if the packet pointer is null.
00161 * \remarks
00162 * Threadsafe (locking).
00163 */
00164 void PacketQueue::push(std::shared_ptr<const Packet> packet, int priority)
00165 {

```

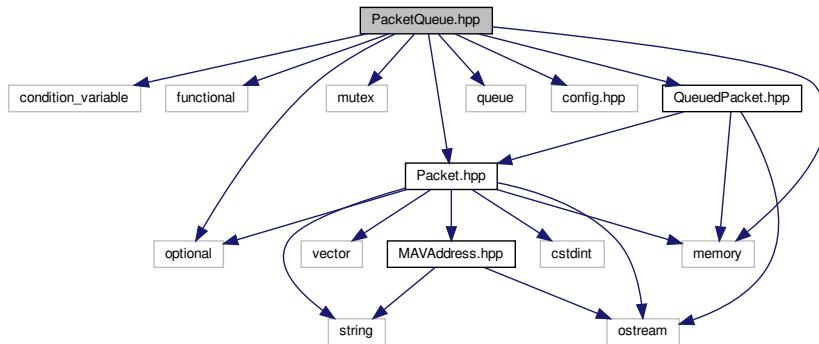
```

00166     if (packet == nullptr)
00167     {
00168         throw std::invalid_argument("Given packet pointer is null.");
00169     }
00170
00171     // Add the packet to the queue.
00172     {
00173         std::lock_guard<std::mutex> lock(mutex_);
00174         queue_.emplace(std::move(packet), priority, ticket_++);
00175     }
00176     // Notify a waiting pop.
00177     cv_.notify_one();
00178
00179     // Trigger the callback.
00180     if (callback_)
00181     {
00182         (*callback_)();
00183     }
00184 }
```

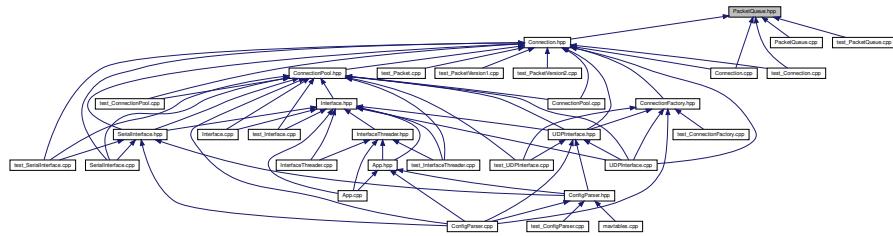
16.133 PacketQueue.hpp File Reference

```

#include <condition_variable>
#include <functional>
#include <memory>
#include <mutex>
#include <optional>
#include <queue>
#include "config.hpp"
#include "Packet.hpp"
#include "QueuedPacket.hpp"
Include dependency graph for PacketQueue.hpp:
```



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



Classes

- class [PacketQueue](#)

16.134 PacketQueue.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef PACKETQUEUE_HPP_
00019 #define PACKETQUEUE_HPP_
00020
00021
00022 #include <condition_variable>
00023 #include <functional>
00024 #include <memory>
00025 #include <mutex>
00026 #include <optional>
00027 #include <queue>
00028
00029 #include "config.hpp"
00030 #include "Packet.hpp"
00031 #include "QueuedPacket.hpp"
00032
00033
00034 /** A threadsafe priority queue for MAVLink packets.
00035 *
00036 * This priority queue will order packets based on priority but also maintains
00037 * insertion order among packets of the same priority.
00038 *
00039 * This is used to implement the packet priority of the firewall and to provide
00040 * a queueing mechanism for packets when consumers are slower than the
00041 * producers.
00042 *
00043 * \sa QueuedPacket
00044 */
00045 class PacketQueue
00046 {
00047     public:
00048         PacketQueue(std::optional<std::function<void(void)>> callback = {});

```

```

00049     // LCOV_EXCL_START
00050     TEST_VIRTUAL ~PacketQueue() = default;
00051     // LCOV_EXCL_STOP
00052     TEST_VIRTUAL void close();
00053     TEST_VIRTUAL bool empty();
00054     TEST_VIRTUAL std::shared_ptr<const Packet> pop();
00055     TEST_VIRTUAL std::shared_ptr<const Packet> pop(
00056         const std::chrono::nanoseconds &timeout);
00057     TEST_VIRTUAL void push(
00058         std::shared_ptr<const Packet> packet, int priority = 0);
00059
00060     private:
00061     // Variables.
00062     std::optional<std::function<void(void)>> callback_;
00063     unsigned long long ticket_;
00064     bool running_;
00065     std::priority_queue<QueuedPacket> queue_;
00066     std::mutex mutex_;
00067     std::condition_variable cv_;
00068     // Methods
00069     std::shared_ptr<const Packet> get_packet_();
00070 };
00071
00072
00073 #endif // PACKETQUEUE_HPP_

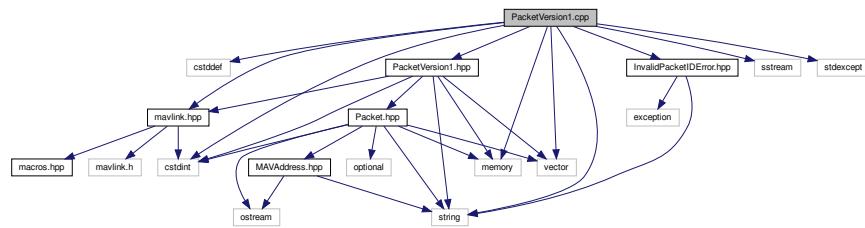
```

16.135 PacketVersion1.cpp File Reference

```

#include <cstdint>
#include <memory>
#include <iostream>
#include <vector>
#include "InvalidPacketIDError.hpp"
#include "mavlink.hpp"
#include "PacketVersion1.hpp"
Include dependency graph for PacketVersion1.cpp:

```



Namespaces

- [packet_v1](#)

16.136 PacketVersion1.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <cstddef>
00019 #include <cstdint>
00020 #include <memory>
00021 #include <iostream>
00022 #include <sstream>
00023 #include <stdexcept>
00024 #include <vector>
00025
00026 #include "InvalidPacketIDError.hpp"
00027 #include "mavlink.hpp"
00028 #include "PacketVersion1.hpp"
00029
00030
00031 namespace packet_v1
00032 {
00033
00034     /** \copydoc ::Packet::Packet(std::vector<uint8_t> data)
00035      *
00036      * \throws std::invalid_argument if packet data does not start with the
00037      * magic byte (0xFE).
00038      * \throws std::length_error if packet data is either too short or too
00039      * long.
00040      */
00041     Packet::Packet(std::vector<uint8_t> data)
00042         : ::Packet(std::move(data))
00043     {
00044         const std::vector<uint8_t> &packet_data = this->data();
00045
00046         // Check that data was given.
00047         if (packet_data.empty())
00048         {
00049             throw std::length_error("Packet is empty.");
00050         }
00051
00052         // Check that a complete header was given (including magic number).
00053         if (!header_complete(packet_data))
00054         {
00055             // Could be the magic number.
00056             if (START_BYTE != packet_data.front())
00057             {
00058                 std::stringstream ss;
00059                 ss << "Invalid packet starting byte (0x"
00060                 << std::uppercase << std::hex
00061                 << static_cast<unsigned int>(packet_data.front())
00062                 << std::nouppercase << "), v1.0 packets should start with 0x"
00063                 << std::uppercase << std::hex
00064                 << static_cast<unsigned int>(START_BYTE)
00065                 << std::nouppercase << ".";
00066                 throw std::invalid_argument(ss.str());
00067             }
00068             // Otherwise the packet is not long enough.
00069             else
00070             {
00071                 throw std::length_error(
00072                     "Packet (" + std::to_string(packet_data.size()) +
00073                     " bytes) is shorter than a v1.0 header (" +
00074                     std::to_string(HEADER_LENGTH) + " bytes).");
00075             }
00076         }
00077     }

```

```
00078     // Verify the message ID.
00079     if (mavlink_get_message_info_by_id(header(packet_data)->msgid) ==
00080         nullptr)
00081     {
00082         throw InvalidPacketIDError(header(packet_data)->msgid);
00083     }
00084
00085     // Ensure a complete packet was given.
00086     if (!packet_complete(packet_data))
00087     {
00088         size_t expected_length =
00089             HEADER_LENGTH + header(packet_data)->len +
00090             CHECKSUM_LENGTH;
00091         throw std::length_error(
00092             "Packet is " + std::to_string(this->data().size()) +
00093             " bytes, should be " +
00094             std::to_string(expected_length) + " bytes.");
00095     }
00096
00097
00098 /** \copydoc ::Packet::version()
00099 *
00100 * \returns 0x0100 (v1.0) - ::Packet::V1
00101 */
00102 ::Packet::Version Packet::version() const
00103 {
00104     return VERSION;
00105 }
00106
00107
00108 unsigned long Packet::id() const
00109 {
00110     return header(data())->msgid;
00111 }
00112
00113
00114 /** \copydoc ::Packet::name()
00115 *
00116 * \throws std::runtime_error if the packet data has an invalid ID.
00117 */
00118 std::string Packet::name() const
00119 {
00120     if (const mavlink_message_info_t *msg_info =
00121         mavlink_get_message_info_by_id(header(data())->msgid))
00122     {
00123         return std::string(msg_info->name);
00124     }
00125
00126     // There should never be any way to reach this point since the message
00127     // ID was checked in the constructor. It is here just in case the
00128     // MAVLink C library has an error in it.
00129     // LCOV_EXCL_START
00130     throw InvalidPacketIDError(header(data())->msgid);
00131     // LCOV_EXCL_STOP
00132 }
00133
00134
00135 MAVAddress Packet::source() const
00136 {
00137     return MAVAddress(header(data())->sysid, header(
00138         data())->compid);
00139 }
00140
00141 /** \copydoc ::Packet::dest()
00142 *
00143 * \thanks The [mavlink-router] (https://github.com/intel/mavlink-router)
00144 * project for an example of how to extract the destination address.
00145 */
00146 std::optional<MAVAddress> Packet::dest() const
00147 {
00148     if (const mavlink_msg_entry_t *msg_entry = mavlink_get_msg_entry(
00149         header(data())->msgid))
00150     {
00151         int dest_system = -1;
00152         int dest_component = 0;
00153
00154         // Extract destination system.
00155         if (msg_entry->flags & MAV_MSG_ENTRY_FLAG_HAVE_TARGET_SYSTEM)
00156     {
```

```

00157         // target_system_ofs is offset from start of payload
00158         size_t offset = msg_entry->target_system_ofs +
00159             sizeof(mavlink::vl_header);
00160         dest_system = data()[offset];
00161     }
00162
00163     // Extract destination component.
00164     if (msg_entry->flags & MAV_MSG_ENTRY_FLAG_HAVE_TARGET_COMPONENT)
00165     {
00166         // target_compoent_ofs is offset from start of payload
00167         size_t offset = msg_entry->target_component_ofs +
00168             sizeof(mavlink::vl_header);
00169         dest_component = data()[offset];
00170     }
00171
00172     // Construct MAVLink address.
00173     if (dest_system >= 0)
00174     {
00175         return MAVAddress(static_cast<unsigned int>(dest_system),
00176                           static_cast<unsigned int>(dest_component));
00177     }
00178
00179     // No destination address.
00180     return {};
00181 }
00182
00183 // There should never be any way to reach this point since the message
00184 // ID was checked in the constructor. It is here just in case the
00185 // MAVLink C library has an error in it.
00186 // LCOV_EXCL_START
00187 throw InvalidPacketIDError(header(data())->msgid);
00188 // LCOV_EXCL_STOP
00189 }
00190
00191 /**
00192 * Determine if the given data contains a complete v1.0 header.
00193 */
00194 * \relates packet_v1::Packet
00195 * \param data The packet data.
00196 * \retval true if \p data contains a complete header (starting with the
00197 *   magic byte).
00198 * \retval false if \p data does not contain a complete v1.0 header.
00199 */
00200 bool header_complete(const std::vector<uint8_t> &data)
00201 {
00202     // We cant use the \ref header function here because that would cause
00203     // infinite recursion.
00204     return (data.size() >= HEADER_LENGTH) &&
00205         (START_BYTE == data.front());
00206 }
00207
00208 /**
00209 * Determine if the given data contains a complete v1.0 packet.
00210 */
00211 * \relates packet_v1::Packet
00212 * \param data The packet data.
00213 * \retval true if \p data contains a complete packet (starting with the
00214 *   magic byte).
00215 * \retval false if \p data does not contain a complete v1.0 packet, or if
00216 *   there is extra bytes in \p data beyond the packet.
00217 */
00218 bool packet_complete(const std::vector<uint8_t> &data)
00219 {
00220     if (header_complete(data))
00221     {
00222         size_t expected_length =
00223             HEADER_LENGTH + header(data)->len +
00224             CHECKSUM_LENGTH;
00225         return data.size() == expected_length;
00226     }
00227
00228     return false;
00229 }
00230
00231 /**
00232 * \relates packet_v1::Packet
00233 * \param data The packet data.
00234 * \returns A pointer to the given data, cast to a v1.0 header structure.
00235 *   %If an incomplete header is given a nullptr will be returned.
00236

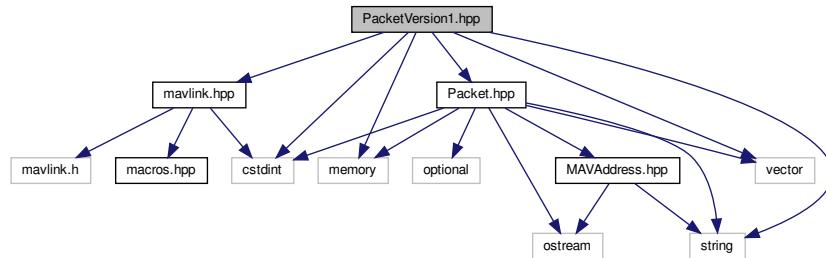
```

```

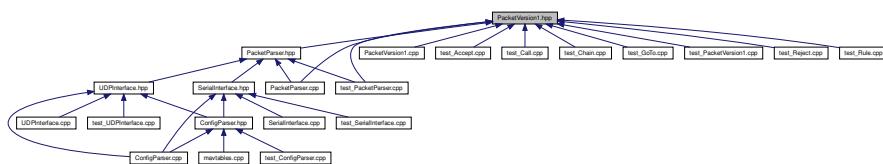
00237     */
00238     const struct mavlink::vl_header *header(
00239         const std::vector<uint8_t> &data)
00240     {
00241         if (header_complete(data))
00242         {
00243             return reinterpret_cast<
00244                 const struct mavlink::vl_header * >(&(data[0]));
00245         }
00246
00247         return nullptr;
00248     }
00249
00250 }
```

16.137 PacketVersion1.hpp File Reference

```
#include <cstdint>
#include <memory>
#include <string>
#include <vector>
#include "mavlink.hpp"
#include "Packet.hpp"
Include dependency graph for PacketVersion1.hpp:
```



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



Classes

- class `packet_v1::Packet`

Namespaces

- `packet_v1`

Variables

- `const uint8_t packet_v1::START_BYTE = MAVLINK_STX_MAVLINK1`
- `const size_t packet_v1::HEADER_LENGTH = 1 + MAVLINK_CORE_HEADER_MAVLINK1_LEN`
- `const size_t packet_v1::CHECKSUM_LENGTH = MAVLINK_NUM_CHECKSUM_BYTES`
- `const ::Packet::Version packet_v1::VERSION = ::Packet::V1`

16.138 PacketVersion1.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef PACKETVERSION1_HPP_
00019 #define PACKETVERSION1_HPP_
00020
00021
00022 #include <cstdint>
00023 #include <memory>
00024 #include <string>
00025 #include <vector>
00026
00027 #include "mavlink.hpp"
00028 #include "Packet.hpp"
00029
00030
00031 namespace packet_v1
00032 {
00033
00034     /** MAVLink v1.0 start byte (0xFE).
00035      */
00036     const uint8_t START_BYTE = MAVLINK_STX_MAVLINK1;
00037
00038
00039     /** MAVLink v1.0 header length (6 bytes).
00040      */
00041     const size_t HEADER_LENGTH = 1 + MAVLINK_CORE_HEADER_MAVLINK1_LEN;
00042
00043
00044     /** MAVLink v1.0 checksum length (2 bytes).
00045      */
00046     const size_t CHECKSUM_LENGTH = MAVLINK_NUM_CHECKSUM_BYTES;
00047
00048
00049     /** MAVLink v1.0 version.
00050      */
00051     const ::Packet::Version VERSION = ::Packet::V1;
00052
00053
00054     /** A MAVLink packet with the version 1 wire protocol.
00055      */
00056     class Packet : public ::Packet

```

```

00057     {
00058         public:
00059             /** Copy constructor.
00060             *
00061             * \param other Packet to copy from.
00062             */
00063             Packet(const Packet &other) = default;
00064             /** Move constructor.
00065             *
00066             * \param other Packet to move from.
00067             */
00068             Packet(Packet &&other) = default;
00069             Packet(std::vector<uint8_t> data);
00070             virtual ::Packet::Version version() const;
00071             virtual unsigned long id() const;
00072             virtual std::string name() const;
00073             virtual MAVAddress source() const;
00074             virtual std::optional<MAVAddress> dest() const;
00075             /** Assignment operator.
00076             *
00077             * \param other Packet to copy from.
00078             */
00079             Packet &operator=(const Packet &other) = default;
00080             /** Assignment operator (by move semantics).
00081             *
00082             * \param other Packet to move from.
00083             */
00084             Packet &operator=(Packet &&other) = default;
00085     };
00086
00087
00088     bool header_complete(const std::vector<uint8_t> &data);
00089     bool packet_complete(const std::vector<uint8_t> &data);
00090     const struct mavlink::vl_header *header(
00091         const std::vector<uint8_t> &data);
00092
00093 }
00094
00095
00096 #endif // PACKETVERSION1_HPP_

```

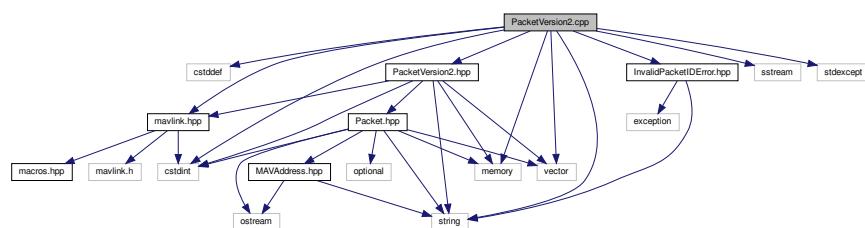
16.139 PacketVersion2.cpp File Reference

```

#include <cstdint>
#include <memory>
#include <iostream>
#include <sstream>
#include <stdexcept>
#include <string>
#include <vector>
#include "InvalidPacketIDError.hpp"
#include "mavlink.hpp"
#include "PacketVersion2.hpp"

```

Include dependency graph for PacketVersion2.cpp:



Namespaces

- [packet_v2](#)

16.140 PacketVersion2.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <cstdint>
00019 #include <memory>
00020 #include <iostream>
00021 #include <sstream>
00022 #include <stdexcept>
00023 #include <string>
00024 #include <vector>
00025
00026 #include "InvalidPacketIDError.hpp"
00027 #include "mavlink.hpp"
00028 #include "PacketVersion2.hpp"
00029
00030
00031 namespace packet_v2
00032 {
00033
00034     /** \copydoc ::Packet::Packet(std::vector<uint8_t> data)
00035      *
00036      * \throws std::invalid_argument if packet data does not start with the
00037      * magic byte (0xFD).
00038      * \throws std::length_error if packet data is either too short or too
00039      * long.
00040      */
00041     Packet::Packet(std::vector<uint8_t> data)
00042         : ::Packet(std::move(data))
00043     {
00044         const std::vector<uint8_t> &packet_data = this->data();
00045
00046         // Check that data was given.
00047         if (packet_data.empty())
00048         {
00049             throw std::length_error("Packet is empty.");
00050         }
00051
00052         // Check that a complete header was given (including magic number).
00053         if (!header_complete(packet_data))
00054         {
00055             // Could be the magic number.
00056             if (START_BYTE != packet_data.front())
00057             {
00058                 std::stringstream ss;
00059                 ss << "Invalid packet starting byte (0x"
00060                 << std::uppercase << std::hex
00061                 << static_cast<unsigned int>(packet_data.front())
00062                 << std::nouppercase << "), v2.0 packets should start with 0x"
00063                 << std::uppercase << std::hex
00064                 << static_cast<unsigned int>(START_BYTE)
00065                 << std::nouppercase << ".";
00066                 throw std::invalid_argument(ss.str());
00067             }
00068             // Otherwise the packet is not long enough.
00069         }
00070     }
00071 }
```

```
00069         else
00070         {
00071             throw std::length_error(
00072                 "Packet (" + std::to_string(packet_data.size()) +
00073                 " bytes) is shorter than a v2.0 header (" +
00074                 std::to_string(HEADER_LENGTH) +
00075                 " bytes).");
00076         }
00077     }
00078
00079     // Verify the message ID.
00080     if (mavlink_get_message_info_by_id(header(packet_data)->msgid) ==
00081         nullptr)
00082     {
00083         throw InvalidPacketIDError(header(packet_data)->msgid);
00084     }
00085
00086     // Ensure a complete packet was given.
00087     if (!packet_complete(packet_data))
00088     {
00089         std::string prefix = "Packet";
00090         size_t expected_length =
00091             HEADER_LENGTH + header(packet_data)->len +
00092             CHECKSUM_LENGTH;
00093
00094         if (is_signed(packet_data))
00095         {
00096             expected_length += SIGNATURE_LENGTH;
00097             prefix = "Signed packet";
00098         }
00099
00100         throw std::length_error(
00101             prefix + " is " + std::to_string(packet_data.size()) +
00102             " bytes, should be " +
00103             std::to_string(expected_length) + " bytes.");
00104     }
00105
00106
00107     /** \copydoc ::Packet::version()
00108     *
00109     * \returns 0x0200 (v2.0) - ::Packet::V2
00110     */
00111     ::Packet::Version Packet::version() const
00112     {
00113         return ::Packet::V2;
00114     }
00115
00116
00117     unsigned long Packet::id() const
00118     {
00119         return header(data())->msgid;
00120     }
00121
00122
00123     /** \copydoc ::Packet::name()
00124     *
00125     * \throws std::runtime_error If the packet data has an invalid ID.
00126     */
00127     std::string Packet::name() const
00128     {
00129         if (const mavlink_message_info_t *msg_info =
00130             mavlink_get_message_info_by_id(header(data())->msgid))
00131         {
00132             return std::string(msg_info->name);
00133         }
00134
00135         // There should never be any way to reach this point since the message
00136         // ID was checked in the constructor. It is here just in case the
00137         // MAVLink C library has an error in it.
00138         // LCOV_EXCL_START
00139         throw InvalidPacketIDError(header(data())->msgid);
00140         // LCOV_EXCL_STOP
00141     }
00142
00143
00144     MAVAddress Packet::source() const
00145     {
00146         return MAVAddress(header(data())->sysid, header(
00147             data())->compid);
00148     }
```

```

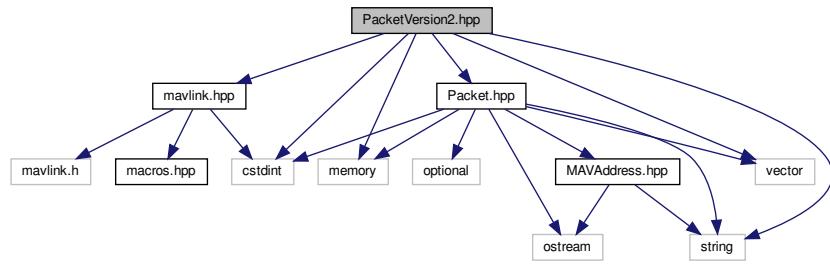
00148
00149
00150     /** \copydoc ::Packet::dest()
00151     *
00152     * \thanks The [mavlink-router] (https://github.com/intel/mavlink-router)
00153     * project for an example of how to extract the destination address.
00154     */
00155     std::optional<MAVAddress> Packet::dest() const
00156     {
00157         if (const mavlink_msg_entry_t *msg_entry =
00158             mavlink_get_msg_entry(header(data())->msgid))
00159         {
00160             int dest_system = -1;
00161             int dest_component = 0;
00162
00163             // Extract destination system.
00164             if (msg_entry->flags & MAV_MSG_ENTRY_FLAG_HAVE_TARGET_SYSTEM)
00165             {
00166                 // Must check to make sure the target system offset is within
00167                 // the packet payload because it can be striped out in v2.0
00168                 // packets if it is 0.
00169                 if (msg_entry->target_system_ofs < header(data())->len)
00170                 {
00171                     // target_system_ofs is offset from start of payload
00172                     size_t offset = msg_entry->target_system_ofs +
00173                         sizeof(mavlink::v2_header);
00174                     dest_system = data()[offset];
00175                 }
00176                 else
00177                 {
00178                     dest_system = 0;
00179                 }
00180             }
00181
00182             // Extract destination component.
00183             if (msg_entry->flags & MAV_MSG_ENTRY_FLAG_HAVE_TARGET_COMPONENT)
00184             {
00185                 // Must check to make sure the target component offset is within
00186                 // the packet payload because it can be striped out in v2.0
00187                 // packets if it is 0.
00188                 if (msg_entry->target_component_ofs < header(data())->len)
00189                 {
00190                     // target_compoent_ofs is offset from start of payload
00191                     size_t offset = msg_entry->target_component_ofs +
00192                         sizeof(mavlink::v2_header);
00193                     dest_component = data()[offset];
00194                 }
00195                 else
00196                 {
00197                     dest_component = 0;
00198                 }
00199             }
00200
00201             // Construct MAVLink address.
00202             if (dest_system >= 0)
00203             {
00204                 return MAVAddress(static_cast<unsigned int>(dest_system),
00205                                 static_cast<unsigned int>(dest_component));
00206             }
00207
00208             // No destination address.
00209             return {};
00210         }
00211
00212         // There should never be any way to reach this point since the message
00213         // ID was checked in the constructor. It is here just in case the
00214         // MAVLink C library has an error in it.
00215         // LCOV_EXCL_START
00216         throw InvalidPacketIDError(header(data())->msgid);
00217         // LCOV_EXCL_STOP
00218     }
00219
00220
00221     /** Determine if a MAVLink v2.0 packet is signed or not.
00222     *
00223     * \relates packet_v2::Packet
00224     * \param data The packet data.
00225     * \throws std::invalid_argument Header is not complete or is invalid.
00226     */
00227     bool is_signed(const std::vector<uint8_t> &data)
00228     {

```

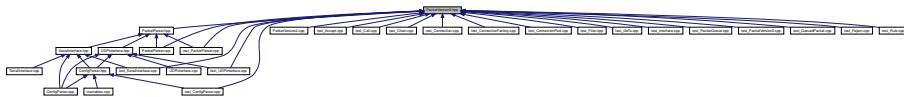
```
00229     // Check that a complete header was given (including magic number).
00230     if (!header_complete(data))
00231     {
00232         throw std::invalid_argument("Header is incomplete or invalid.");
00233     }
00234
00235     return (header(data)->incompat_flags & MAVLINK_IFLAG_SIGNED);
00236 }
00237
00238
00239 /** Determine if the given data contains a complete v2.0 header.
00240 *
00241 * \relates packet_v2::Packet
00242 * \param data The packet data.
00243 * \retval true if \p data contains a complete header (starting with the
00244 *     magic byte).
00245 * \retval false if \p data does not contain a complete v2.0 header.
00246 */
00247 bool header_complete(const std::vector<uint8_t> &data)
00248 {
00249     return (data.size() >= HEADER_LENGTH) &&
00250            (START_BYTE == data.front());
00251 }
00252
00253
00254 /** Determine if the given data contains a complete v2.0 packet.
00255 *
00256 * \relates packet_v2::Packet
00257 * \param data The packet data.
00258 * \retval true if \p data contains a complete packet (starting with the
00259 *     magic byte).
00260 * \retval false if \p data does not contain a complete v1.0 packet, or if
00261 *     there is extra bytes in \p data beyond the packet.
00262 */
00263 bool packet_complete(const std::vector<uint8_t> &data)
00264 {
00265     if (header_complete(data))
00266     {
00267         size_t expected_length =
00268             HEADER_LENGTH + header(data)->len +
00269             CHECKSUM_LENGTH;
00270
00271         if (header(data)->incompat_flags & MAVLINK_IFLAG_SIGNED)
00272         {
00273             expected_length += SIGNATURE_LENGTH;
00274         }
00275
00276         return data.size() == expected_length;
00277     }
00278
00279     return false;
00280 }
00281
00282
00283 /** Cast data as a v2.0 packet header structure pointer.
00284 *
00285 * \relates packet_v2::Packet
00286 * \param data The packet data.
00287 * \returns A pointer to the given data, cast to a v2.0 header structure.
00288 * %If an incomplete header is given a nullptr will be returned.
00289 */
00290 const struct mavlink::v2_header *header(
00291     const std::vector<uint8_t> &data)
00292 {
00293     if (header_complete(data))
00294     {
00295         return reinterpret_cast<
00296                 const struct mavlink::v2_header * >(&(data[0]));
00297     }
00298
00299     return nullptr;
00300 }
00301 }
```

16.141 PacketVersion2.hpp File Reference

```
#include <cstdint>
#include <memory>
#include <string>
#include <vector>
#include "mavlink.hpp"
#include "Packet.hpp"
Include dependency graph for PacketVersion2.hpp:
```



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



Classes

- class [packet_v2::Packet](#)

Namespaces

- [packet_v2](#)

Variables

- const uint8_t [packet_v2::START_BYTE](#) = MAVLINK_STX
- const size_t [packet_v2::HEADER_LENGTH](#) = MAVLINK_NUM_HEADER_BYTES
- const size_t [packet_v2::CHECKSUM_LENGTH](#) = MAVLINK_NUM_CHECKSUM_BYTES
- const size_t [packet_v2::SIGNATURE_LENGTH](#) = MAVLINK_SIGNATURE_BLOCK_LEN
- const ::Packet::Version [packet_v2::VERSION](#) = ::Packet::V2

16.142 PacketVersion2.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef PACKETVERSION2_HPP_
00019 #define PACKETVERSION2_HPP_
00020
00021
00022 #include <cstdint>
00023 #include <memory>
00024 #include <string>
00025 #include <vector>
00026
00027 #include "mavlink.hpp"
00028 #include "Packet.hpp"
00029
00030
00031 namespace packet_v2
00032 {
00033
00034     /** MAVLink v2.0 start byte (0xFD).
00035     */
00036     const uint8_t START_BYTE = MAVLINK_STX;
00037
00038
00039     /** MAVLink v2.0 header length (10 bytes).
00040     */
00041     const size_t HEADER_LENGTH = MAVLINK_NUM_HEADER_BYTES;
00042
00043
00044     /** MAVLink v2.0 checksum length (2 bytes).
00045     */
00046     const size_t CHECKSUM_LENGTH = MAVLINK_NUM_CHECKSUM_BYTES;
00047
00048
00049     /** MAVLink v2.0 signature length (13 bytes) if signed.
00050     */
00051     const size_t SIGNATURE_LENGTH = MAVLINK_SIGNATURE_BLOCK_LEN;
00052
00053
00054     /** MAVLink v2.0 version..
00055     */
00056     const ::Packet::Version VERSION = ::Packet::V2;
00057
00058
00059     /** A MAVLink packet with the version 2 wire protocol.
00060     */
00061     class Packet : public ::Packet
00062     {
00063         public:
00064             /** Copy constructor.
00065             *
00066             * \param other Packet to copy from.
00067             */
00068             Packet(const Packet &other) = default;
00069             /** Move constructor.
00070             *
00071             * \param other Packet to move from.
00072             */
00073             Packet(Packet &&other) = default;
00074             Packet(std::vector<uint8_t> data);
00075             virtual ::Packet::Version version() const;
00076             virtual unsigned long id() const;
00077             virtual std::string name() const;
```

```

00078     virtual MAVAddress source() const;
00079     virtual std::optional<MAVAddress> dest() const;
00080     /** Assignment operator.
00081     *
00082     * \param other Packet to copy from.
00083     */
00084     Packet &operator=(const Packet &other) = default;
00085     /** Assignment operator (by move semantics).
00086     *
00087     * \param other Packet to move from.
00088     */
00089     Packet &operator=(Packet &&other) = default;
00090 };
00091
00092     bool is_signed(const std::vector<uint8_t> &data);
00093     bool header_complete(const std::vector<uint8_t> &data);
00094     bool packet_complete(const std::vector<uint8_t> &data);
00095     const struct mavlink::v2_header *header(
00096         const std::vector<uint8_t> &data);
00097
00098 }
00099
00100 #endif // PACKETVERSION2_HPP_

```

16.143 parse_tree.hpp File Reference

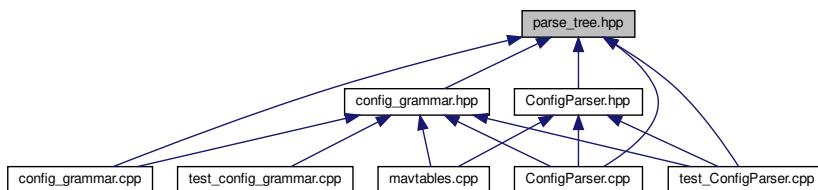
```

#include <cassert>
#include <memory>
#include <typeinfo>
#include <type_traits>
#include <utility>
#include <vector>
#include <pegtl/config.hpp>
#include <pegtl/internal/demangle.hpp>
#include <pegtl/internal/iterator.hpp>
#include <pegtl/normal.hpp>
#include <pegtl/nothing.hpp>
#include <pegtl/parse.hpp>
Include dependency graph for parse_tree.hpp:

```



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



16.144 parse_tree.hpp

```
00001 // Copyright (c) 2007-2018 Dr. Colin Hirsch and Daniel Frey
00002 //
00003 // Permission is hereby granted, free of charge, to any person obtaining a copy
00004 // of this software and associated documentation files (the "Software"), to deal
00005 // in the Software without restriction, including without limitation the rights
00006 // to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00007 // copies of the Software, and to permit persons to whom the Software is
00008 // furnished to do so, subject to the following conditions:
00009 //
00010 // The above copyright notice and this permission notice shall be included in
00011 // all copies or substantial portions of the Software.
00012 //
00013 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00014 // IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00015 // FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00016 // AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00017 // LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00018 // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00019 // SOFTWARE.
00020
00021
00022 // Modified by Michael R. Shannon for use in mavtables.
00023
00024
00025 #ifndef PARSE_TREE_HPP_
00026 #define PARSE_TREE_HPP_
00027
00028
00029 #include <cassert>
00030 #include <memory>
00031 #include <typeinfo>
00032 #include <type_traits>
00033 #include <utility>
00034 #include <vector>
00035
00036 #include <pegtl/config.hpp>
00037 #include <pegtl/internal/demangle.hpp>
00038 #include <pegtl/internal/iterator.hpp>
00039 #include <pegtl/normal.hpp>
00040 #include <pegtl/nothing.hpp>
00041 #include <pegtl/parse.hpp>
00042
00043
00044 // Not included in documentation because it is mostly copied from PEGTL.
00045
00046 /// @cond INTERNAL
00047 namespace config
00048 {
00049     namespace parse_tree
00050     {
00051         namespace pgtl = tao::pgtl;
00052
00053         template<typename T>
00054         struct default_node_children
00055         {
00056             using node_t = T;
00057             using children_t = std::vector< std::unique_ptr< node_t > >;
00058             children_t children;
00059
00060             // each node will be default constructed
00061             default_node_children() = default;
00062
00063             // no copy/move is necessary
00064             // (nodes are always owned/handled by a std::unique_ptr)
00065             default_node_children(const default_node_children &) = delete;
00066             default_node_children(default_node_children &&) = delete;
00067
00068             ~default_node_children() = default;
00069
00070             // no assignment either
00071             default_node_children &operator=(const default_node_children &) = delete;
00072             default_node_children &operator=(default_node_children &&) = delete;
00073
00074             typename children_t::reference at(
00075                 const typename children_t::size_type pos)
```

```
00078      {
00079          return children.at(pos);
00080      }
00081
00082      typename children_t::const_reference at(
00083          const typename children_t::size_type pos) const
00084      {
00085          return children.at(pos);
00086      }
00087
00088      typename children_t::reference front()
00089      {
00090          return children.front();
00091      }
00092
00093      typename children_t::const_reference front() const
00094      {
00095          return children.front();
00096      }
00097
00098      typename children_t::reference back()
00099      {
00100          return children.back();
00101      }
00102
00103      typename children_t::const_reference back() const
00104      {
00105          return children.back();
00106      }
00107
00108      bool empty() const noexcept
00109      {
00110          return children.empty();
00111      }
00112
00113      typename children_t::size_type size() const noexcept
00114      {
00115          return children.size();
00116      }
00117
00118      // if parsing succeeded and the (optional) transform call
00119      // did not discard the node, it is appended to its parent.
00120      // note that "child" is the node whose Rule just succeeded
00121      // and "*this" is the parent where the node should be appended.
00122      template<typename... States>
00123      void emplace_back(
00124          std::unique_ptr<node_t> child, States &&... /*unused*/)
00125      {
00126          assert(child);
00127          children.emplace_back(std::move(child));
00128      }
00129  };
00130
00131  struct default_node_content
00132  {
00133      const std::type_info *id_ = nullptr;
00134      pegtl::internal::iterator begin_;
00135      pegtl::internal::iterator end_;
00136      std::string source_;
00137
00138      bool is_root() const noexcept
00139      {
00140          return id_ == nullptr;
00141      }
00142
00143      template<typename T>
00144      bool is() const noexcept
00145      {
00146          return id_ == &typeid(T);
00147      }
00148
00149      std::string name() const
00150      {
00151          assert(!is_root());
00152          return pegtl::internal::demangle(id_->name());
00153      }
00154
00155      pegtl::position begin() const
00156      {
00157          return pegtl::position(begin_, source_);
00158      }
```

```
00159     pegtl::position end() const
00160     {
00161         return pegtl::position(end_, source_);
00162     }
00163
00164     const std::string &source() const noexcept
00165     {
00166         return source_;
00167     }
00168
00169     bool has_content() const noexcept
00170     {
00171         return end_.data != nullptr;
00172     }
00173
00174     std::string content() const
00175     {
00176         assert(has_content());
00177         return std::string(begin_.data, end_.data);
00178     }
00179
00180     template< typename... States >
00181     void remove_content(States &&... /*unused*/) noexcept
00182     {
00183         end_.reset();
00184     }
00185
00186 // all non-root nodes are initialized by calling this method
00187 template<typename Rule, typename Input, typename... States>
00188 void start(const Input &in, States &&... /*unused*/)
00189 {
00190     id_ = &typeid(Rule);
00191     begin_ = in.iterator();
00192     source_ = in.source();
00193 }
00194
00195 // if parsing of the rule succeeded, this method is called
00196 template<typename Rule, typename Input, typename... States>
00197 void success(const Input &in, States &&... /*unused*/) noexcept
00198 {
00199     end_ = in.iterator();
00200 }
00201
00202 // if parsing of the rule failed, this method is called
00203 template<typename Rule, typename Input, typename... States>
00204 void failure(
00205     const Input & /*unused*/, States &&... /*unused*/) noexcept
00206 {
00207 }
00208
00209 };
00210
00211 template<typename T>
00212 struct basic_node
00213     : default_node_children<T>, default_node_content
00214 {
00215 };
00216
00217 struct node
00218     : basic_node< node >
00219 {
00220 };
00221
00222 namespace internal
00223 {
00224     template<typename Node>
00225     struct state
00226     {
00227         std::vector<std::unique_ptr<Node>> stack;
00228
00229         state()
00230         {
00231             emplace_back();
00232         }
00233
00234         void emplace_back()
00235         {
00236             stack.emplace_back(std::unique_ptr< Node >(new Node));
00237         }
00238
00239         std::unique_ptr<Node> &back() noexcept
```

```

00240         {
00241             return stack.back();
00242         }
00243
00244         void pop_back() noexcept
00245         {
00246             return stack.pop_back();
00247         }
00248     };
00249
00250     template<typename Node, typename S, typename = void>
00251     struct transform
00252     {
00253         template<typename... States>
00254         static void call(std::unique_ptr<Node> & /*unused*/,
00255                           States &&... /*unused*/) noexcept
00256         {
00257         }
00258     };
00259
00260     template<typename Node, typename S>
00261     struct transform <
00262         Node, S, decltype(
00263             S::transform(std::declval<std::unique_ptr<Node>&>()),
00264             void())
00265         {
00266         template<typename... States>
00267         static void call(
00268             std::unique_ptr<Node> &n, States &&... st)
00269             noexcept(noexcept(S::transform(n)))
00270         {
00271             S::transform(n, st...);
00272         }
00273     };
00274
00275 // https://stackoverflow.com/a/16000226
00276 template <typename T, typename = int>
00277 struct has_error_message : std::false_type {};
00278
00279 // https://stackoverflow.com/a/16000226
00280 template <typename T>
00281 struct has_error_message <T, decltype((void) T::error_message, 0)>
00282 : std::true_type {};
00283
00284     template<template<typename> class S>
00285     struct make_control
00286     {
00287         template<typename Rule, bool = S<Rule>::value>
00288         struct control;
00289
00290         template<typename Rule>
00291         using type = control<Rule>;
00292     };
00293
00294     template<template<typename> class S>
00295     template<typename Rule>
00296     struct make_control< S >::control< Rule, false >
00297     : pegtl::normal< Rule >
00298     {
00299
00300 // custom error message
00301     template< typename Input, typename... States >
00302     static void raise(const Input &in, States &&...)
00303     {
00304         if constexpr(has_error_message<S<Rule>>::value)
00305         {
00306             throw pegtl::parse_error(S<Rule>::error_message, in);
00307         }
00308         else
00309         {
00310             throw pegtl::parse_error(
00311                 "Parse error matching " +
00312                 pegtl::internal::demangle<Rule>(), in);
00313         }
00314     };
00315
00316     template<template<typename> class S>
00317     template<typename Rule>
00318     struct make_control<S>::control< Rule, true >
00319     : pegtl::normal<Rule>

```

```
00321         {
00322             // custom error message
00323             template< typename Input, typename... States >
00324             static void raise(const Input &in, States &&...)
00325             {
00326                 if constexpr(has_error_message<S<Rule>>::value)
00327                 {
00328                     throw pegtl::parse_error(S<Rule>::error_message, in);
00329                 }
00330                 else
00331                 {
00332                     throw pegtl::parse_error(
00333                         "Parse error matching " +
00334                         pegtl::internal::demangle<Rule>(), in);
00335                 }
00336             }
00337         }
00338
00339         template<typename Input, typename Node, typename... States>
00340         static void start(
00341             const Input &in, state< Node > &state, States &&... st)
00342         {
00343             state.emplace_back();
00344             state.back()->template start<Rule>(in, st...);
00345         }
00346
00347         template< typename Input, typename Node, typename... States >
00348         static void success(
00349             const Input &in, state<Node> &state, States &&... st)
00350         {
00351             auto n = std::move(state.back());
00352             state.pop_back();
00353             n->template success<Rule>(in, st...);
00354             transform<Node, S<Rule>>::call(n, st...);
00355
00356             if (n)
00357             {
00358                 state.back()->emplace_back(std::move(n), st...);
00359             }
00360         }
00361
00362         template<typename Input, typename Node, typename... States>
00363         static void failure(
00364             const Input &in, state< Node > &state, States &&... st)
00365             noexcept(noexcept(std::declval<node &>().template
00366                             failure<Rule>(in, st...)))
00367         {
00368             state.back()->template failure< Rule >(in, st...);
00369             state.pop_back();
00370         }
00371     };
00372
00373     template<typename>
00374     struct store_all : std::true_type
00375     {
00376     };
00377
00378 } // namespace internal
00379
00380 template <
00381     typename Rule,
00382     typename Node,
00383     template<typename> class S = internal::store_all,
00384     typename Input,
00385     typename... States >
00386     std::unique_ptr<Node> parse(Input &in, States && ... st)
00387     {
00388         internal::state< Node > state;
00389
00390         if (!pegtl::parse<
00391             Rule, pegtl::nothing,
00392             internal::make_control<S>::template type >
00393             (in, state, st...))
00394         {
00395             return nullptr;
00396         }
00397
00398         assert(state.stack.size() == 1);
00399         return std::move(state.back());
00400     }
00401 }
```

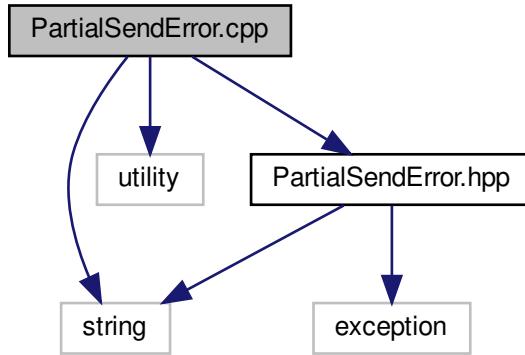
```

00402     template <
00403         typename Rule,
00404         template<typename> class S = internal::store_all,
00405         typename Input,
00406         typename... States >
00407     std::unique_ptr<node> parse(Input &in, States && ... st)
00408     {
00409         return parse<Rule, node, S>(in, st...);
00410     }
00411 }
00412 // namespace parse_tree
00413
00414 } // namespace config
00415 /// @endcond
00416
00417 #endif

```

16.145 PartialSendError.cpp File Reference

```
#include <string>
#include <utility>
#include "PartialSendError.hpp"
Include dependency graph for PartialSendError.cpp:
```



16.146 PartialSendError.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
0010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
0011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
0012 // GNU General Public License for more details.

```

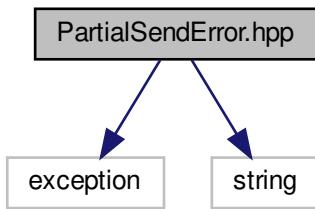
```

00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <string>
00019 #include <utility>
00020
00021 #include "PartialSendError.hpp"
00022
00023
00024 /** Construct a \ref PartialSendError.
00025 *
00026 * \param bytes_sent Number of bytes that were sent.
00027 * \param total_bytes Total number of bytes in the packet.
00028 */
00029 PartialSendError::PartialSendError(
00030     unsigned long bytes_sent, unsigned long total_bytes)
00031 {
00032     message_ = "Could only write " + std::to_string(bytes_sent) +
00033                 " of " + std::to_string(total_bytes) + " bytes.";
00034 }
00035
00036
00037 /** Return error message string.
00038 *
00039 * \returns Error message string containing sent to total bytes ratio.
00040 */
00041 const char *PartialSendError::what() const noexcept
00042 {
00043     return message_.c_str();
00044 }

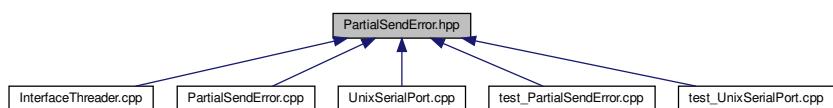
```

16.147 PartialSendError.hpp File Reference

```
#include <exception>
#include <string>
Include dependency graph for PartialSendError.hpp:
```



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



Classes

- class [PartialSendError](#)

16.148 PartialSendError.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef PARTIALSENDERROR_HPP_
00019 #define PARTIALSENDERROR_HPP_
00020
00021
00022 #include <exception>
00023 #include <string>
00024
00025
00026 /** Exception type emitted when an interface fails to send a complete packet.
00027 */
00028 class PartialSendError : public std::exception
00029 {
00030     public:
00031         PartialSendError(unsigned long bytes_sent, unsigned long total_bytes);
00032         const char *what() const noexcept;
00033
00034     private:
00035         std::string message_;
00036 };
00037
00038
00039 #endif // PARTIALSENDRROR_HPP_

```

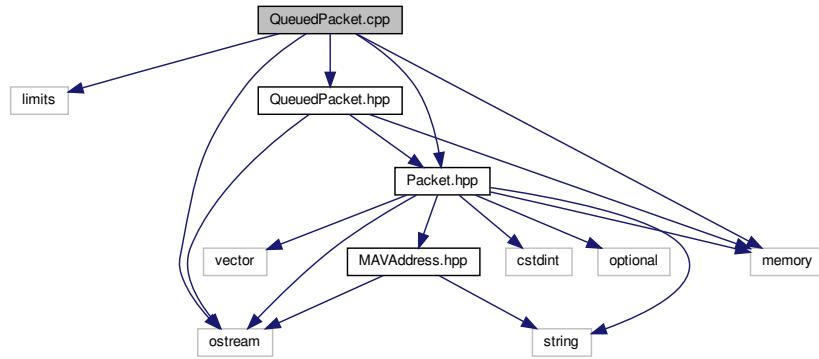
16.149 QueuedPacket.cpp File Reference

```

#include <limits>
#include <memory>
#include <ostream>
#include "Packet.hpp"
#include "QueuedPacket.hpp"

```

Include dependency graph for QueuedPacket.cpp:



16.150 QueuedPacket.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <limits>
00019 #include <memory>
00020 #include <iostream>
00021
00022 #include "Packet.hpp"
00023 #include "QueuedPacket.hpp"
00024
00025
00026 /** Construct a queued packet.
00027 */
00028 * \param packet The packet to store in the queue.
00029 * \param priority The priority to send the packet with, higher numbers result
00030 *      in a higher priority.
00031 * \param ticket_number A number that should always be incremented for each
00032 *      queued packet created per packet queue, even if this increment causes an
00033 *      unsigned integer wraparound.
00034 * \throws std::invalid_argument if the given packet pointer is nullptr.
00035 */
00036 QueuedPacket::QueuedPacket(
00037     std::shared_ptr<const Packet> packet, int priority,
00038     unsigned long long ticket_number)
00039 : packet_(std::move(packet)), priority_(priority),
00040   ticket_number_(ticket_number)
00041 {
00042     if (packet_ == nullptr)
00043     {
00044         throw std::invalid_argument("Given packet pointer is null.");
00045     }
00046 }

```

```

00047
00048
00049      /** Return the contained packet.
00050      *
00051      * \returns The contained MAVLink packet.
00052      */
00053 std::shared_ptr<const Packet> QueuedPacket::packet() const
00054 {
00055     return packet_;
00056 }
00057
00058
00059      /** Equality comparison.
00060      *
00061      * \note It should never be the case that two queued packets have the same
00062      * ticket number and thus two queued packets should never be equal.
00063      *
00064      * \relates QueuedPacket
00065      * \param lhs The left hand side queued packet.
00066      * \param rhs The right hand side queued packet.
00067      * \retval true if \p lhs and \p rhs have the same priority and ticket number.
00068      * \retval false if \p lhs and \p rhs do not have the same priority and
00069      * ticket number.
00070      */
00071 bool operator==(const QueuedPacket &lhs, const
00072     QueuedPacket &rhs)
00073 {
00074     return (lhs.priority_ == rhs.priority_) &&
00075         (lhs.ticket_number_ == rhs.ticket_number_);
00076 }
00077
00078      /** Inequality comparison.
00079      *
00080      * \note It should never be the case that two queued packets have the same
00081      * ticket number and thus two queued packets should never be equal.
00082      *
00083      * \relates QueuedPacket
00084      * \param lhs The left hand side queued packet.
00085      * \param rhs The right hand side queued packet.
00086      * \retval true if \p lhs and \p rhs do not have the same priority and ticket
00087      * number.
00088      * \retval false if \p lhs and \p rhs have the same priority and ticket number.
00089      */
00090 bool operator!=(const QueuedPacket &lhs, const
00091     QueuedPacket &rhs)
00092 {
00093     return (lhs.priority_ != rhs.priority_) ||  

00094         (lhs.ticket_number_ != rhs.ticket_number_);
00095 }
00096
00097      /** Less than comparison.
00098      *
00099      * The priority is considered first, followed by the ticket number in reverse
00100      * order (lower ticket number is greater).
00101      *
00102      * \note The ticket number is considered to be a wrapping integer and thus
00103      * numbers that are within 'std::numeric_limits<unsigned long
00104      * long>::max()/2' of each other are considered in the same range. In
00105      * this way 0 is greater than 'std::numeric_limits<unsigned long
00106      * long>::max()'. Because of this it is important that anything relying
00107      * on ordering must not contain a range of ticket numbers equal to or
00108      * grater than 'std::numeric_limits<unsigned long long>::max()/2'.
00109      *
00110      * \relates QueuedPacket
00111      * \param lhs The left hand side queued packet.
00112      * \param rhs The right hand side queued packet.
00113      * \retval true if \p lhs is less than \p rhs.
00114      * \retval false if \p lhs is not less than \p rhs.
00115      */
00116 bool operator<(const QueuedPacket &lhs, const QueuedPacket &rhs)
00117 {
00118     return (lhs.priority_ < rhs.priority_) || (lhs.priority_ == rhs.priority_ &&  

00119         (rhs.ticket_number_ - lhs.ticket_number_ >  

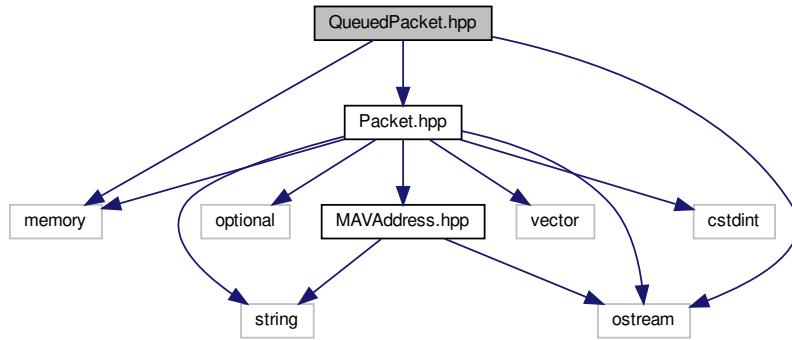
00120             std::numeric_limits<unsigned long long>::max() / 2));
00121 }
00122
00123
00124      /** Greater than comparison.
00125      *

```

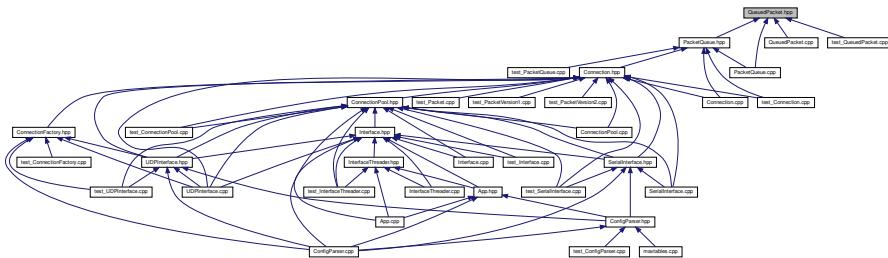
```
00126 * The priority is considered first, followed by the ticket number in reverse
00127 * order (lower ticket number is greater).
00128 *
00129 * \note The ticket number is considered to be a wrapping integer and thus
00130 * numbers that are within `std::numeric_limits<unsigned long
00131 * long>::max()/2` of each other are considered in the same range. In
00132 * this way 0 is greater than `std::numeric_limits<unsigned long
00133 * long>::max()`. Because of this it is important that anything relying
00134 * on ordering must not contain a range of ticket numbers equal to or
00135 * grater than `std::numeric_limits<unsigned long long>::max()/2`.
00136 *
00137 * \relates QueuedPacket
00138 * \param lhs The left hand side queued packet.
00139 * \param rhs The right hand side queued packet.
00140 * \retval true if \p lhs is less than \p rhs.
00141 * \retval false if \p lhs is not less than \p rhs.
00142 */
00143 bool operator>(const QueuedPacket &lhs, const QueuedPacket &rhs)
00144 {
00145     return (lhs.priority_ > rhs.priority_) || (lhs.priority_ == rhs.priority_ &&
00146         (lhs.ticket_number_ - rhs.ticket_number_ >
00147          std::numeric_limits<unsigned long long>::max() / 2));
00148 }
00149
00150
00151 /** Less than or equal comparison.
00152 *
00153 * \relates QueuedPacket
00154 * \param lhs The left hand side queued packet.
00155 * \param rhs The right hand side queued packet.
00156 * \retval true if \p lhs is less than or eqaul to \p rhs.
00157 * \retval false if \p lhs is greater than \p rhs.
00158 */
00159 bool operator<=(const QueuedPacket &lhs, const
    QueuedPacket &rhs)
00160 {
00161     return lhs == rhs || lhs < rhs;
00162 }
00163
00164
00165 /** Greater than or equal comparison.
00166 *
00167 * \relates QueuedPacket
00168 * \param lhs The left hand side queued packet.
00169 * \param rhs The right hand side queued packet.
00170 * \retval true if \p lhs is greater than or equal to \p rhs.
00171 * \retval false if \p lhs is less than \p rhs.
00172 */
00173 bool operator>=(const QueuedPacket &lhs, const
    QueuedPacket &rhs)
00174 {
00175     return lhs == rhs || lhs > rhs;
00176 }
00177
00178
00179 /** Print the packet to the given output stream.
00180 *
00181 * Some examples are:
00182 * - 'HEARTBEAT (#1) from 16.8 (v1.0) with priority -3'
00183 * - 'PING (#4) from 128.4 to 16.8 (v2.0) with priority 0'
00184 * - 'DATA_TRANSMISSION_HANDSHAKE (#130) from 16.8 (v2.0) with priority 3'
00185 * - 'ENCAPSULATED_DATA (#131) from 128.4 (v2.0) with priority 1'
00186 *
00187 * \relates QueuedPacket
00188 * \param os The output stream to print to.
00189 * \param queued_packet The queued packet to print.
00190 * \returns The output stream.
00191 */
00192 std::ostream &operator<<(std::ostream &os, const QueuedPacket &queued_packet)
00193 {
00194     os << *queued_packet.packet_;
00195     os << " with priority " << queued_packet.priority_;
00196     return os;
00197 }
```

16.151 QueuedPacket.hpp File Reference

```
#include <memory>
#include <iostream>
#include "Packet.hpp"
Include dependency graph for QueuedPacket.hpp:
```



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



Classes

- class QueuedPacket

Functions

- bool `operator==` (const `QueuedPacket` &lhs, const `QueuedPacket` &rhs)
 - bool `operator!=` (const `QueuedPacket` &lhs, const `QueuedPacket` &rhs)
 - bool `operator<` (const `QueuedPacket` &lhs, const `QueuedPacket` &rhs)
 - bool `operator>` (const `QueuedPacket` &lhs, const `QueuedPacket` &rhs)
 - bool `operator<=` (const `QueuedPacket` &lhs, const `QueuedPacket` &rhs)
 - bool `operator>=` (const `QueuedPacket` &lhs, const `QueuedPacket` &rhs)
 - std::ostream & `operator<<` (std::ostream &os, const `QueuedPacket` &queued_packet)

16.151.1 Function Documentation

16.151.1.1 operator"!=()

```
bool operator!= (
    const QueuedPacket & lhs,
    const QueuedPacket & rhs )
```

16.151.1.2 operator<()

```
bool operator< (
    const QueuedPacket & lhs,
    const QueuedPacket & rhs )
```

16.151.1.3 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const QueuedPacket & queued_packet )
```

16.151.1.4 operator<=()

```
bool operator<= (
    const QueuedPacket & lhs,
    const QueuedPacket & rhs )
```

16.151.1.5 operator==()

```
bool operator== (
    const QueuedPacket & lhs,
    const QueuedPacket & rhs )
```

16.151.1.6 operator>()

```
bool operator> (
    const QueuedPacket & lhs,
    const QueuedPacket & rhs )
```

16.151.1.7 operator>=()

```
bool operator>= (
    const QueuedPacket & lhs,
    const QueuedPacket & rhs )
```

16.152 QueuedPacket.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef QUEUEDPACKET_HPP_
00019 #define QUEUEDPACKET_HPP_
00020
00021
00022 #include <memory>
00023 #include <iostream>
00024
00025 #include "Packet.hpp"
00026
00027
00028 /** A packet in the queue to be sent out.
00029 *
00030 * Forms a node in the \ref PacketQueue class.
00031 *
00032 * This is the data structure used in the priority queues used by the \ref
00033 * Connection class. It stores a MAVLink packet as well as a priority and
00034 * ticket number used to maintain packet order in the priority queue when
00035 * packets have the same priority.
00036 *
00037 * \sa PacketQueue
00038 */
00039 class QueuedPacket
00040 {
00041     public:
00042         /** Copy constructor.
00043         *
00044         * \param other QueuedPacket to copy from.
00045         */
00046         QueuedPacket(const QueuedPacket &other) = default;
00047         /** Move constructor.
00048         *
00049         * \param other QueuedPacket to move from.
00050         */
```

```

00051     QueuedPacket (QueuedPacket &&other) = default;
00052     QueuedPacket(
00053         std::shared_ptr<const Packet> packet, int priority,
00054         unsigned long long ticket_number);
00055     std::shared_ptr<const Packet> packet() const;
00056     /** Assignment operator.
00057     *
00058     * \param other QueuedPacket to copy from.
00059     */
00060     QueuedPacket &operator=(const QueuedPacket &other) = default;
00061     /** Assignment operator (by move semantics).
00062     *
00063     * \param other QueuedPacket to move from.
00064     */
00065     QueuedPacket &operator=(QueuedPacket &&other) = default;
00066
00067     friend bool operator==(const QueuedPacket &lhs, const QueuedPacket &rhs);
00068     friend bool operator!=(const QueuedPacket &lhs, const QueuedPacket &rhs);
00069     friend bool operator<(const QueuedPacket &lhs, const QueuedPacket &rhs);
00070     friend bool operator>(const QueuedPacket &lhs, const QueuedPacket &rhs);
00071     friend bool operator<=(const QueuedPacket &lhs, const QueuedPacket &rhs);
00072     friend bool operator>=(const QueuedPacket &lhs, const QueuedPacket &rhs);
00073     friend std::ostream &operator<<(std::ostream &os, const QueuedPacket &queued_packet);
00074
00075 private:
00076     std::shared_ptr<const Packet> packet_;
00077     int priority_;
00078     unsigned long long ticket_number_;
00079 };
00080
00081 bool operator==(const QueuedPacket &lhs, const QueuedPacket &rhs);
00082 bool operator!=(const QueuedPacket &lhs, const QueuedPacket &rhs);
00083 bool operator<(const QueuedPacket &lhs, const QueuedPacket &rhs);
00084 bool operator>(const QueuedPacket &lhs, const QueuedPacket &rhs);
00085 bool operator<=(const QueuedPacket &lhs, const QueuedPacket &rhs);
00086 bool operator>=(const QueuedPacket &lhs, const QueuedPacket &rhs);
00087 std::ostream &operator<<(std::ostream &os, const QueuedPacket &queued_packet);
00088
00089 #endif // QUEUEDPACKET_HPP_

```

16.153 README.md File Reference

16.154 README.md

```

00001 mavtables {#readme}
00002 =====
00003
00004 ![[Build
00005   Status](https://travis-ci.org/shamuproject/mavtables.svg?branch=master)](https://travis-ci.org/shamuproject/mavtables) ![[Status](https://coveralls.io/repos/github/shamuproject/mavtables/badge.svg?branch=master)](https://coveralls.io/github/shamuproject/mavtables)
00006 A MAVLink router and firewall. It can connect 2 or more MAVLink endpoints such
00007 as autopilots, ground control software, loggers, image capture systems, etc over
00008 serial and UDP. MAVLink packets will be routed to specific components when they
00009 have a destination address. Any packet, targeted or broadcasted, can be
00010 filtered based on source system/component, destination system/component and
00011 message type. The filter can also apply a priority to packets allowing more
00012 important packets to take priority over lower priority packets when an endpoint
00013 is choked.
00014

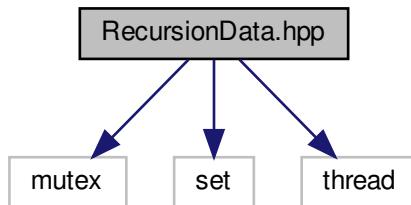
```

```
00015
00016 ## Links
00017
00018 * [User Documentation] (doc/user_manual.md)
00019 * [HTML Developer Documentation] (https://shamuproject.github.io/mavtables)
00020 * [PDF Developer Documentation] (https://shamuproject.github.io/mavtables/mavtables.pdf)
00021 * [Download] (https://github.com/shamuproject/mavtables/archive/master.zip)
00022 * [GitHub] (https://github.com/shamuproject/mavtables)
00023 * [Theory of MAVLink Routing] (http://ardupilot.org/dev/docs/mavlink-routing-in-ardupilot.html)
00024
00025
00026 ## Compilation and Installation
00027
00028 In order to compile you will need the following packages:
00029
00030 * GCC 7+ or Clang 5+ (needed for C++17 support)
00031 * [CMake v3.3+] (https://cmake.org/)
00032 * [Boost v1.54+] (https://www.boost.org/)
00033
00034 __Clang is recommended when contributing to mavtables as it's warnings are
00035 more comprehensive. However, GCC must be used when generating code coverage
00036 reports.__
00037
00038 The following packages are only needed for development work:
00039
00040 * [Artistic Style] (http://astyle.sourceforge.net/) (used for
00041   checking/fixing the code style)
00042 * [Gcovr] (http://gcovr.com/) (coverage report)
00043 * [LCOV] (http://ltp.sourceforge.net/coverage/lcov.php) (detailed coverage html
00044   report)
00045 * [socat] (http://www.dest-unreach.org/socat/) (for testing serial port
00046   communications)
00047
00048 'mavtables' can be easily installed using the standard procedure of
00049 ``
00050 $ make
00051 # make install
00052 ``
00053 The installation prefix is '/usr/local' by default but can be changed with
00054 ``
00055 $ make
00056 # make PREFIX=/desired/install/path install
00057 ``
00058 The makefile will download and use the default MAVLink implementation with the
00059 ArduPilot dialect. This can be overridden by setting the 'MDIR' environment
00060 variable to the library path (containing the 'DIALECT') and/or the 'DIALECT'
00061 environment variable to the MAVLink dialect to use. For instance the default
00062 value of 'DIALECT' is 'ardupilotmega'.
00063
00064
00065 ## Running
00066
00067 To run mavtables and begin routing packets
00068 ``
00069 $ mavtables
00070 ``
00071 This will use the first configuration file it finds in the configuration file
00072 priority order given in the next section. To force a specific configuration
00073 file the '--config' flag may be used.
00074 ``
00075 $ mavtables --config <path/to/config>
00076 ``
00077 The inbuilt help may be accessed with the '-h' or '--help' flags.
00078 ``
00079 $ mavtables --help
00080 usage: mavtables:
00081   -h [ --help ]          print this message
00082   --config arg           specify configuration file
00083   --ast                  print AST of configuration file (do not run)
00084   --version              print version and license information
00085   --loglevel arg         level of logging, between 0 and 3
00086 ``
00087
00088
00089 ## Configuration File
00090
00091 Both interfaces and filter rules are defined in a configuration file. The
00092 format of this configuration file is documented in
00093 ['doc/configuration.md'] (doc/configuration.md) and an example is located at
00094 'examples/mavtables.conf' which is the same file that is installed at
00095 '/etc/mavtables.conf' when using 'make install'. The configuration file used is
```

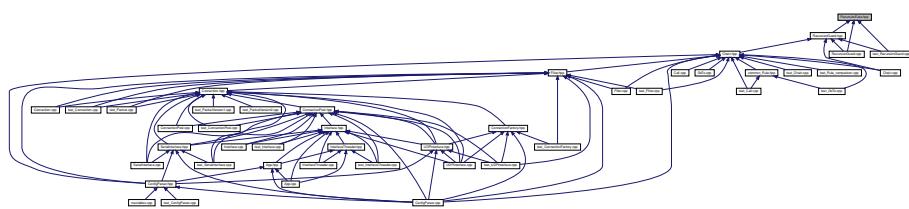
```
00096 the first one found in the following order:  
00097  
00098 1. The target of the 'MAVTABLES_CONFIG_PATH' environment variable.  
00099 2. '.mavtablesrc' in the current directory.  
00100 3. '.mavtablesrc' at '$HOME/.mavtablesrc'.  
00101 4. The main configuration file at '/etc/mavtables.conf'.  
00102  
00103 If the '--config' flag is given then mavtables will only look for the given  
00104 configuration file.  
00105  
00106  
00107 ## Contributing  
00108  
00109 Before contributing read the 'CONTRIBUTING.md' file which gives guidelines that  
00110 must be followed by all developers working on mavtables.
```

16.155 RecursionData.hpp File Reference

```
#include <mutex>  
#include <set>  
#include <thread>  
Include dependency graph for RecursionData.hpp:
```



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



Classes

- class [RecursionData](#)

16.156 RecursionData.hpp

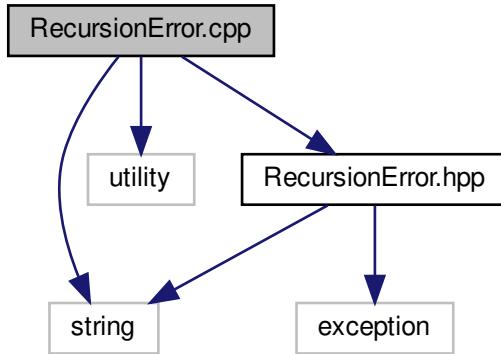
```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef RECURSIONDATA_HPP_
00019 #define RECURSIONDATA_HPP_
00020
00021
00022 #include <mutex>
00023 #include <set>
00024 #include <thread>
00025
00026
00027 /** A data structure used by \ref RecursionGuard to detect unwanted recursion.
00028 *
00029 * \note While \ref RecursionData supports copy and move semantics both with
00030 * constructors and assignment operator, a recursion data structure should
00031 * never change instance. One way to deal with this would have been to
00032 * delete these operators but this would force users to manually implement
00033 * copy and move semantics for their classes. Therefore, \ref
00034 * RecursionData will always make a new data structure on copy, move or
00035 * assignment, allowing default copy and move constructors/assignment
00036 * operators to be created for classes containing a \ref RecursionData
00037 * instance.
00038 */
00039 class RecursionData
00040 {
00041     friend class RecursionGuard;
00042
00043 public:
00044     RecursionData() = default;
00045     RecursionData(const RecursionData &other)
00046     {
00047         (void)other;
00048     }
00049     RecursionData(RecursionData &&other)
00050     {
00051         (void)other;
00052     }
00053     RecursionData &operator=(const RecursionData &other)
00054     {
00055         (void)other;
00056         calling_threads_.clear();
00057         return *this;
00058     }
00059     RecursionData &operator=(RecursionData &&other)
00060     {
00061         (void)other;
00062         calling_threads_.clear();
00063         return *this;
00064     }
00065
00066 private:
00067     std::set<std::thread::id> calling_threads_;
00068     std::mutex mutex_;
00069 };
00070
00071
00072 #endif // RECURSIONDATA_HPP_

```

16.157 RecursionError.cpp File Reference

```
#include <string>
#include <utility>
#include "RecursionError.hpp"
Include dependency graph for RecursionError.cpp:
```



16.158 RecursionError.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <string>
00019 #include <utility>
00020
00021 #include "RecursionError.hpp"
00022
00023
00024 /** Construct a RecursionError given a message.
00025 *
00026 * \param message The error message.
00027 */
00028 RecursionError::RecursionError(std::string message)
00029     : message_(std::move(message))
00030 {
00031 }
00032
00033

```

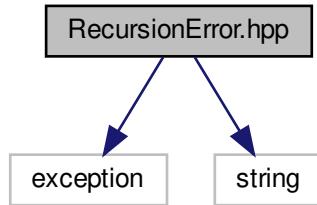
```

00034 /** Return error message string.
00035   *
00036   * \returns Error message string.
00037   */
00038 const char *RecursionError::what() const noexcept
00039 {
00040     return message_.c_str();
00041 }

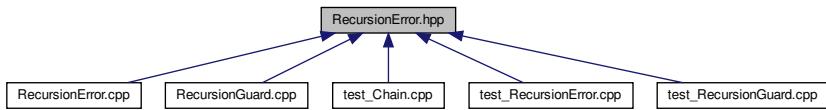
```

16.159 RecursionError.hpp File Reference

```
#include <exception>
#include <string>
Include dependency graph for RecursionError.hpp:
```



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



Classes

- class [RecursionError](#)

16.160 RecursionError.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //

```

```

00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef RECURSIONERROR_HPP_
00019 #define RECURSIONERROR_HPP_
00020
00021
00022 #include <exception>
00023 #include <string>
00024
00025
00026 /** Exception type emitted by a recursion guard.
00027 */
00028 class RecursionError : public std::exception
00029 {
00030     public:
00031         RecursionError(std::string message);
00032         const char *what() const noexcept;
00033
00034     private:
00035         std::string message_;
00036 };
00037
00038
00039 #endif // RECURSIONERROR_HPP_

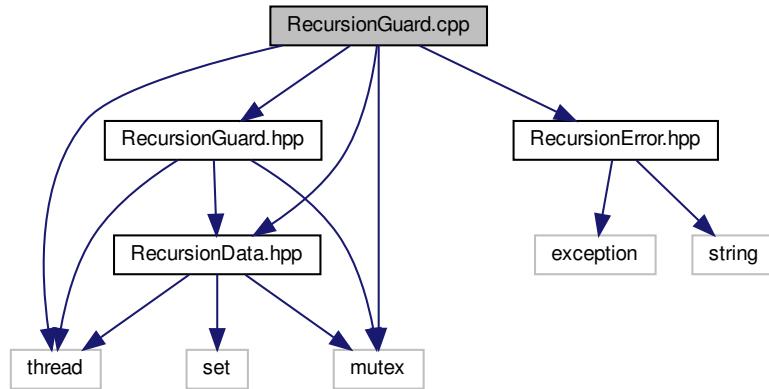
```

16.161 RecursionGuard.cpp File Reference

```

#include <mutex>
#include <thread>
#include "RecursionData.hpp"
#include "RecursionError.hpp"
#include "RecursionGuard.hpp"
Include dependency graph for RecursionGuard.cpp:

```



16.162 RecursionGuard.cpp

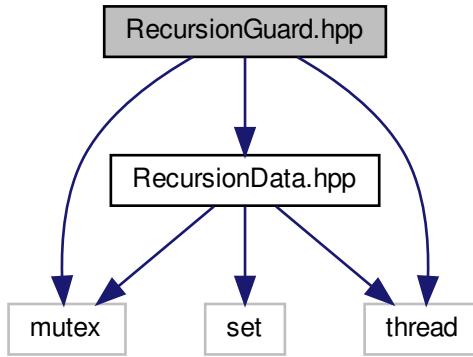
```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <mutex>
00019 #include <thread>
00020
00021 #include "RecursionData.hpp"
00022 #include "RecursionError.hpp"
00023 #include "RecursionGuard.hpp"
00024
00025
00026 /** Construct a RecursionGuard.
00027 *
00028 * This marks the given \ref RecursionData structure, ensuring it cannot be
00029 * used to construct another guard without raising a \ref RecursionError.
00030 *
00031 * \param data The object used to prevent recursion.
00032 * \throws RecursionError if the given \p data structure has already been
00033 * marked by a \ref RecursionGuard instance that is still in scope.
00034 */
00035 RecursionGuard::RecursionGuard(RecursionData &data)
00036 : data_(data)
00037 {
00038     std::lock_guard<std::mutex> lock(data_.mutex_);
00039     std::thread::id id = std::this_thread::get_id();
00040
00041     if (!data_.calling_threads_.insert(id).second)
00042     {
00043         throw RecursionError("Recursion detected.");
00044     }
00045 }
00046
00047
00048 /** Release the lock on the contain \ref RecursionData.
00049 *
00050 * This allows the function to be called again without error.
00051 */
00052 RecursionGuard::~RecursionGuard()
00053 {
00054     std::lock_guard<std::mutex> lock(data_.mutex_);
00055     data_.calling_threads_.erase(std::this_thread::get_id());
00056 }
```

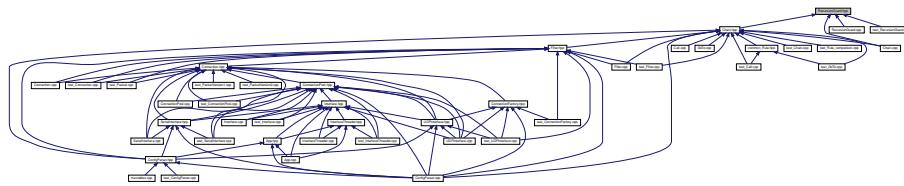
16.163 RecursionGuard.hpp File Reference

```
#include <mutex>
#include <thread>
#include "RecursionData.hpp"
```

Include dependency graph for RecursionGuard.hpp:



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



Classes

- class [RecursionGuard](#)

16.164 RecursionGuard.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
```

```

00018 #ifndef RECURSIONGUARD_HPP_
00019 #define RECURSIONGUARD_HPP_
00020
00021
00022 #include <mutex>
00023 #include <thread>
00024
00025 #include "RecursionData.hpp"
00026
00027
00028 /** Guard against recursion.
00029 *
00030 * A recursion guard is an RAII (Resource Acquisition Is Initialization) data
00031 * structure used to raise an error upon recursion. The constructor marks a
00032 * \ref RecursionData structure, acquiring ownership of the containing function
00033 * (within the given thread). Recursion guards treat calls from different
00034 * threads separately, therefore, it will not guard against re-entrancy.
00035 *
00036 * An example of how to use this is:
00037 *
00038 * ``
00039 * #include "RecursionGuard.hpp"
00040 *
00041 * int a_function(int value)
00042 * {
00043 *     // shared data between calls
00044 *     static RecursionData rdata;
00045 *     // take ownership of the call
00046 *     RecursionGuard rguard(rdata);
00047 *     return b_function(value);
00048 *     // the recursion guard is released upon destruction of rguard
00049 * }
00050 * ``
00051 *
00052 * %If 'b_function', or any function it calls, ever calls 'a_function' then
00053 * this will throw \ref RecursionError. However, if multiple threads call
00054 * 'a_function' (possibly at the same time) but 'b_function' does not call
00055 * 'a_function' then no error will be thrown.
00056 *
00057 * \sa RecursionData
00058 * \sa RecursionError
00059 */
00060 class RecursionGuard
00061 {
00062     public:
00063         RecursionGuard(RecursionData &data);
00064         ~RecursionGuard();
00065
00066     private:
00067         RecursionData &data_;
00068 };
00069
00070
00071 #endif // RECURSIONGUARD_HPP_

```

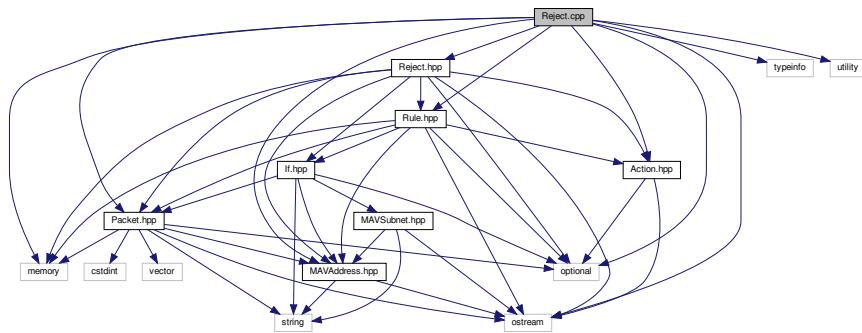
16.165 Reject.cpp File Reference

```

#include <memory>
#include <optional>
#include <ostream>
#include <typeinfo>
#include <utility>
#include "Action.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "Reject.hpp"
#include "Rule.hpp"

```

Include dependency graph for Reject.cpp:



16.166 Reject.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <memory>
00019 #include <optional>
00020 #include <iostream>
00021 #include <typeinfo>
00022 #include <utility>
00023
00024 #include "Action.hpp"
00025 #include "MAVAddress.hpp"
00026 #include "Packet.hpp"
00027 #include "Reject.hpp"
00028 #include "Rule.hpp"
00029
00030
00031 /** Construct a reject rule.
00032 *
00033 * A reject rule is used to reject packet/address combinations that match the
00034 * condition of the rule.
00035 *
00036 * \param condition The condition used to determine the rule matches a
00037 * particular packet/address combination given to the \ref action method.
00038 * The default is {} which indicates the rule matches any packet/address
00039 * combination.
00040 * \sa action
00041 */
00042 Reject::Reject(std::optional<If> condition)
00043     : Rule(std::move(condition))
00044 {
00045 }
00046
00047
00048 /** \copydoc Rule::print_(std::ostream &os) const
00049 *
00050 * Prints "reject" or "reject <If Statement>" if the rule's condition was
  
```

```

00051     * set.
00052     */
00053     std::ostream &Reject::print_(std::ostream &os) const
00054     {
00055         os << "reject";
00056
00057         if (condition_)
00058         {
00059             os << " " << condition_.value();
00060         }
00061
00062         return os;
00063     }
00064
00065
00066 /** \copydoc Rule::action(const Packet&,const MAVAddress&) const
00067 *
00068 * %If the condition has not been set or it matches the given packet/address
00069 * combination then it will return the reject \ref Action object, otherwise it
00070 * will return the continue \ref Action object.
00071 */
00072     Action Reject::action(
00073         const Packet &packet, const MAVAddress &address) const
00074     {
00075         if (!condition_ || condition_->check(packet, address))
00076         {
00077             return Action::make_reject();
00078         }
00079
00080         return Action::make_continue();
00081     }
00082
00083
00084     std::unique_ptr<Rule> Reject::clone() const
00085     {
00086         return std::make_unique<Reject>(condition_);
00087     }
00088
00089
00090     bool Reject::operator==(const Rule &other) const
00091     {
00092         return typeid(*this) == typeid(other) &&
00093             condition_ == static_cast<const Reject &>(other).condition_;
00094     }
00095
00096
00097     bool Reject::operator!=(const Rule &other) const
00098     {
00099         return typeid(*this) != typeid(other) ||
00100             condition_ != static_cast<const Reject &>(other).condition_;
00101     }

```

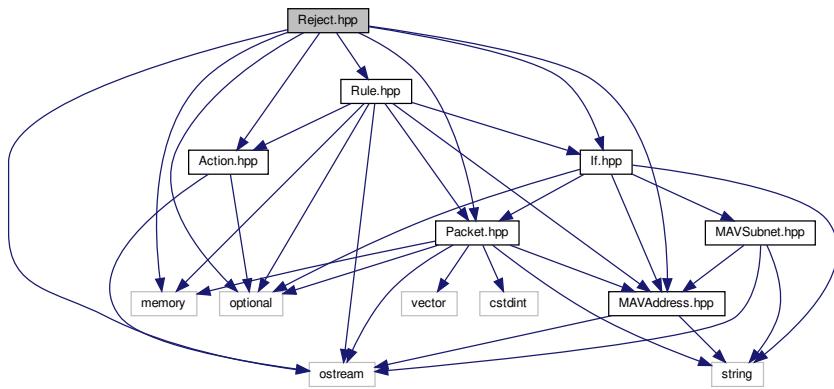
16.167 Reject.hpp File Reference

```

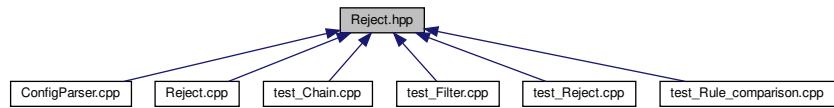
#include <memory>
#include <optional>
#include <iostream>
#include "Action.hpp"
#include "If.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "Rule.hpp"

```

Include dependency graph for Reject.hpp:



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



Classes

- class [Reject](#)

16.168 Reject.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef REJECT_HPP_
00019 #define REJECT_HPP_
00020
00021
  
```

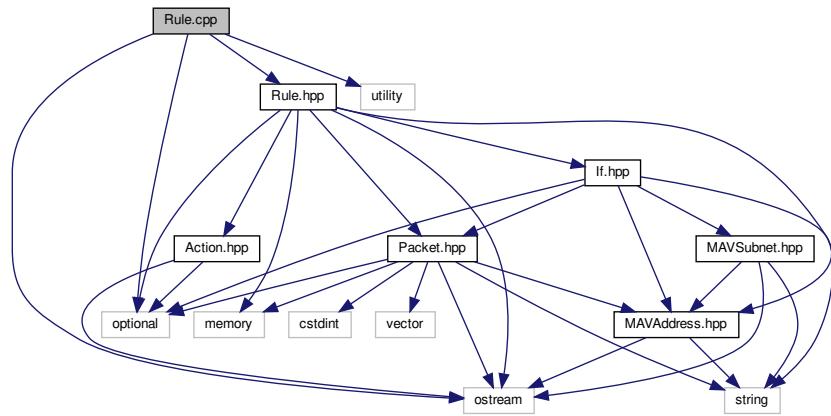
```

00022 #include <memory>
00023 #include <optional>
00024 #include <iostream>
00025
00026 #include "Action.hpp"
00027 #include "If.hpp"
00028 #include "MAVAddress.hpp"
00029 #include "Packet.hpp"
00030 #include "Rule.hpp"
00031
00032
00033 /** Rule to reject a packet.
00034 */
00035 class Reject : public Rule
00036 {
00037     public:
00038         Reject(std::optional<If> condition = {});
00039         virtual Action action(
00040             const Packet &packet, const MAVAddress &address) const;
00041         virtual std::unique_ptr<Rule> clone() const;
00042         virtual bool operator==(const Rule &other) const;
00043         virtual bool operator!=(const Rule &other) const;
00044
00045     protected:
00046         virtual std::ostream &print_(std::ostream &os) const;
00047 };
00048
00049
00050 #endif // REJECT_HPP_

```

16.169 Rule.cpp File Reference

```
#include <optional>
#include <iostream>
#include <utility>
#include "Rule.hpp"
Include dependency graph for Rule.cpp:
```



16.170 Rule.cpp

```
00001 // MAVLink router and firewall.
```

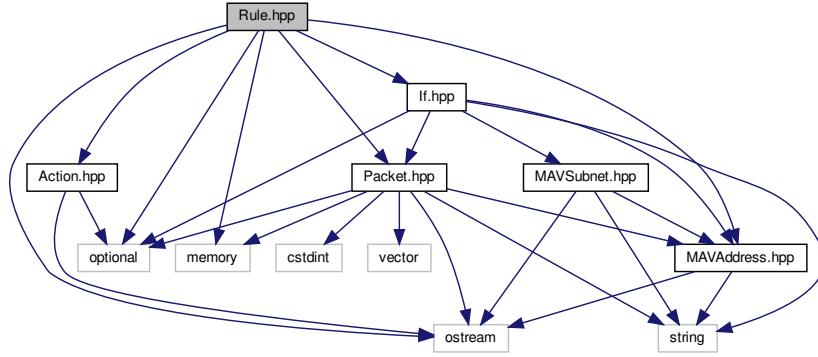
```

00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <optional>
00019 #include <ostream>
00020 #include <utility>
00021
00022 #include "Rule.hpp"
00023
00024
00025 /** Base constructor for Rule classes.
00026 *
00027 * \param condition The condition used to determine if the rule matches a
00028 * particular packet/address combination given to the \ref action method.
00029 * The default is {} which indicates the rule matches any packet/address
00030 * combination.
00031 * \sa action
00032 */
00033 Rule::Rule(std::optional<If> condition)
00034 : condition_(std::move(condition))
00035 {
00036 }
00037
00038
00039 // GCC generates a seemingly uncallable destructor for pure virtual classes.
00040 // Therefore, it must be excluded from test coverage.
00041 // LCOV_EXCL_START
00042 Rule::~Rule()
00043 {
00044 }
00045 // LCOV_EXCL_STOP
00046
00047
00048 /** Print the given rule to the given output stream.
00049 *
00050 * \note This is a polymorphic print, it will work on any child of \ref Rule
00051 * even if the pointer/reference is to the base class \ref Rule.
00052 *
00053 * Some examples are:
00054 * - 'accept'
00055 * - 'accept with priority 3'
00056 * - 'reject if ENCAPSULATED_DATA'
00057 * - 'call gcs_in if from 192.168'
00058 * - 'goto autopilot_out'
00059 *
00060 * \relates Rule
00061 * \param os The output stream to print to.
00062 * \param rule The rule (or any child of \ref Rule) to print.
00063 * \returns The output stream.
00064 */
00065 std::ostream &operator<<(std::ostream &os, const Rule &rule)
00066 {
00067     return rule.print_(os);
00068 }
```

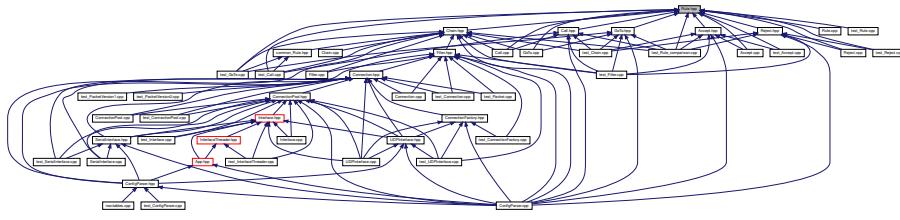
16.171 Rule.hpp File Reference

```
#include <memory>
#include <optional>
#include <ostream>
#include "Action.hpp"
```

```
#include "If.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
Include dependency graph for Rule.hpp:
```



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



Classes

- class [Rule](#)

Functions

- std::ostream & [operator<<](#) (std::ostream &os, const [Rule](#) &rule)

16.171.1 Function Documentation

16.171.1.1 [operator<<\(\)](#)

```
std::ostream& operator<< (
    std::ostream & os,
    const Rule & rule )
```

16.172 Rule.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef RULE_HPP_
00019 #define RULE_HPP_
00020
00021
00022 #include <memory>
00023 #include <optional>
00024 #include <iostream>
00025
00026 #include "Action.hpp"
00027 #include "If.hpp"
00028 #include "MAVAddress.hpp"
00029 #include "Packet.hpp"
00030
00031
00032 /** Base class of all rules, used in filter \ref Chain's.
00033 *
00034 * \ref Rule's are used to determine an \ref Action to take with a packet based
00035 * on its type, source address, and destination address. They are very much
00036 * like the rules found in a typical software defined firewalls.
00037 */
00038 class Rule
00039 {
00040     public:
00041         Rule(std::optional<If> condition = {});
00042         virtual ~Rule(); // Clang does not like pure virtual destructors.
00043         /** Decide what to do with a \ref Packet.
00044         *
00045         * Determine what action to take with the given \p packet sent to the
00046         * given \p address. The possible actions are documented in the \ref
00047         * Action class. The continue object is always returned if the
00048         * condition was set and does not match the packet/address combination.
00049         *
00050         * \param packet The packet to determine whether to allow or not.
00051         * \param address The address the \p packet will be sent out on if the
00052         *     action dictates it.
00053         * \returns The action to take with the packet. %If this is the accept
00054         *     object, it may also contain a priority for the packet.
00055         */
00056         virtual Action action(
00057             const Packet &packet, const MAVAddress &address) const = 0;
00058         /** Return a copy of the Rule polymorphically.
00059         *
00060         * This allows Rule's to be copied without knowing the derived type.
00061         *
00062         * \returns A pointer to a new object with base type \ref Rule which
00063         *     is an exact copy of this one.
00064         */
00065         virtual std::unique_ptr<Rule> clone() const = 0;
00066         /** Equality comparison.
00067         *
00068         * Compares the type of the \ref Rule and the condition (\ref If) if
00069         * set.
00070         *
00071         * \param other The other rule to compare this to.
00072         * \retval true if this rule is the same as \p other.
00073         * \retval false if this rule is not the same as \p other.
00074         */
00075         virtual bool operator==(const Rule &other) const = 0;
00076         /** Inequality comparison.
00077         *
```

```

00078     * Compares the type of the \ref Rule and the condition (\ref If) if
00079     * set.
00080     *
00081     * \param other The other rule to compare this to.
00082     * \retval true if this rule is not the same as \p other.
00083     * \retval false if this rule is the same as \p other.
00084     */
00085     virtual bool operator!=(const Rule &other) const = 0;
00086
00087     friend std::ostream &operator<<(std::ostream &os, const Rule &
00088     action);
00089
00090     protected:
00091     // Variables
00092     std::optional<If> condition_;
00093     // Methods
00094     /** Print the rule to the given output stream.
00095     *
00096     * \param os The output stream to print to.
00097     * \returns The output stream.
00098     */
00099     virtual std::ostream &print_(std::ostream &os) const = 0;
00100
00101
00102     std::ostream &operator<<(std::ostream &os, const Rule &rule);
00103
00104
00105 #endif // RULE_HPP_

```

16.173 run_tests.sh File Reference

16.174 unit_tests/run_tests.sh

```

00001#!/bin/bash
00002
00003
00004 function dir() {
00005     DIR=$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )
00006     echo "$DIR"
00007 }
00008
00009
00010 source "$(dir)/../ansi_codes.sh"
00011
00012
00013 function shutdown_background() {
00014     array=($(jobs -p))
00015     for ((i = ${#array[@]} - 1; i >= 0; i--)); do
00016         kill -SIGINT ${array[i]} >/dev/null
00017         sleep 0.5
00018     done
00019 }
00020
00021
00022 echo -en "${_BOLD}${_BLUE}*-----*"
00023 echo -en "-----*\n"
00024 echo -en "${_BOLD}${_BLUE}|"
00025 echo -en "Unit Tests"
00026 echo -en "-----*\n"
00027 echo -en "${_BOLD}${_BLUE}*-----*"
00028 echo -en "-----*\n"
00029 echo -en "${ANSI_RESET}"
00030
00031
00032 socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00033 sleep 5
00034
00035 ./build/unit_tests
00036 UNIT_TESTS_EXIT=$?
00037 sleep 1
00038

```

```

00039 shutdown_background
00040
00041 if [ "$UNIT_TESTS_EXIT" -ne "0" ]; then
00042     exit 1;
00043 fi

```

16.175 run_tests.sh File Reference

16.176 integration_tests/run_tests.sh

```

00001 #!/bin/bash
00002
00003
00004 FAIL=0
00005
00006
00007 function dir() {
00008     DIR=$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )
00009     echo "$DIR"
00010 }
00011
00012
00013 source "$(dir)/../ansi_codes.sh"
00014
00015
00016 function run_test() {
00017     PAD=$(printf '%0.1s' ".{1..80}")
00018     printf "${_BOLD}%s${ANSI_RESET}" "$1  "
00019     $2
00020     DIFF=$(diff "$(dir)"/$4 "$(dir)"/$3)
00021     if [ "$DIFF" != "" ]; then
00022         printf "%.*.*s" 0 ${((67 - ${#1}))} "$PAD"
00023         printf "  ${_BOLD}${_RED}%s${ANSI_RESET}\n" "[FAILED]"
00024         echo "$DIFF"
00025         FAIL=1
00026     else
00027         printf "%.*.*s" 0 ${((66 - ${#1}))} "$PAD"
00028         printf "  ${_BOLD}${_GREEN}%s${ANSI_RESET}\n" "[SUCCESS]"
00029     fi
00030 }
00031
00032
00033 function shutdown_background() {
00034     array=($(jobs -p))
00035     for ((i = ${#array[@]} - 1; i >= 0; i--)); do
00036         kill -SIGINT ${array[i]} 2>/dev/null
00037         sleep 2
00038     done
00039 }
00040
00041
00042 function do_nothing() {
00043     echo "do nothing" >/dev/null
00044 }
00045
00046
00047 function test_ast_printing() {
00048     "$(dir)/../../build/mavtables" --ast --conf "$(dir)/../mavtables.conf" \
00049         | tail -n +2 > "$(dir)/mavtables.log"
00050 }
00051
00052
00053 function test_complex_ast_printing() {
00054     "$(dir)/../../build/mavtables" --ast \
00055         --conf "$(dir)/complex_config.conf" \
00056         | tail -n +2 > "$(dir)/complex_config.log"
00057 }
00058
00059
00060 function test_all_v1_packets_udp() {
00061     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00062     sleep 2

```

```

00063     "$(dir)/../../../../build/mavtables" --conf "$(dir)/all_1serial.conf" &
00064     "$(dir)/logger.py" 12 26 --udp 127.0.0.1:14500 \
00065         > "$(dir)/all_v1_packets_udp_to_udp.log" &
00066     "$(dir)/logger.py" 29 39 --serial ./ttyS1 \
00067         > "$(dir)/all_v1_packets_udp_to_serial.log" &
00068     sleep 2
00069     "$(dir)/packet_scripter.py" 192 168 "$(dir)/all_v1_packets.pks" \
00070         --udp 127.0.0.1:14500 --mavlink1
00071     sleep 2
00072     shutdown_background
00073 }
00074
00075
00076 function test_all_v2_packets_udp() {
00077     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00078     sleep 2
00079     "$(dir)/../../../../build/mavtables" --conf "$(dir)/all_1serial.conf" &
00080     "$(dir)/logger.py" 12 26 --udp 127.0.0.1:14500 \
00081         > "$(dir)/all_v2_packets_udp_to_udp.log" &
00082     "$(dir)/logger.py" 29 39 --serial ./ttyS1 \
00083         > "$(dir)/all_v2_packets_udp_to_serial.log" &
00084     sleep 2
00085     "$(dir)/packet_scripter.py" 192 168 "$(dir)/all_v2_packets.pks" \
00086         --udp 127.0.0.1:14500
00087     sleep 2
00088     shutdown_background
00089 }
00090
00091
00092 function test_all_v1_packets_serial() {
00093     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00094     socat pty,link=./ttyS2,raw,echo=0 pty,link=./ttyS3,raw,echo=0 &
00095     sleep 2
00096     "$(dir)/../../../../build/mavtables" --conf "$(dir)/all_2serial.conf" &
00097     "$(dir)/logger.py" 12 26 --udp 127.0.0.1:14500 \
00098         > "$(dir)/all_v1_packets_serial_to_udp.log" &
00099     "$(dir)/logger.py" 29 39 --serial ./ttyS1 \
00100         > "$(dir)/all_v1_packets_serial_to_serial.log" &
00101     sleep 2
00102     "$(dir)/packet_scripter.py" 192 168 "$(dir)/all_v1_packets.pks" \
00103         --serial ./ttyS3 --mavlink1
00104     sleep 2
00105     shutdown_background
00106 }
00107
00108
00109 function test_all_v2_packets_serial() {
00110     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00111     socat pty,link=./ttyS2,raw,echo=0 pty,link=./ttyS3,raw,echo=0 &
00112     sleep 2
00113     "$(dir)/../../../../build/mavtables" --conf "$(dir)/all_2serial.conf" &
00114     "$(dir)/logger.py" 12 26 --udp 127.0.0.1:14500 \
00115         > "$(dir)/all_v2_packets_serial_to_udp.log" &
00116     "$(dir)/logger.py" 29 39 --serial ./ttyS1 \
00117         > "$(dir)/all_v2_packets_serial_to_serial.log" &
00118     sleep 2
00119     "$(dir)/packet_scripter.py" 192 168 "$(dir)/all_v2_packets.pks" \
00120         --serial ./ttyS3
00121     sleep 2
00122     shutdown_background
00123 }
00124
00125
00126 function test_multiple_senders_v1_packets() {
00127     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00128     socat pty,link=./ttyS2,raw,echo=0 pty,link=./ttyS3,raw,echo=0 &
00129     sleep 2
00130     "$(dir)/../../../../build/mavtables" --conf "$(dir)/all_2serial.conf" &
00131     "$(dir)/logger.py" 12 26 --udp 127.0.0.1:14500 --verbose \
00132         | sort --version-sort \
00133             > "$(dir)/multiple_senders_v1_packets_to_udp.log" &
00134     "$(dir)/logger.py" 10 20 --serial ./ttyS1 --verbose \
00135         | sort --version-sort \
00136             > "$(dir)/multiple_senders_v1_packets_to_serial.log" &
00137     sleep 2
00138     "$(dir)/packet_scripter.py" 192 168 "$(dir)/all_v1_packets.pks" \
00139         --udp 127.0.0.1:14500 --mavlink1 &
00140     "$(dir)/packet_scripter.py" 172 128 "$(dir)/all_v1_packets.pks" \
00141         --udp 127.0.0.1:14500 --mavlink1 &
00142     "$(dir)/packet_scripter.py" 127 1 "$(dir)/all_v1_packets.pks" \
00143         --serial ./ttyS3 --mavlink1

```

```

00144     sleep 2
00145     shutdown_background
00146 }
00147
00148
00149 function test_multiple_senders_v2_packets() {
00150     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00151     socat pty,link=./ttyS2,raw,echo=0 pty,link=./ttyS3,raw,echo=0 &
00152     sleep 2
00153     "$(dir)/../../build/mavtables" --conf "$(dir)/all_2serial.conf" &
00154     "$(dir)/logger.py" 12 26 --udp 127.0.0.1:14500 --verbose \
00155         | sort --version-sort \
00156         > "$(dir)/multiple_senders_v2_packets_to_udp.log" &
00157     "$(dir)/logger.py" 10 20 --serial ./ttyS1 --verbose \
00158         | sort --version-sort \
00159         > "$(dir)/multiple_senders_v2_packets_to_serial.log" &
00160     sleep 2
00161     "$(dir)/packet_scripter.py" 192 168 "$(dir)/all_v2_packets.pks" \
00162         --udp 127.0.0.1:14500 &
00163     "$(dir)/packet_scripter.py" 172 128 "$(dir)/all_v2_packets.pks" \
00164         --udp 127.0.0.1:14500 &
00165     "$(dir)/packet_scripter.py" 127 1 "$(dir)/all_v2_packets.pks" \
00166         --serial ./ttyS3
00167     sleep 2
00168     shutdown_background
00169 }
00170
00171
00172 function test_routing_v1_packets() {
00173     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00174     sleep 2
00175     "$(dir)/../../build/mavtables" --conf "$(dir)/routing.conf" &
00176     "$(dir)/logger.py" 127 1 --udp 127.0.0.1:14500 \
00177         > "$(dir)/routing_v1_packets_127.1.log" &
00178     "$(dir)/logger.py" 192 168 --udp 127.0.0.1:14500 \
00179         > "$(dir)/routing_v1_packets_192.168.log" &
00180     "$(dir)/logger.py" 172 128 --serial ./ttyS1 \
00181         > "$(dir)/routing_v1_packets_172.128.log" &
00182     sleep 2
00183     "$(dir)/packet_scripter.py" 10 10 "$(dir)/routing.pks" \
00184         --udp 127.0.0.1:14500 --mavlink1
00185     sleep 2
00186     shutdown_background
00187 }
00188
00189
00190 function test_routing_v2_packets() {
00191     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00192     sleep 2
00193     "$(dir)/../../build/mavtables" --conf "$(dir)/routing.conf" &
00194     "$(dir)/logger.py" 127 1 --udp 127.0.0.1:14500 \
00195         > "$(dir)/routing_v2_packets_127.1.log" &
00196     "$(dir)/logger.py" 192 168 --udp 127.0.0.1:14500 \
00197         > "$(dir)/routing_v2_packets_192.168.log" &
00198     "$(dir)/logger.py" 172 128 --serial ./ttyS1 \
00199         > "$(dir)/routing_v2_packets_172.128.log" &
00200     sleep 2
00201     "$(dir)/packet_scripter.py" 10 10 "$(dir)/routing.pks" \
00202         --udp 127.0.0.1:14500
00203     sleep 2
00204     shutdown_background
00205 }
00206
00207
00208 function test_priority() {
00209     perl -e 'for$i(1..50){print "ENCAPSULATED_DATA\n"}' \
00210         > "$(dir)/priority.tmp"
00211     perl -e 'for$i(1..100){print "ATTITUDE\n"}' \
00212         >> "$(dir)/priority.tmp"
00213     perl -e 'for$i(1..100){print "GLOBAL_POSITION_INT\n"}' \
00214         >> "$(dir)/priority.tmp"
00215     perl -e 'for$i(1..50){print "ENCAPSULATED_DATA\n"}' \
00216         >> "$(dir)/priority.tmp"
00217     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00218     sleep 2
00219     "$(dir)/../../build/mavtables" --conf "$(dir)/priority.conf" &
00220     "$(dir)/logger.py" 192 168 --udp 127.0.0.1:14500 > "$(dir)/priority.log" &
00221     sleep 2
00222     "$(dir)/packet_scripter.py" 10 10 "$(dir)/priority.tmp" --serial ./ttyS1
00223     sleep 2
00224     perl -e 'for$i(1..100){print "ENCAPSULATED_DATA\n"}' \

```

```

00225      > "$(dir)/priority.tmp"
00226 perl -e 'for$i(1..100){print "GLOBAL_POSITION_INT\n"}' \
00227      >> "$(dir)/priority.tmp"
00228 perl -e 'for$i(1..100){print "ATTITUDE\n"}' \
00229      >> "$(dir)/priority.tmp"
00230 shutdown_background
00231 }
00232
00233
00234 function test_preload_addresses() {
00235     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00236     sleep 2
00237     "$(dir)../../build/mavtables" --conf "$(dir)/preload.conf" &
00238     "$(dir)/logger.py" 10 10 --noheartbeat --serial ./ttyS1 \
00239      > "$(dir)/preload.log" &
00240     sleep 2
00241     "$(dir)/packet_scripter.py" 123 123 "$(dir)/routing.pks" \
00242      --udp 127.0.0.1:14500
00243     sleep 2
00244     shutdown_background
00245 }
00246
00247
00248 function test_component_broadcast_fallback() {
00249     perl -e 'for$i(0..255){print "MISSION_REQUEST_LIST to 1.$i\n"}' \
00250      > "$(dir)/component_broadcast_fallback.tmp"
00251     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00252     sleep 2
00253     "$(dir)../../build/mavtables" \
00254      --conf "$(dir)/component_broadcast_fallback.conf" &
00255     "$(dir)/logger.py" 1 1 --verbose --serial ./ttyS1 \
00256      > "$(dir)/component_broadcast_fallback.log" &
00257     sleep 2
00258     "$(dir)/packet_scripter.py" 10 10 \
00259      "> $(dir)/component_broadcast_fallback.tmp" \
00260      --udp 127.0.0.1:14500
00261     perl -e \
00262      'for$i(128..255){print "MISSION_REQUEST_LIST from 10.10 to 1.$i\n"}' \
00263      > "$(dir)/component_broadcast_fallback.tmp"
00264     sleep 2
00265     shutdown_background
00266 }
00267
00268
00269 function test_large_packets() {
00270     perl -e 'for$i(1..5000){print "ENCAPSULATED_DATA\n"}' \
00271      > "$(dir)/large_packets.tmp"
00272     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00273     socat pty,link=./ttyS2,raw,echo=0 pty,link=./ttyS3,raw,echo=0 &
00274     sleep 2
00275     "$(dir)../../build/mavtables" --conf "$(dir)/all_2serial.conf" &
00276     "$(dir)/logger.py" 12 26 --udp 127.0.0.1:14500 \
00277      > "$(dir)/large_packets_to_udp.log" &
00278     "$(dir)/logger.py" 10 20 --serial ./ttyS1 \
00279      > "$(dir)/large_packets_to_serial.log" &
00280     sleep 2
00281     "$(dir)/packet_scripter.py" 192 168 "$(dir)/large_packets.tmp" \
00282      --udp 127.0.0.1:14500 &
00283     "$(dir)/packet_scripter.py" 172 128 "$(dir)/large_packets.tmp" \
00284      --serial ./ttyS3
00285     sleep 60
00286     perl -e 'for$i(1..5000){print "ENCAPSULATED_DATA\n"}' \
00287      >> "$(dir)/large_packets.tmp"
00288     shutdown_background
00289 }
00290
00291
00292 function test_logging_level1() {
00293     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00294     sleep 1
00295     "$(dir)../../build/mavtables" --loglevel 1 --conf "$(dir)/routing.conf" \
00296      > "$(dir)/logging_level1.log" &
00297     sleep 1
00298     "$(dir)/logger.py" 127 1 --udp 127.0.0.1:14500 > /dev/null &
00299     sleep 1
00300     "$(dir)/logger.py" 192 168 --udp 127.0.0.1:14500 > /dev/null &
00301     sleep 1
00302     "$(dir)/logger.py" 172 128 --serial ./ttyS1 > /dev/null &
00303     sleep 1
00304     "$(dir)/packet_scripter.py" 10 10 "$(dir)/routing.pks" \
00305      --udp 127.0.0.1:14500

```

```
00306     sleep 1
00307     sed -i 's/^.....//' "$(dir)/logging_level1.log"
00308     sed -i 's/:[0-9]*//g' "$(dir)/logging_level1.log"
00309     shutdown_background
00310 }
00311
00312
00313 function test_logging_level2() {
00314     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00315     sleep 1
00316     "$(dir)/../../build/mavtables" --loglevel 2 --conf "$(dir)/routing.conf" \
00317         > "$(dir)/logging_level2.log" &
00318     sleep 1
00319     "$(dir)/logger.py" 127 1 --udp 127.0.0.1:14500 > /dev/null &
00320     sleep 1
00321     "$(dir)/logger.py" 192 168 --udp 127.0.0.1:14500 > /dev/null &
00322     sleep 1
00323     "$(dir)/logger.py" 172 128 --serial ./ttyS1 > /dev/null &
00324     sleep 1
00325     "$(dir)/packet_scripter.py" 10 10 "$(dir)/routing.pks" \
00326         --udp 127.0.0.1:14500
00327     sleep 1
00328     sed -i 's/^.....//' "$(dir)/logging_level2.log"
00329     sed -i 's/:[0-9]*//g' "$(dir)/logging_level2.log"
00330     shutdown_background
00331 }
00332
00333
00334 function test_logging_level3() {
00335     socat pty,link=./ttyS0,raw,echo=0 pty,link=./ttyS1,raw,echo=0 &
00336     sleep 1
00337     "$(dir)/../../build/mavtables" --loglevel 3 --conf "$(dir)/routing.conf" \
00338         > "$(dir)/logging_level3.log" &
00339     sleep 1
00340     "$(dir)/logger.py" 127 1 --udp 127.0.0.1:14500 > /dev/null &
00341     sleep 1
00342     "$(dir)/logger.py" 192 168 --udp 127.0.0.1:14500 > /dev/null &
00343     sleep 1
00344     "$(dir)/logger.py" 172 128 --serial ./ttyS1 > /dev/null &
00345     sleep 1
00346     "$(dir)/packet_scripter.py" 10 10 "$(dir)/routing.pks" \
00347         --udp 127.0.0.1:14500
00348     sleep 1
00349     sed -i 's/^.....//' "$(dir)/logging_level3.log"
00350     sed -i 's/:[0-9]*//g' "$(dir)/logging_level3.log"
00351     shutdown_background
00352 }
00353
00354
00355 echo -en "${_BOLD}${_BLUE}*-----"
00356 echo -en "-----*\n"
00357 echo -en "${_BOLD}${_BLUE}|"
00358 echo -en "Integration Tests"
00359 echo -en "|\n"
00360 echo -en "${_BOLD}${_BLUE}*-----"
00361 echo -en "-----*\n"
00362 echo -en "${ANSI_RESET}"
00363
00364
00365 rm "$(dir)/*.*.log 2>/dev/null
00366
00367
00368 case "$OSTYPE" in
00369     solaris*)
00370     ;;
00371     darwin*)
00372         echo -en "${_BOLD}${_ITALICS}${_RED}Integration testing is not currently "
00373         echo -en "supported on Mac OS X.${ANSI_RESET}\n\n"
00374         exit 0
00375     ;;
00376     linux*)
00377     ;;
00378     bsd*)
00379     ;;
00380     *)
00381         echo "unknown operating system: $OSTYPE"
00382         exit 1
00383     ;;
00384 esac
00385
00386
```

```

00387 function check_fail() {
00388     if [ "$?" -ne "0" ]; then
00389         FAIL=1
00390     fi
00391 }
00392
00393
00394 run_test "Abstract Syntax Tree printing with --ast flag" \
00395     test_ast_printing \
00396     mavtables.cmp \
00397     mavtables.log
00398 run_test "Complex Abstract Syntax Tree printing with --ast flag" \
00399     test_complex_ast_printing \
00400     complex_config.cmp \
00401     complex_config.log
00402
00403
00404 run_test "All MAVLink v1.0 packets from UDP to UDP" \
00405     test_all_v1_packets_udp \
00406     all_v1_packets.pks \
00407     all_v1_packets_udp_to_udp.log
00408 run_test "All MAVLink v1.0 packets from UDP to serial port" \
00409     do_nothing \
00410     all_v1_packets.pks \
00411     all_v1_packets_udp_to_serial.log
00412 run_test "All MAVLink v2.0 packets from UDP to UDP" \
00413     test_all_v2_packets_udp \
00414     all_v2_packets.pks \
00415     all_v2_packets_udp_to_udp.log
00416 run_test "All MAVLink v2.0 packets from UDP to serial port" \
00417     do_nothing \
00418     all_v2_packets.pks \
00419     all_v2_packets_udp_to_serial.log
00420
00421
00422 run_test "All MAVLink v1.0 packets from serial port to UDP" \
00423     test_all_v1_packets_serial \
00424     all_v1_packets.pks \
00425     all_v1_packets_serial_to_udp.log
00426 run_test "All MAVLink v1.0 packets from serial port to serial port" \
00427     do_nothing \
00428     all_v1_packets.pks \
00429     all_v1_packets_serial_to_serial.log
00430 run_test "All MAVLink v2.0 packets from serial port to UDP" \
00431     test_all_v2_packets_serial \
00432     all_v2_packets.pks \
00433     all_v2_packets_serial_to_udp.log
00434 run_test "All MAVLink v2.0 packets from serial port to serial port" \
00435     do_nothing \
00436     all_v2_packets.pks \
00437     all_v2_packets_serial_to_serial.log
00438
00439
00440 run_test "Multiple senders with MAVLink v1.0 packets to UDP" \
00441     test_multiple_senders_v1_packets \
00442     multiple_senders_v1_packets.cmp \
00443     multiple_senders_v1_packets_to_udp.log
00444 run_test "Multiple senders with MAVLink v1.0 packets to serial port" \
00445     do_nothing \
00446     multiple_senders_v1_packets.cmp \
00447     multiple_senders_v1_packets_to_serial.log
00448 run_test "Multiple senders with MAVLink v2.0 packets to UDP" \
00449     test_multiple_senders_v2_packets \
00450     multiple_senders_v2_packets.cmp \
00451     multiple_senders_v2_packets_to_udp.log
00452 run_test "Multiple senders with MAVLink v2.0 packets to serial port" \
00453     do_nothing \
00454     multiple_senders_v2_packets.cmp \
00455     multiple_senders_v2_packets_to_serial.log
00456
00457
00458 run_test "Routing MAVLink v1.0 packets (part 1 - 127.1)" \
00459     test_routing_v1_packets_routing_127.1.cmp routing_v1_packets_127.1.log
00460 run_test "Routing MAVLink v1.0 packets (part 2 - 192.168)" \
00461     do_nothing routing_192.168.cmp routing_v1_packets_192.168.log
00462 run_test "Routing MAVLink v1.0 packets (part 3 - 172.128)" \
00463     do_nothing routing_172.128.cmp routing_v1_packets_172.128.log
00464
00465
00466 run_test "Routing MAVLink v2.0 packets (part 1 - 127.1)" \
00467     test_routing_v2_packets_routing_127.1.cmp routing_v2_packets_127.1.log

```

```

00468 run_test "Routing MAVLink v2.0 packets (part 2 - 192.168)" \
00469     do_nothing routing_192.168.cmp routing_v2_packets_192.168.log
00470 run_test "Routing MAVLink v2.0 packets (part 3 - 172.128)" \
00471     do_nothing routing_172.128.cmp routing_v2_packets_172.128.log
00472
00473
00474 run_test "High priority packets are transmitted first" \
00475     test_priority priority.tmp priority.log
00476
00477
00478 run_test "Serial ports can be preloaded with MAVLink addresses" \
00479     test_preload_addresses preload.cmp preload.log
00480
00481
00482 run_test "Component broadcast fallback and MAVLink subnets" \
00483     test_component_broadcastFallback \
00484     component_broadcastFallback.tmp \
00485     component_broadcastFallback.log
00486
00487
00488 run_test "Many large packets with multiple senders to UDP" \
00489     test_large_packets large_packets.tmp large_packets_to_udp.log
00490 run_test "Many large packets with multiple senders to serial port" \
00491     do_nothing large_packets.tmp large_packets_to_serial.log
00492
00493
00494 run_test "Logging level 1" \
00495     test_logging_level1 logging_level1.cmp logging_level1.log
00496 run_test "Logging level 2" \
00497     test_logging_level2 logging_level2.cmp logging_level2.log
00498 run_test "Logging level 3" \
00499     test_logging_level3 logging_level3.cmp logging_level3.log
00500
00501
00502 if [ "$FAIL" -ne "0" ]; then
00503     echo -en "\n"
00504     exit $FAIL
00505 fi

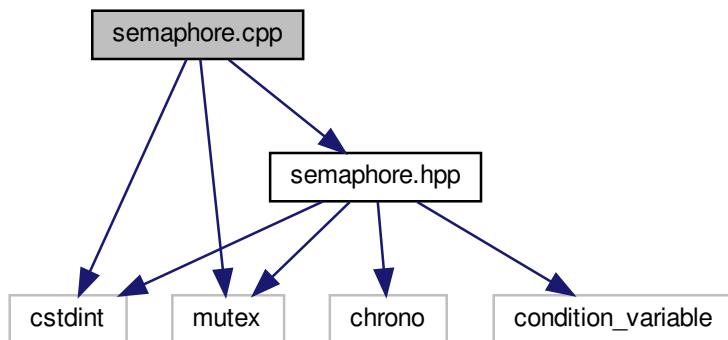
```

16.177 semaphore.cpp File Reference

```

#include <cstdint>
#include <mutex>
#include "semaphore.hpp"
Include dependency graph for semaphore.cpp:

```



16.178 semaphore.cpp

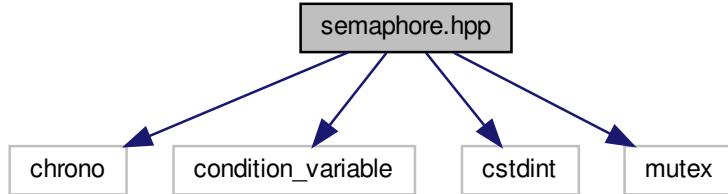
```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
0010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
0011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
0012 // GNU General Public License for more details.
0013 //
0014 // You should have received a copy of the GNU General Public License
0015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
0016
0017
0018 #include <cstdint>
0019 #include <mutex>
0020
0021 #include "semaphore.hpp"
0022
0023
0024 /** Construct a semaphore with the given initial value.
0025 *
0026 * \param initial_value The initial value of the semaphore. Defaults to 0.
0027 */
0028 semaphore::semaphore(size_t initial_value)
0029     : value_(initial_value)
0030 {
0031 }
0032
0033
0034 /** Signal the semaphore.
0035 *
0036 * Increments the semaphore.
0037 */
0038 void semaphore::notify()
0039 {
0040     {
0041         std::lock_guard<std::mutex> lock(mutex_);
0042         ++value_;
0043     }
0044     cv_.notify_one();
0045 }
0046
0047
0048 /** Wait on the semaphore.
0049 *
0050 * Decrement the semaphore.
0051 */
0052 void semaphore::wait()
0053 {
0054     std::unique_lock<std::mutex> lock(mutex_);
0055     cv_.wait(lock, [this]()
0056     {
0057         return value_ > 0;
0058     });
0059     --value_;
0060 }
```

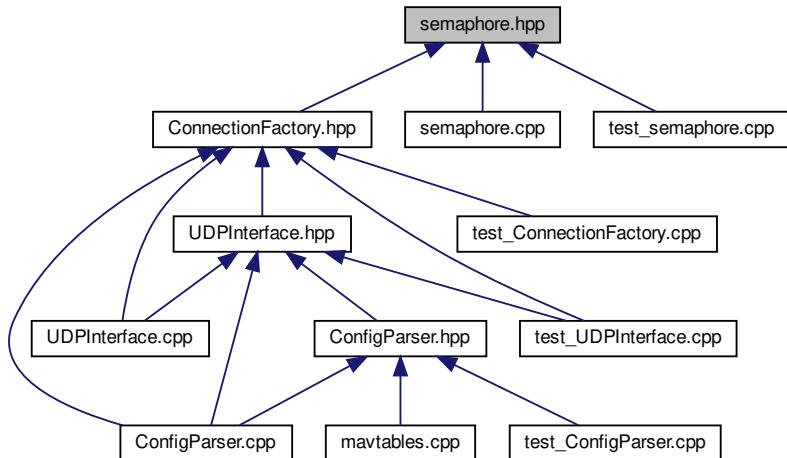
16.179 semaphore.hpp File Reference

```
#include <chrono>
#include <condition_variable>
#include <cstdint>
```

```
#include <mutex>
Include dependency graph for semaphore.hpp:
```



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



Classes

- class [semaphore](#)

16.180 semaphore.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
  
```

```
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
0010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
0011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
0012 // GNU General Public License for more details.
0013 //
0014 // You should have received a copy of the GNU General Public License
0015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
0016
0017
0018 #ifndef SEMAPHORE_HPP_
0019 #define SEMAPHORE_HPP_
0020
0021
0022 #include <chrono>
0023 #include <condition_variable>
0024 #include <cstdint>
0025 #include <mutex>
0026
0027
0028 /** A weak semaphore implementation.
0029 *
0030 * \note This uses a different naming scheme than the rest of this project
0031 *       because it is intended to have the same feel as the standard library.
0032 *
0033 * \note This semaphore implementation is based on
0034 *       https://gist.github.com/sguzman/9594227
0035 */
0036 class semaphore
0037 {
0038     public:
0039         semaphore(const semaphore &other) = delete;
0040         semaphore(semaphore &&other) = delete;
0041         semaphore(size_t initial_value = 0);
0042         void notify();
0043         void wait();
0044         template<class Rep, class Period>
0045         bool wait_for(const std::chrono::duration<Rep, Period> &rel_time);
0046         template<class Clock, class Duration>
0047         bool wait_until(
0048             const std::chrono::time_point<Clock, Duration> &timeout_time);
0049         semaphore &operator=(const semaphore &other) = delete;
0050         semaphore &operator=(semaphore &&other) = delete;
0051
0052     private:
0053         size_t value_;
0054         std::mutex mutex_;
0055         std::condition_variable cv_;
0056 };
0057
0058
0059 /** Wait on the semaphore, or a given duration timeout.
0060 *
0061 * \param rel_time The duration of the timeout.
0062 * \retval true The semaphore has been successfully decremented.
0063 * \retval false The semaphore timed out.
0064 */
0065 template<class Rep, class Period>
0066 bool semaphore::wait_for(const std::chrono::duration<Rep, Period> &rel_time)
0067 {
0068     std::unique_lock<std::mutex> lock(mutex_);
0069     bool result = cv_.wait_for(lock, rel_time, [this]()
0070     {
0071         return value_ > 0;
0072     });
0073
0074     if (result)
0075     {
0076         --value_;
0077     }
0078
0079     return result;
0080 }
0081
0082
0083 /** Wait on the semaphore, or a given timepoint timeout.
0084 *
0085 * \param timeout_time The timepoint for the timeout.
```

```

00086 * \retval true The semaphore has been successfully decremented.
00087 * \retval false The semaphore timed out.
00088 */
00089 template<class Clock, class Duration>
00090 bool semaphore::wait_until(
00091     const std::chrono::time_point<Clock, Duration> &timeout_time)
00092 {
00093     std::unique_lock<std::mutex> lock(mutex_);
00094     bool result = cv_.wait_until(lock, timeout_time, [this]()
00095     {
00096         return value_ > 0;
00097     });
00098     if (result)
00099     {
00100         --value_;
00101     }
00102     return result;
00103 }
00104
00105 #endif // SEMAPHORE_HPP_
00106
00107
00108 #endif // SERIALINTERFACE_HPP_

```

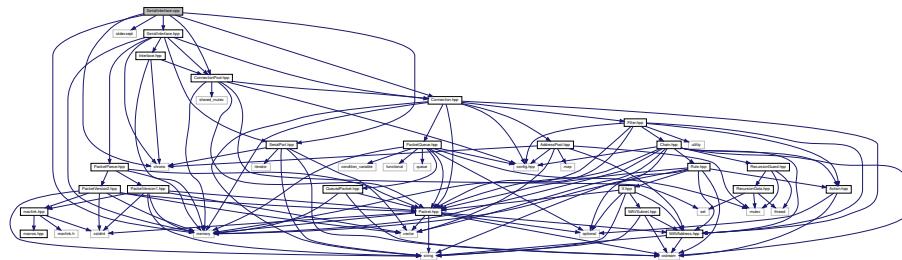
16.181 SerialInterface.cpp File Reference

```

#include <chrono>
#include <memory>
#include <stdexcept>
#include "Connection.hpp"
#include "ConnectionPool.hpp"
#include "SerialInterface.hpp"
#include "SerialPort.hpp"

```

Include dependency graph for SerialInterface.cpp:



16.182 SerialInterface.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.

```

```

00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <chrono>
00019 #include <memory>
00020 #include <stdexcept>
00021
00022 #include "Connection.hpp"
00023 #include "ConnectionPool.hpp"
00024 #include "SerialInterface.hpp"
00025 #include "SerialPort.hpp"
00026
00027
00028 /** Construct a serial port interface using a given device.
00029 *
00030 * \param port The serial port device to communicate over.
00031 * \param connection_pool The connection pool to use for sending packets the
00032 *     interface has received and to register the \p connection with.
00033 * \param connection The connection to get packets to send packets from. This
00034 *     will be registered with the given \ref ConnectionPool.
00035 * \throws std::invalid_argument if the serial \p port device pointer is null.
00036 * \throws std::invalid_argument if the \p connection_pool pointer is null.
00037 * \throws std::invalid_argument if the \p connection pointer is null.
00038 */
00039 SerialInterface::SerialInterface(
00040     std::unique_ptr<SerialPort> port,
00041     std::shared_ptr<ConnectionPool> connection_pool,
00042     std::unique_ptr<Connection> connection)
00043 : port_(std::move(port)),
00044   connection_pool_(std::move(connection_pool)),
00045   connection_(std::move(connection))
00046 {
00047     if (port_ == nullptr)
00048     {
00049         throw std::invalid_argument("Given serial port pointer is null.");
00050     }
00051
00052     if (connection_pool_ == nullptr)
00053     {
00054         throw std::invalid_argument("Given connection pool pointer is null.");
00055     }
00056
00057     if (connection_ == nullptr)
00058     {
00059         throw std::invalid_argument("Given connection pointer is null.");
00060     }
00061
00062     connection_pool_->add(connection_);
00063 }
00064
00065
00066 /** \copydoc Interface::send_packet(const std::chrono::nanoseconds &)
00067 *
00068 * Writes up to one packet from the contained connection to the serial port.
00069 */
00070 void SerialInterface::send_packet(const std::chrono::nanoseconds &timeout)
00071 {
00072     auto packet = connection_->next_packet(timeout);
00073
00074     if (packet != nullptr)
00075     {
00076         port_->write(packet->data());
00077     }
00078 }
00079
00080
00081 /** \copydoc Interface::receive_packet(const std::chrono::nanoseconds &)
00082 *
00083 * Reads the data in the serial port's receive buffer or waits for up to \p
00084 * timeout until data arrives if no data is present in the serial port buffer.
00085 */
00086 void SerialInterface::receive_packet(const std::chrono::nanoseconds &timeout
00087 )
00088 {
00089     auto buffer = port_->read(timeout);
00090
00091     if (!buffer.empty())
00092     {
00093         // Parse the bytes.

```

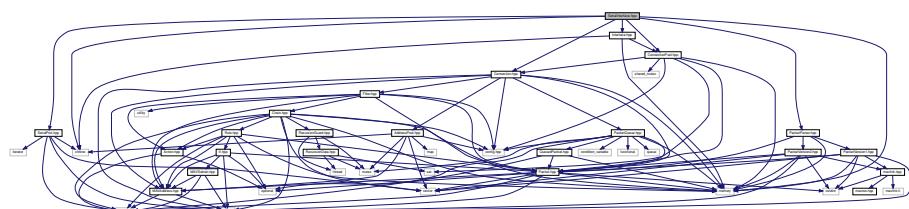
```

00093     for (auto byte : buffer)
00094     {
00095         auto packet = parser_.parse_byte(byte);
00096
00097         if (packet != nullptr)
00098         {
00099             packet->connection(connection_);
00100             connection_->add_address(packet->source());
00101             connection_pool_->send(std::move(packet));
00102         }
00103     }
00104 }
00105
00106
00107
00108 /** \copydoc Interface::print_(std::ostream &os) const
00109 *
00110 * Example:
00111 * ``
00112 * serial {
00113 *     device /dev/ttyUSB0;
00114 *     baudrate 115200;
00115 *     flow_control yes;
00116 * }
00117 * ``
00118 *
00119 * \param os The output stream to print to.
00120 */
00121 std::ostream &SerialInterface::print_(std::ostream &os) const
00122 {
00123     os << *port_;
00124     return os;
00125 }
```

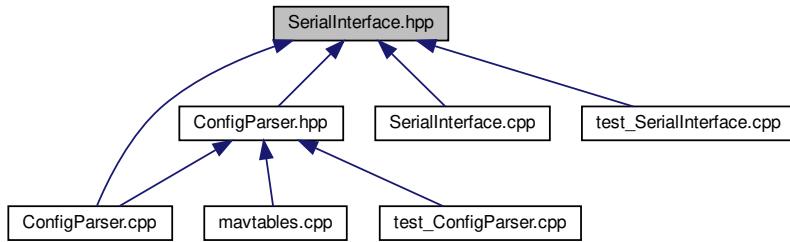
16.183 SerialInterface.hpp File Reference

```
#include <chrono>
#include <memory>
#include "Connection.hpp"
#include "ConnectionPool.hpp"
#include "Interface.hpp"
#include "PacketParser.hpp"
#include "SerialPort.hpp"
```

Include dependency graph for SerialInterface.hpp:



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



Classes

- class [SerialInterface](#)

16.184 SerialInterface.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef SERIALINTERFACE_HPP_
00019 #define SERIALINTERFACE_HPP_
00020
00021
00022 #include <chrono>
00023 #include <memory>
00024
00025 #include "Connection.hpp"
00026 #include "ConnectionPool.hpp"
00027 #include "Interface.hpp"
00028 #include "PacketParser.hpp"
00029 #include "SerialPort.hpp"
00030
00031
00032 /** A serial port interface.
00033 *
00034 * An interface (for sending and receiving packets) implementing the serial
00035 * port protocol.
00036 */
00037 class SerialInterface : public Interface
00038 {
00039     public:
00040         SerialInterface(
00041             std::unique_ptr<SerialPort> port,
00042             std::shared_ptr<ConnectionPool> connection_pool,
  
```

```

00043     std::unique_ptr<Connection> connection);
00044 // LCOV_EXCL_START
00045 ~SerialInterface() = default;
00046 // LCOV_EXCL_STOP
00047 void send_packet(const std::chrono::nanoseconds &timeout) final;
00048 void receive_packet(const std::chrono::nanoseconds &timeout) final;
00049
00050 protected:
00051     std::ostream &print_(std::ostream &os) const final;
00052
00053 private:
00054     // Variables.
00055     std::unique_ptr<SerialPort> port_;
00056     std::shared_ptr<ConnectionPool> connection_pool_;
00057     std::shared_ptr<Connection> connection_;
00058     PacketParser parser_;
00059 };
00060
00061
00062 #endif // SERIALINTERFACE_HPP_

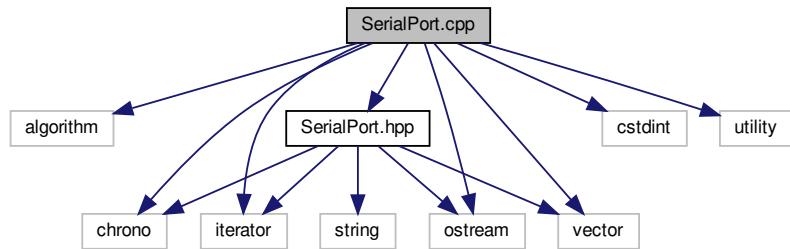
```

16.185 SerialPort.cpp File Reference

```

#include <algorithm>
#include <chrono>
#include <cstdint>
#include <iterator>
#include <ostream>
#include <utility>
#include <vector>
#include "SerialPort.hpp"
Include dependency graph for SerialPort.cpp:

```



16.186 SerialPort.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,

```

```
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017 #include <algorithm>
00018 #include <chrono>
00019 #include <cstdint>
00020 #include <iterator>
00021 #include <iostream>
00022 #include <utility>
00023 #include <vector>
00024
00025 #include "SerialPort.hpp"
00026
00027
00028
00029 // Placed here to avoid weak-vtables error.
00030 // LCOV_EXCL_START
00031 SerialPort::~SerialPort()
00032 {
00033 }
00034 // LCOV_EXCL_STOP
00035
00036
00037 /** Read data from the serial port.
00038 */
00039 * \note The \p timeout is not guaranteed to be up to nanosecond precision, the
00040 *       actual precision is up to the operating system's implementation but is
00041 *       guaranteed to have at least millisecond precision.
00042 *
00043 * \param timeout How long to wait for data to arrive on the serial port if
00044 *                 there is not already data to read. The default is to not wait.
00045 * \returns The data read from the serial port.
00046 */
00047 std::vector<uint8_t> SerialPort::read(const std::chrono::nanoseconds &timeout)
00048 {
00049     std::vector<uint8_t> vec;
00050     read(std::back_inserter(vec), timeout);
00051     return vec;
00052 }
00053
00054
00055 /** Read data from the serial port.
00056 */
00057 * \note The \p timeout is not guaranteed to be up to nanosecond precision, the
00058 *       actual precision is up to the operating system's implementation but is
00059 *       guaranteed to have at least millisecond precision.
00060 *
00061 * \param it A back insert iterator to read bytes into.
00062 * \param timeout How long to wait for data to arrive on the serial port if
00063 *                 there is not already data to read. The default is to not wait.
00064 */
00065 void SerialPort::read(
00066     std::back_insert_iterator<std::vector<uint8_t>> it,
00067     const std::chrono::nanoseconds &timeout)
00068 {
00069     auto vec = read(timeout);
00070     std::copy(vec.begin(), vec.end(), it);
00071 }
00072
00073
00074 /** Write data to the serial port (blocking write).
00075 */
00076 * \param data The bytes to send.
00077 */
00078 void SerialPort::write(const std::vector<uint8_t> &data)
00079 {
00080     write(data.begin(), data.end());
00081 }
00082
00083
00084 /** Write data to the serial port (blocking write).
00085 */
00086 * \param first Iterator to first byte in range to send.
00087 * \param last Iterator to one past the last byte to send.
00088 */
00089 void SerialPort::write(
00090     std::vector<uint8_t>::const_iterator first,
```

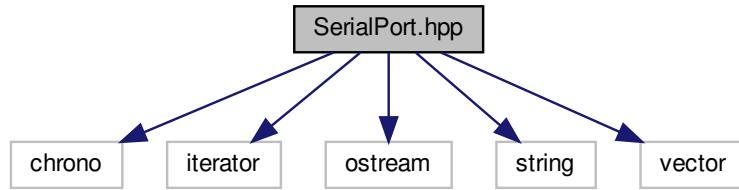
```

00091     std::vector<uint8_t>::const_iterator last)
00092 {
00093     std::vector<uint8_t> vec;
00094     std::copy(first, last, std::back_inserter(vec));
00095     write(vec);
00096 }
00097
00098
00099 /** Print the serial port to the given output stream.
00100 *
00101 * \param os The output stream to print to.
00102 * \returns The output stream.
00103 */
00104 std::ostream &SerialPort::print_(std::ostream &os) const
00105 {
00106     os << "unknown serial port";
00107     return os;
00108 }
00109
00110
00111 /** Print the given serial port to the given output stream.
00112 *
00113 * \note This is a polymorphic print, it will work on any child of \ref
00114 *       SerialPort even if the pointer/reference is to the base class \ref
00115 *       SerialPort.
00116 *
00117 * An example:
00118 * ``
00119 *   serial {
00120 *     device /dev/ttyUSB0;
00121 *     baudrate 115200;
00122 *     flow_control yes;
00123 *   }
00124 * ``
00125 *
00126 * The base \ref SerialPort class will print:
00127 * ``
00128 * unknown serial port
00129 * ``
00130
00131 * \relates SerialPort
00132 * \param os The output stream to print to.
00133 * \param serial_port The serial port (or any child of \ref SerialPort) to
00134 *       print.
00135 * \returns The output stream.
00136 */
00137 std::ostream &operator<<(std::ostream &os, const SerialPort &serial_port)
00138 {
00139     return serial_port.print_(os);
00140 }
```

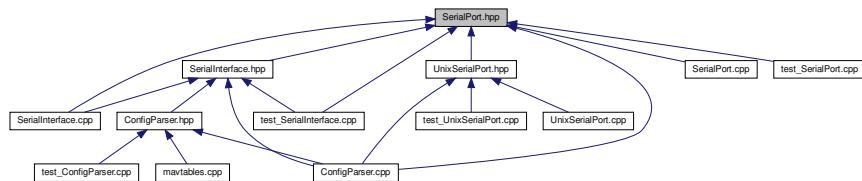
16.187 SerialPort.hpp File Reference

```
#include <chrono>
#include <iterator>
#include <iostream>
#include <string>
#include <vector>
```

Include dependency graph for SerialPort.hpp:



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



Classes

- class [SerialPort](#)

Functions

- `std::ostream & operator<< (std::ostream &os, const SerialPort &serial_port)`

16.187.1 Function Documentation

16.187.1.1 operator<<()

```

std::ostream& operator<< (
    std::ostream & os,
    const SerialPort & serial_port )
  
```

16.188 SerialPort.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef SERIALPORT_HPP_
00019 #define SERIALPORT_HPP_
00020
00021
00022 #include <chrono>
00023 #include <iterator>
00024 #include <ostream>
00025 #include <string>
00026 #include <vector>
00027
00028
00029 /** The base class of all serial port classes.
00030 */
00031 * This provides an abstraction of serial ports across operating systems.
00032 *
00033 * \warning This class should be treated as pure virtual and should never be
00034 * instantiated.
00035 *
00036 * \warning Either \ref read(const std::chrono::nanoseconds &) or
00037 * read(std::back_insert_iterator<std::vector<uint8_t>>, const std::chrono::nanoseconds &)
00038 * must be overridden in child classes to avoid infinite recursion.
00039 *
00040 * \warning Either \ref write(const std::vector<uint8_t> &data) or
00041 * write(std::vector<uint8_t>::const_iterator, std::vector<uint8_t>::const_iterator)
00042 * must be overridden in child classes to avoid infinite recursion.
00043 */
00044 class SerialPort
00045 {
00046     public:
00047         /** Parity options.
00048         */
00049         enum Parity
00050     {
00051             NONE,    //!< No parity.
00052             ODD,     //!< Odd parity, must have odd number of set bits.
00053             EVEN,   //!< Even parity, must have even number of set bits.
00054             MARK,   //!< Fill parity bit with 1.
00055             SPACE   //!< Fill parity bit with 0.
00056         };
00057         /** Feature bitflags.
00058         */
00059         enum Feature
00060     {
00061         DEFAULT = 0,           //!< No special features.
00062         FLOW_CONTROL = 1 << 0 //!< Enable flow control.
00063     };
00064     virtual ~SerialPort();
00065     virtual std::vector<uint8_t> read(
00066         const std::chrono::nanoseconds &timeout =
00067             std::chrono::nanoseconds::zero());
00068     virtual void read(
00069         std::back_insert_iterator<std::vector<uint8_t>> it,
00070         const std::chrono::nanoseconds &timeout =
00071             std::chrono::nanoseconds::zero());
00072     virtual void write(const std::vector<uint8_t> &data);
00073     virtual void write(
00074         std::vector<uint8_t>::const_iterator first,
00075         std::vector<uint8_t>::const_iterator last);
00076
00077     friend std::ostream &operator<<
```

```

00078         std::ostream &os, const SerialPort &serial_port);
00079
00080     protected:
00081         virtual std::ostream &print_(std::ostream &os) const;
00082     };
00083
00084
00085     std::ostream &operator<<(std::ostream &os, const SerialPort &serial_port);
00086
00087
00088 #endif // SERIALPORT_HPP_

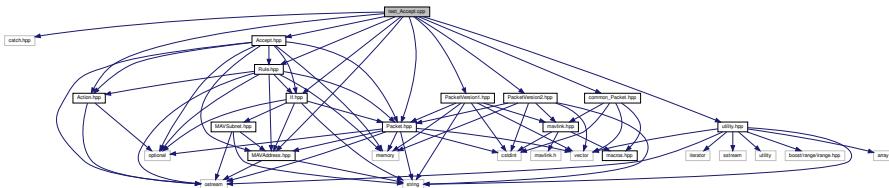
```

16.189 test_Accept.cpp File Reference

```

#include <catch.hpp>
#include "Accept.hpp"
#include "Action.hpp"
#include "If.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "PacketVersion1.hpp"
#include "PacketVersion2.hpp"
#include "Rule.hpp"
#include "utility.hpp"
#include "common_Packet.hpp"
Include dependency graph for test_Accept.cpp:

```



Functions

- [TEST_CASE \("Accept's are constructable.", "\[Accept\]"\)](#)
- [TEST_CASE \("Accept's are comparable.", "\[Accept\]"\)](#)
- [TEST_CASE \("Accept's 'action' method determines what to do with a " "packet/address combination.", "\[Accept\]"\)](#)
- [TEST_CASE \("Accept's are printable \(without a condition or a priority\).", "\[Accept\]"\)](#)
- [TEST_CASE \("Accept's are printable \(without a condition but with a priority\).", "\[Accept\]"\)](#)
- [TEST_CASE \("Accept's are printable \(with a condition but without a priority\).", "\[Accept\]"\)](#)
- [TEST_CASE \("Accept's are printable \(with a condition and a priority\).", "\[Accept\]"\)](#)
- [TEST_CASE \("Accept's 'clone' method returns a polymorphic copy.", "\[Accept\]"\)](#)

16.189.1 Function Documentation

16.189.1.1 TEST_CASE() [1/8]

```
TEST_CASE (
    "Accept's are constructable." ,
    "" [Accept] )
```

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

16.189.1.2 TEST_CASE() [2/8]

```
TEST_CASE (
    "Accept's are comparable." ,
    "" [Accept] )
```

Definition at line 62 of file [test_Accept.cpp](#).

16.189.1.3 TEST_CASE() [3/8]

```
TEST_CASE (
    "Accept's 'action' method determines what to do with a \"packet/address combination.\""
,
    "" [Accept] )
```

Definition at line 93 of file [test_Accept.cpp](#).

References [ping](#).

16.189.1.4 TEST_CASE() [4/8]

```
TEST_CASE (
    "Accept's are printable (without a condition or a priority)." ,
    "" [Accept] )
```

Definition at line 145 of file [test_Accept.cpp](#).

References [ping](#), and [rule](#).

16.189.1.5 TEST_CASE() [5/8]

```
TEST_CASE (
    "Accept's are printable (without a condition but with a priority).",
    "" {Accept} )
```

Definition at line 162 of file [test_Accept.cpp](#).

References [ping](#), and [rule](#).

16.189.1.6 TEST_CASE() [6/8]

```
TEST_CASE (
    "Accept's are printable (with a condition but without a priority).",
    "" {Accept} )
```

Definition at line 179 of file [test_Accept.cpp](#).

References [ping](#), and [rule](#).

16.189.1.7 TEST_CASE() [7/8]

```
TEST_CASE (
    "Accept's are printable (with a condition and a priority).",
    "" {Accept} )
```

Definition at line 196 of file [test_Accept.cpp](#).

References [ping](#), and [rule](#).

16.189.1.8 TEST_CASE() [8/8]

```
TEST_CASE (
    "Accept's 'clone' method returns a polymorphic copy.",
    "" {Accept} )
```

Definition at line 217 of file [test_Accept.cpp](#).

16.190 test_Accept.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <catch.hpp>
00019
00020 #include "Accept.hpp"
00021 #include "Action.hpp"
00022 #include "If.hpp"
00023 #include "MAVAddress.hpp"
00024 #include "Packet.hpp"
00025 #include "PacketVersion1.hpp"
00026 #include "PacketVersion2.hpp"
00027 #include "Rule.hpp"
00028 #include "utility.hpp"
00029
00030 #include "common_Packet.hpp"
00031
00032
00033 TEST_CASE("Accept's are constructable.", "[Accept]")
00034 {
00035     SECTION("Without a condition (match all packet/address combinations) or a "
00036             "priority.")
00037     {
00038         REQUIRE_NO_THROW(Accept());
00039     }
00040     SECTION("Without a condition (match all packet/address combinations) but "
00041             "with a priority.")
00042     {
00043         REQUIRE_NO_THROW(Accept(3));
00044     }
00045     SECTION("With a condition and without a priority.")
00046     {
00047         REQUIRE_NO_THROW(Accept(If()));
00048         REQUIRE_NO_THROW(Accept(If().type("PING")));
00049         REQUIRE_NO_THROW(Accept(If().from("192.168")));
00050         REQUIRE_NO_THROW(Accept(If().to("172.16")));
00051     }
00052     SECTION("With both a condition and a priority.")
00053     {
00054         REQUIRE_NO_THROW(Accept(3, If()));
00055         REQUIRE_NO_THROW(Accept(3, If().type("PING")));
00056         REQUIRE_NO_THROW(Accept(3, If().from("192.168")));
00057         REQUIRE_NO_THROW(Accept(3, If().to("172.16")));
00058     }
00059 }
00060
00061
00062 TEST_CASE("Accept's are comparable.", "[Accept]")
00063 {
00064     SECTION("with ==")
00065     {
00066         REQUIRE(Accept() == Accept());
00067         REQUIRE(Accept(If().type("PING")) == Accept(If().type("PING")));
00068         REQUIRE(Accept(3) == Accept(3));
00069         REQUIRE(Accept(3, If().type("PING")) == Accept(3, If().type("PING")));
00070         REQUIRE_FALSE(
00071             Accept(If().type("PING")) == Accept(If().type("SET_MODE")));
00072         REQUIRE_FALSE(Accept(If().type("PING")) == Accept(If()));
00073         REQUIRE_FALSE(Accept(If().type("PING")) == Accept());
00074         REQUIRE_FALSE(Accept(3) == Accept(-3));
00075         REQUIRE_FALSE(Accept(3) == Accept());
00076     }
00077     SECTION("with !=")
```

```

00078    {
00079        REQUIRE(Accept(If().type("PING")) != Accept());
00080        REQUIRE(Accept(If().type("PING")) != Accept(If()));
00081        REQUIRE(Accept(If().type("PING")) != Accept(If().type("SET_MODE")));
00082        REQUIRE(Accept(3) != Accept(-3));
00083        REQUIRE(Accept(3) != Accept());
00084        REQUIRE_FALSE(Accept() != Accept());
00085        REQUIRE_FALSE(Accept(If().type("PING")) != Accept(If().type("PING")));
00086        REQUIRE_FALSE(Accept(3) != Accept(3));
00087        REQUIRE_FALSE(
00088            Accept(3, If().type("PING")) != Accept(3, If().type("PING")));
00089    }
00090 }
00091
00092
00093 TEST_CASE("Accept's 'action' method determines what to do with a "
00094         "packet/address combination.", "[Accept]")
00095 {
00096     auto ping = packet_v2::Packet(to_vector(PingV2()));
00097     SECTION("Returns the accept action if there is no conditional.")
00098     {
00099         // Without priority.
00100         REQUIRE(
00101             Accept().action(ping, MAVAddress("192.168")) ==
00102             Action::make_accept());
00103         // With priority.
00104         REQUIRE(
00105             Accept(3).action(ping, MAVAddress("192.168")) ==
00106             Action::make_accept(3));
00107     }
00108     SECTION("Returns the accept action if the conditional is a match.")
00109     {
00110         // Without priority.
00111         REQUIRE(
00112             Accept(If().type("PING")).action(
00113                 ping, MAVAddress("192.168")) ==
00114             Action::make_accept());
00115         REQUIRE(
00116             Accept(If().to("192.168")).action(
00117                 ping, MAVAddress("192.168")) ==
00118             Action::make_accept());
00119         // With priority.
00120         REQUIRE(
00121             Accept(3, If().type("PING")).action(
00122                 ping, MAVAddress("192.168")) ==
00123             Action::make_accept(3));
00124     }
00125     SECTION("Returns the continue action if the conditional does not match.")
00126     {
00127         // Without priority.
00128         REQUIRE(
00129             Accept(If().type("SET_MODE")).action(
00130                 ping, MAVAddress("192.168")) ==
00131             Action::make_continue());
00132         REQUIRE(
00133             Accept(If().to("172.16")).action(
00134                 ping, MAVAddress("192.168")) ==
00135             Action::make_continue());
00136         // With priority.
00137         REQUIRE(
00138             Accept(3, If().type("SET_MODE")).action(
00139                 ping, MAVAddress("192.168")) ==
00140             Action::make_continue());
00141     }
00142 }
00143
00144
00145 TEST_CASE("Accept's are printable (without a condition or a priority).",
00146         "[Accept]")
00147 {
00148     auto ping = packet_v2::Packet(to_vector(PingV2()));
00149     Accept accept;
00150     Rule &rule = accept;

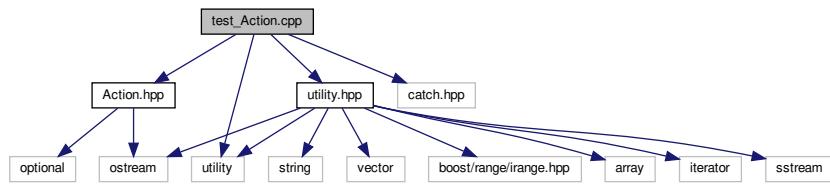
```

```
00151     SECTION("By direct type.")
00152     {
00153         REQUIRE(str(accept) == "accept");
00154     }
00155     SECTION("By polymorphic type.")
00156     {
00157         REQUIRE(str(rule) == "accept");
00158     }
00159 }
00160
00161
00162 TEST_CASE("Accept's are printable (without a condition but with a priority).",
00163             "[Accept]")
00164 {
00165     auto ping = packet_v2::Packet(to_vector(PingV2()));
00166     Accept accept(-3);
00167     Rule &rule = accept;
00168     SECTION("By direct type.")
00169     {
00170         REQUIRE(str(accept) == "accept with priority -3");
00171     }
00172     SECTION("By polymorphic type.")
00173     {
00174         REQUIRE(str(rule) == "accept with priority -3");
00175     }
00176 }
00177
00178
00179 TEST_CASE("Accept's are printable (with a condition but without a priority).",
00180             "[Accept]")
00181 {
00182     auto ping = packet_v2::Packet(to_vector(PingV2()));
00183     Accept accept(If().type("PING").from("192.168/8").to("172.16/4"));
00184     Rule &rule = accept;
00185     SECTION("By direct type.")
00186     {
00187         REQUIRE(str(accept) == "accept if PING from 192.168/8 to 172.16/4");
00188     }
00189     SECTION("By polymorphic type.")
00190     {
00191         REQUIRE(str(rule) == "accept if PING from 192.168/8 to 172.16/4");
00192     }
00193 }
00194
00195
00196 TEST_CASE("Accept's are printable (with a condition and a priority).",
00197             "[Accept]")
00198 {
00199     auto ping = packet_v2::Packet(to_vector(PingV2()));
00200     Accept accept(-3, If().type("PING").from("192.168/8").to("172.16/4"));
00201     Rule &rule = accept;
00202     SECTION("By direct type.")
00203     {
00204         REQUIRE(
00205             str(accept) ==
00206             "accept with priority -3 if PING from 192.168/8 to 172.16/4");
00207     }
00208     SECTION("By polymorphic type.")
00209     {
00210         REQUIRE(
00211             str(rule) ==
00212             "accept with priority -3 if PING from 192.168/8 to 172.16/4");
00213     }
00214 }
00215
00216
00217 TEST_CASE("Accept's 'clone' method returns a polymorphic copy.", "[Accept]")
00218 {
00219     SECTION("Without a priority.")
00220     {
00221         Accept accept(If().type("PING"));
00222         Rule &rule = accept;
00223         std::unique_ptr<Rule> polymorphic_copy = rule.clone();
00224         REQUIRE(accept == *polymorphic_copy);
00225     }
00226     SECTION("With a priority.")
00227     {
00228         Accept accept(4, If().type("PING"));
00229         Rule &rule = accept;
00230         std::unique_ptr<Rule> polymorphic_copy = rule.clone();
00231         REQUIRE(accept == *polymorphic_copy);
```

```
00232     }
00233 }
```

16.191 test_Action.cpp File Reference

```
#include <utility>
#include <catch.hpp>
#include "Action.hpp"
#include "utility.hpp"
Include dependency graph for test_Action.cpp:
```



Functions

- [TEST_CASE](#) ("Action's 'make_accept' factory method constructs an ACCEPT action.", "[[Action](#)]"")
- [TEST_CASE](#) ("Action's 'make_reject' factory method constructs a REJECT action.", "[[Action](#)]"")
- [TEST_CASE](#) ("Action's 'make_continue' factory method constructs a CONTINUE "action.", "[[Action](#)]"")
- [TEST_CASE](#) ("Action's 'make_default' factory method constructs a DEFAULT action.", "[[Action](#)]"")
- [TEST_CASE](#) ("Action's 'priority' method sets and gets the priority.", "[[Action](#)]"")
- [TEST_CASE](#) ("Action's are comparable.", "[[Action](#)]"")
- [TEST_CASE](#) ("Action's are copyable.", "[[Action](#)]"")
- [TEST_CASE](#) ("Action's are movable.", "[[Action](#)]"")
- [TEST_CASE](#) ("Action's are assignable.", "[[Action](#)]"")
- [TEST_CASE](#) ("Action's are printable.")

Variables

- auto [action_b](#) = Action::make_accept(100)
- [action_a](#) = std::move([action_b](#))

16.191.1 Function Documentation

16.191.1.1 TEST_CASE() [1/10]

```
TEST_CASE (
    "Action's 'make_accept' factory method constructs an ACCEPT action." ,
    "" [Action] )
```

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

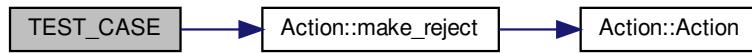
16.191.1.2 TEST_CASE() [2/10]

```
TEST_CASE (
    "Action's 'make_reject' factory method constructs a REJECT action." ,
    "" [Action] )
```

Definition at line 44 of file [test_Action.cpp](#).

References [Action::make_reject\(\)](#).

Here is the call graph for this function:

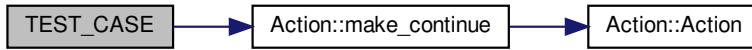
**16.191.1.3 TEST_CASE() [3/10]**

```
TEST_CASE (
    "Action's 'make_continue' factory method constructs a CONTINUE \"action.\" ,
    "" [Action] )
```

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

References [Action::make_continue\(\)](#).

Here is the call graph for this function:



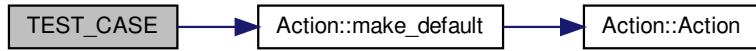
16.191.1.4 TEST_CASE() [4/10]

```
TEST_CASE (
    "Action's 'make_default' factory method constructs a DEFAULT action." ,
    "" [Action] )
```

Definition at line 62 of file [test_Action.cpp](#).

References [Action::make_default\(\)](#).

Here is the call graph for this function:

**16.191.1.5 TEST_CASE()** [5/10]

```
TEST_CASE (
    "Action's 'priority' method sets and gets the priority." ,
    "" [Action] )
```

Definition at line 71 of file [test_Action.cpp](#).

16.191.1.6 TEST_CASE() [6/10]

```
TEST_CASE (
    "Action's are comparable." ,
    "" [Action] )
```

Definition at line 105 of file [test_Action.cpp](#).

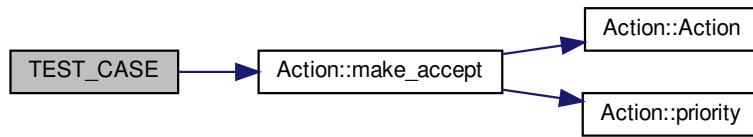
16.191.1.7 TEST_CASE() [7/10]

```
TEST_CASE (
    "Action's are copyable." ,
    "" [Action] )
```

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

References [Action::make_accept\(\)](#).

Here is the call graph for this function:



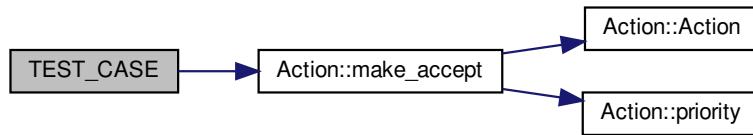
16.191.1.8 TEST_CASE() [8/10]

```
TEST_CASE (
    "Action's are movable." ,
    "" [Action] )
```

Definition at line 138 of file [test_Action.cpp](#).

References [Action::make_accept\(\)](#).

Here is the call graph for this function:



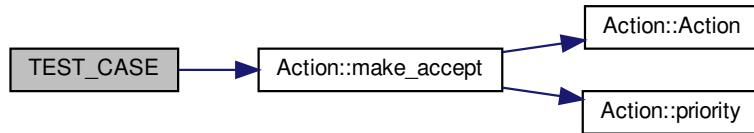
16.191.1.9 TEST_CASE() [9/10]

```
TEST_CASE (
    "Action's are assignable." ,
    "" [Action] )
```

Definition at line 146 of file [test_Action.cpp](#).

References [action_a](#), [action_b](#), and [Action::make_accept\(\)](#).

Here is the call graph for this function:

**16.191.1.10 TEST_CASE() [10/10]**

```
TEST_CASE (
    "Action's are printable." )
```

Definition at line 166 of file [test_Action.cpp](#).

16.191.2 Variable Documentation**16.191.2.1 action_a**

```
action_a = std::move(action_b)
```

Definition at line 161 of file [test_Action.cpp](#).

16.191.2.2 action_b

```
auto action_b = Action::make_accept(100)
```

Definition at line 159 of file [test_Action.cpp](#).

16.192 test_Action.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <utility>
00019
00020 #include <catch.hpp>
00021
00022 #include "Action.hpp"
00023 #include "utility.hpp"
00024
00025
00026 TEST_CASE("Action's 'make_accept' factory method constructs an ACCEPT action.",
00027             "[Action]")
00028 {
00029     SECTION("Without a priority.")
00030     {
00031         auto result = Action::make_accept();
00032         REQUIRE(result.action() == Action::ACCEPT);
00033         REQUIRE(result.priority() == 0);
00034     }
00035     SECTION("With a priority.")
00036     {
00037         auto result = Action::make_accept(-10);
00038         REQUIRE(result.action() == Action::ACCEPT);
00039         REQUIRE(result.priority() == -10);
00040     }
00041 }
00042
00043
00044 TEST_CASE("Action's 'make_reject' factory method constructs a REJECT action.",
00045             "[Action]")
00046 {
00047     auto result = Action::make_reject();
00048     REQUIRE(result.action() == Action::REJECT);
00049     REQUIRE(result.priority() == 0);
00050 }
00051
00052
00053 TEST_CASE("Action's 'make_continue' factory method constructs a CONTINUE "
00054             "action.", "[Action]")
00055 {
00056     auto result = Action::make_continue();
00057     REQUIRE(result.action() == Action::CONTINUE);
00058     REQUIRE(result.priority() == 0);
00059 }
00060
00061
00062 TEST_CASE("Action's 'make_default' factory method constructs a DEFAULT action.",
00063             "[Action]")
00064 {
00065     auto result = Action::make_default();
```

```
00066     REQUIRE(result.action() == Action::DEFAULT);
00067     REQUIRE(result.priority() == 0);
00068 }
00069
00070
00071 TEST_CASE("Action's 'priority' method sets and gets the priority.", "[Action]")
00072 {
00073     SECTION("Can be set exactly once for accept results without a priority.")
00074     {
00075         auto result = Action::make_accept();
00076         REQUIRE(result.priority() == 0);
00077         result.priority(10);
00078         REQUIRE(result.priority() == 10);
00079         result.priority(100);
00080         REQUIRE(result.priority() == 10);
00081     }
00082     SECTION("Cannot be set for accept results with a priority.")
00083     {
00084         auto result = Action::make_accept(10);
00085         REQUIRE(result.priority() == 10);
00086         result.priority(100);
00087         REQUIRE(result.priority() == 10);
00088     }
00089     SECTION("Cannot be set on reject, continue, or default actions.")
00090     {
00091         auto reject = Action::make_reject();
00092         reject.priority(10);
00093         REQUIRE(reject.priority() == 0);
00094         auto continue_ = Action::make_continue();
00095         continue_.priority(10);
00096         REQUIRE(continue_.priority() == 0);
00097         auto default_ = Action::make_default();
00098         default_.priority(10);
00099         REQUIRE(default_.priority() == 0);
00100     }
00101 }
00102 }
00103
00104
00105 TEST_CASE("Action's are comparable.", "[Action]")
00106 {
00107     SECTION("with ==")
00108     {
00109         REQUIRE(Action::make_accept() == Action::make_accept());
00110         REQUIRE(Action::make_accept(10) ==
00111             Action::make_accept(10));
00112         REQUIRE_FALSE(Action::make_accept(1) ==
00113             Action::make_accept());
00114         REQUIRE_FALSE(Action::make_accept(1) ==
00115             Action::make_accept(-1));
00116         REQUIRE_FALSE(Action::make_accept() ==
00117             Action::make_reject());
00118         REQUIRE_FALSE(Action::make_accept() ==
00119             Action::make_continue());
00120         REQUIRE_FALSE(Action::make_accept() ==
00121             Action::make_default());
00122     }
00123     SECTION("with !=")
00124     {
00125         REQUIRE(Action::make_accept(1) != Action::make_accept());
00126         REQUIRE(Action::make_accept(1) != Action::make_accept(-1));
00127         REQUIRE(Action::make_accept() != Action::make_reject());
00128         REQUIRE(Action::make_accept() !=
00129             Action::make_continue());
00130         REQUIRE(Action::make_accept() != Action::make_default());
00131     }
00132     REQUIRE_FALSE(Action::make_accept() != Action::make_accept());
00133     REQUIRE(Action::make_accept() != Action::make_accept());
00134     REQUIRE(copy == Action::make_accept(10));
00135 }
00136
00137
```

```

00138 TEST_CASE("Action's are movable.", "[Action]")
00139 {
00140     auto original = Action::make_accept(10);
00141     auto moved(std::move(original));
00142     REQUIRE(moved == Action::make_accept(10));
00143 }
00144
00145
00146 TEST_CASE("Action's are assignable.", "[Action]")
00147 {
00148     auto action_a = Action::make_accept(-10);
00149     auto action_b = Action::make_accept(100);
00150     REQUIRE(action_a == Action::make_accept(-10));
00151     action_a = action_b;
00152     REQUIRE(action_a == Action::make_accept(100));
00153 }
00154
00155
00156 TEST_CASE("Action's are assignable (by move semantics).", "[Action]")
00157 {
00158     auto action_a = Action::make_accept(-10);
00159     auto action_b = Action::make_accept(100);
00160     REQUIRE(action_a == Action::make_accept(-10));
00161     action_a = std::move(action_b);
00162     REQUIRE(action_a == Action::make_accept(100));
00163 }
00164
00165
00166 TEST_CASE("Action's are printable.")
00167 {
00168     REQUIRE(str(Action::make_accept()) == "accept");
00169     REQUIRE(str(Action::make_accept(-10)) == "accept with priority -10");
00170     REQUIRE(str(Action::make_accept(10)) == "accept with priority 10");
00171     REQUIRE(str(Action::make_reject()) == "reject");
00172     REQUIRE(str(Action::make_continue()) == "continue");
00173     REQUIRE(str(Action::make_default()) == "default");
00174 }

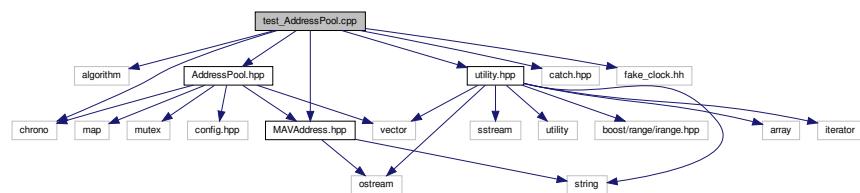
```

16.193 test_AddressPool.cpp File Reference

```

#include <algorithm>
#include <chrono>
#include <catch.hpp>
#include <fake_clock.hh>
#include <AddressPool.hpp>
#include <MAVAddress.hpp>
#include "utility.hpp"
Include dependency graph for test_AddressPool.cpp:

```



Functions

- `TEST_CASE ("AddressPool's can be constructed.", "[AddressPool]")`

- [TEST_CASE](#) ("AddressPool's 'add' method adds an address to the pool.", "[AddressPool]")
- [TEST_CASE](#) ("AddressPool's 'contains' method determines whether an address is " "in the pool or not.", "[AddressPool]")
- [TEST_CASE](#) ("AddressPool removes expired addresses.", "[AddressPool]")

16.193.1 Function Documentation

16.193.1.1 TEST_CASE() [1/4]

```
TEST_CASE (
    "AddressPool's can be constructed." ,
    "" [AddressPool] )
```

Definition at line [34](#) of file [test_AddressPool.cpp](#).

16.193.1.2 TEST_CASE() [2/4]

```
TEST_CASE (
    "AddressPool's 'add' method adds an address to the pool." ,
    "" [AddressPool] )
```

Definition at line [41](#) of file [test_AddressPool.cpp](#).

16.193.1.3 TEST_CASE() [3/4]

```
TEST_CASE (
    "AddressPool's 'contains' method determines whether an address is " "in the pool or
not." ,
    "" [AddressPool] )
```

Definition at line [81](#) of file [test_AddressPool.cpp](#).

16.193.1.4 TEST_CASE() [4/4]

```
TEST_CASE (
    "AddressPool removes expired addresses." ,
    "" [AddressPool] )
```

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

16.194 test_AddressPool.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <algorithm>
00019 #include <chrono>
00020
00021 #include <catch.hpp>
00022 #include <fake_clock.hh>
00023
00024 #include <AddressPool.hpp>
00025 #include <MAVAddress.hpp>
00026
00027 #include "utility.hpp"
00028
00029
00030 using namespace std::chrono_literals;
00031 using namespace testing;
00032
00033
00034 TEST_CASE("AddressPool's can be constructed.", "[AddressPool]")
00035 {
00036     REQUIRE_NO_THROW(AddressPool<>());
00037     REQUIRE_NO_THROW(AddressPool<>(10s));
00038 }
00039
00040
00041 TEST_CASE("AddressPool's 'add' method adds an address to the pool.",
00042             "[AddressPool]")
00043 {
00044     SECTION("With fake_clock.")
00045     {
00046         AddressPool<fake_clock> pool;
00047         pool.add(MAVAddress("192.168"));
00048         pool.add(MAVAddress("172.16"));
00049         pool.add(MAVAddress("10.10"));
00050         auto addr = pool.addresses();
00051         std::sort(addr.begin(), addr.end(), std::greater<MAVAddress>());
00052         REQUIRE(addr.size() == 3);
00053         std::vector<MAVAddress> compare =
00054         {
00055             MAVAddress("192.168"),
00056             MAVAddress("172.16"),
00057             MAVAddress("10.10")
00058         };
00059         REQUIRE(addr == compare);
00060     }
00061     SECTION("With std::chrono::steady_clock.") // for complete coverage
00062     {
00063         AddressPool<std::chrono::steady_clock> pool;
00064         pool.add(MAVAddress("192.168"));
00065         pool.add(MAVAddress("172.16"));
00066         pool.add(MAVAddress("10.10"));
00067         auto addr = pool.addresses();
00068         std::sort(addr.begin(), addr.end(), std::greater<MAVAddress>());
00069         REQUIRE(addr.size() == 3);
00070         std::vector<MAVAddress> compare =
00071         {
00072             MAVAddress("192.168"),
00073             MAVAddress("172.16"),
00074             MAVAddress("10.10")
00075         };
00076         REQUIRE(addr == compare);
00077     }
```

```

00078 }
00079
00080
00081 TEST_CASE("AddressPool's 'contains' method determines whether an address is "
00082     "in the pool or not.", "[AddressPool]")
00083 {
00084     SECTION("With fake_clock.")
00085     {
00086         AddressPool<fake_clock> pool;
00087         pool.add(MAVAddress("192.168"));
00088         pool.add(MAVAddress("172.16"));
00089         pool.add(MAVAddress("10.10"));
00090         REQUIRE(pool.contains(MAVAddress("192.168")));
00091         REQUIRE(pool.contains(MAVAddress("172.16")));
00092         REQUIRE(pool.contains(MAVAddress("10.10")));
00093         REQUIRE_FALSE(pool.contains(MAVAddress("0.0")));
00094     }
00095     SECTION("With std::chrono::steady_clock.") // for complete coverage
00096     {
00097         AddressPool<std::chrono::steady_clock> pool;
00098         pool.add(MAVAddress("192.168"));
00099         pool.add(MAVAddress("172.16"));
00100         pool.add(MAVAddress("10.10"));
00101         REQUIRE(pool.contains(MAVAddress("192.168")));
00102         REQUIRE(pool.contains(MAVAddress("172.16")));
00103         REQUIRE(pool.contains(MAVAddress("10.10")));
00104         REQUIRE_FALSE(pool.contains(MAVAddress("0.0")));
00105     }
00106 }
00107
00108
00109 TEST_CASE("AddressPool removes expired addresses.", "[AddressPool]")
00110 {
00111     SECTION("When using the 'contains' method (and default timeout of 2 min).")
00112     {
00113         AddressPool<fake_clock> pool;
00114         pool.add(MAVAddress("0.0"));
00115         fake_clock::advance(1s); // 00:00:01
00116         pool.add(MAVAddress("1.1"));
00117         fake_clock::advance(1s); // 00:00:02
00118         pool.add(MAVAddress("2.2"));
00119         fake_clock::advance(1s); // 00:00:03
00120         pool.add(MAVAddress("3.3"));
00121         REQUIRE(pool.contains(MAVAddress("0.0")));
00122         REQUIRE(pool.contains(MAVAddress("1.1")));
00123         REQUIRE(pool.contains(MAVAddress("2.2")));
00124         REQUIRE(pool.contains(MAVAddress("3.3")));
00125         fake_clock::advance(117s); // 00:02:00
00126         REQUIRE(pool.contains(MAVAddress("0.0")));
00127         REQUIRE(pool.contains(MAVAddress("1.1")));
00128         REQUIRE(pool.contains(MAVAddress("2.2")));
00129         REQUIRE(pool.contains(MAVAddress("3.3")));
00130         fake_clock::advance(1s); // 00:02:01
00131         REQUIRE_FALSE(pool.contains(MAVAddress("0.0")));
00132         REQUIRE(pool.contains(MAVAddress("1.1")));
00133         REQUIRE(pool.contains(MAVAddress("2.2")));
00134         REQUIRE(pool.contains(MAVAddress("3.3")));
00135         fake_clock::advance(1s); // 00:02:02
00136         REQUIRE_FALSE(pool.contains(MAVAddress("0.0")));
00137         REQUIRE_FALSE(pool.contains(MAVAddress("1.1")));
00138         REQUIRE(pool.contains(MAVAddress("2.2")));
00139         REQUIRE(pool.contains(MAVAddress("3.3")));
00140         fake_clock::advance(1s); // 00:02:03
00141         REQUIRE_FALSE(pool.contains(MAVAddress("0.0")));
00142         REQUIRE_FALSE(pool.contains(MAVAddress("1.1")));
00143         REQUIRE_FALSE(pool.contains(MAVAddress("2.2")));
00144         REQUIRE(pool.contains(MAVAddress("3.3")));
00145         fake_clock::advance(1s); // 00:02:04
00146         REQUIRE_FALSE(pool.contains(MAVAddress("0.0")));
00147         REQUIRE_FALSE(pool.contains(MAVAddress("1.1")));
00148         REQUIRE_FALSE(pool.contains(MAVAddress("2.2")));
00149         REQUIRE_FALSE(pool.contains(MAVAddress("3.3")));
00150     }
00151     SECTION("When using the 'contains' method (and a custom timeout).")
00152     {
00153         AddressPool<fake_clock> pool(1h);
00154         pool.add(MAVAddress("0.0"));
00155         fake_clock::advance(1s); // 00:00:01
00156         pool.add(MAVAddress("1.1"));
00157         fake_clock::advance(1s); // 00:00:02
00158         pool.add(MAVAddress("2.2"));

```

```
00159     fake_clock::advance(1s); // 00:00:03
00160     pool.add(MAVAddress("3.3"));
00161     REQUIRE(pool.contains(MAVAddress("0.0")));
00162     REQUIRE(pool.contains(MAVAddress("1.1")));
00163     REQUIRE(pool.contains(MAVAddress("2.2")));
00164     REQUIRE(pool.contains(MAVAddress("3.3")));
00165     fake_clock::advance(3597s); // 01:00:00
00166     REQUIRE(pool.contains(MAVAddress("0.0")));
00167     REQUIRE(pool.contains(MAVAddress("1.1")));
00168     REQUIRE(pool.contains(MAVAddress("2.2")));
00169     REQUIRE(pool.contains(MAVAddress("3.3")));
00170     fake_clock::advance(1s); // 01:00:01
00171     REQUIRE_FALSE(pool.contains(MAVAddress("0.0")));
00172     REQUIRE(pool.contains(MAVAddress("1.1")));
00173     REQUIRE(pool.contains(MAVAddress("2.2")));
00174     REQUIRE(pool.contains(MAVAddress("3.3")));
00175     fake_clock::advance(1s); // 01:00:02
00176     REQUIRE_FALSE(pool.contains(MAVAddress("0.0")));
00177     REQUIRE_FALSE(pool.contains(MAVAddress("1.1")));
00178     REQUIRE(pool.contains(MAVAddress("2.2")));
00179     REQUIRE(pool.contains(MAVAddress("3.3")));
00180     fake_clock::advance(1s); // 01:00:03
00181     REQUIRE_FALSE(pool.contains(MAVAddress("0.0")));
00182     REQUIRE_FALSE(pool.contains(MAVAddress("1.1")));
00183     REQUIRE_FALSE(pool.contains(MAVAddress("2.2")));
00184     REQUIRE(pool.contains(MAVAddress("3.3")));
00185     fake_clock::advance(1s); // 01:00:04
00186     REQUIRE_FALSE(pool.contains(MAVAddress("0.0")));
00187     REQUIRE_FALSE(pool.contains(MAVAddress("1.1")));
00188     REQUIRE_FALSE(pool.contains(MAVAddress("2.2")));
00189     REQUIRE_FALSE(pool.contains(MAVAddress("3.3")));
00190 }
00191 SECTION("When using the 'addresses' method (and default timeout of 2 min.)")
00192 {
00193     AddressPool<fake_clock> pool;
00194     pool.add(MAVAddress("0.0"));
00195     fake_clock::advance(1s); // 00:00:01
00196     pool.add(MAVAddress("1.1"));
00197     fake_clock::advance(1s); // 00:00:02
00198     pool.add(MAVAddress("2.2"));
00199     fake_clock::advance(1s); // 00:00:03
00200     pool.add(MAVAddress("3.3"));
00201     {
00202         std::vector<MAVAddress> vec =
00203         {
00204             MAVAddress("0.0"),
00205             MAVAddress("1.1"),
00206             MAVAddress("2.2"),
00207             MAVAddress("3.3"),
00208         };
00209         auto addr = pool.addresses();
00210         std::sort(addr.begin(), addr.end());
00211         REQUIRE(vec == addr);
00212     }
00213     fake_clock::advance(117s); // 00:02:00
00214     {
00215         std::vector<MAVAddress> vec =
00216         {
00217             MAVAddress("0.0"),
00218             MAVAddress("1.1"),
00219             MAVAddress("2.2"),
00220             MAVAddress("3.3"),
00221         };
00222         auto addr = pool.addresses();
00223         std::sort(addr.begin(), addr.end());
00224         REQUIRE(vec == addr);
00225     }
00226     fake_clock::advance(1s); // 00:02:01
00227     {
00228         std::vector<MAVAddress> vec =
00229         {
00230             MAVAddress("1.1"),
00231             MAVAddress("2.2"),
00232             MAVAddress("3.3"),
00233         };
00234         auto addr = pool.addresses();
00235         std::sort(addr.begin(), addr.end());
00236         REQUIRE(vec == addr);
00237     }
00238     fake_clock::advance(1s); // 00:02:02
00239 }
```

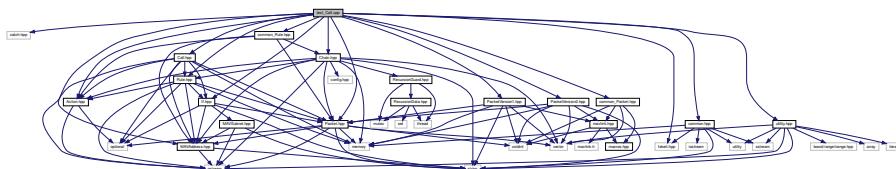
```
00240         std::vector<MAVAddress> vec =
00241         {
00242             MAVAddress("2.2"),
00243             MAVAddress("3.3"),
00244         };
00245         auto addr = pool.addresses();
00246         std::sort(addr.begin(), addr.end());
00247         REQUIRE(vec == addr);
00248     }
00249     fake_clock::advance(1s); // 00:02:03
00250     {
00251         std::vector<MAVAddress> vec =
00252         {
00253             MAVAddress("3.3"),
00254         };
00255         auto addr = pool.addresses();
00256         std::sort(addr.begin(), addr.end());
00257         REQUIRE(vec == addr);
00258     }
00259     fake_clock::advance(1s); // 00:02:04
00260     {
00261         std::vector<MAVAddress> vec;
00262         REQUIRE(pool.addresses().empty());
00263     }
00264 }
00265 SECTION("When using the 'addresses' method (and a custom timeout).")
00266 {
00267     AddressPool<fake_clock> pool(1h);
00268     pool.add(MAVAddress("0.0"));
00269     fake_clock::advance(1s); // 00:00:01
00270     pool.add(MAVAddress("1.1"));
00271     fake_clock::advance(1s); // 00:00:02
00272     pool.add(MAVAddress("2.2"));
00273     fake_clock::advance(1s); // 00:00:03
00274     pool.add(MAVAddress("3.3"));
00275     {
00276         std::vector<MAVAddress> vec =
00277         {
00278             MAVAddress("0.0"),
00279             MAVAddress("1.1"),
00280             MAVAddress("2.2"),
00281             MAVAddress("3.3"),
00282         };
00283         auto addr = pool.addresses();
00284         std::sort(addr.begin(), addr.end());
00285         REQUIRE(vec == addr);
00286     }
00287     fake_clock::advance(3597s); // 01:00:00
00288     {
00289         std::vector<MAVAddress> vec =
00290         {
00291             MAVAddress("0.0"),
00292             MAVAddress("1.1"),
00293             MAVAddress("2.2"),
00294             MAVAddress("3.3"),
00295         };
00296         auto addr = pool.addresses();
00297         std::sort(addr.begin(), addr.end());
00298         REQUIRE(vec == addr);
00299     }
00300     fake_clock::advance(1s); // 01:00:01
00301     {
00302         std::vector<MAVAddress> vec =
00303         {
00304             MAVAddress("1.1"),
00305             MAVAddress("2.2"),
00306             MAVAddress("3.3"),
00307         };
00308         auto addr = pool.addresses();
00309         std::sort(addr.begin(), addr.end());
00310         REQUIRE(vec == addr);
00311     }
00312     fake_clock::advance(1s); // 01:00:02
00313     {
00314         std::vector<MAVAddress> vec =
00315         {
00316             MAVAddress("2.2"),
00317             MAVAddress("3.3"),
00318         };
00319         auto addr = pool.addresses();
00320         std::sort(addr.begin(), addr.end());
```

```

00321     REQUIRE(vec == addr);
00322 }
00323 fake_clock::advance(1s); // 01:00:03
00324 {
00325     std::vector<MAVAddress> vec =
00326     {
00327         MAVAddress("3.3"),
00328     };
00329     auto addr = pool.addresses();
00330     std::sort(addr.begin(), addr.end());
00331     REQUIRE(vec == addr);
00332 }
00333 fake_clock::advance(1s); // 01:00:04
00334 {
00335     std::vector<MAVAddress> vec;
00336     REQUIRE(pool.addresses().empty());
00337 }
00338 }
00339 }
```

16.195 test_Call.cpp File Reference

```
#include <catch.hpp>
#include <fakeit.hpp>
#include "Action.hpp"
#include "Call.hpp"
#include "Chain.hpp"
#include "If.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "PacketVersion1.hpp"
#include "PacketVersion2.hpp"
#include "Rule.hpp"
#include "utility.hpp"
#include "common.hpp"
#include "common_Packet.hpp"
#include "common_Rule.hpp"
Include dependency graph for test_Call.cpp:
```



Functions

- [TEST_CASE \("Call's are constructable.", "\[Call\]"\)](#)
- [TEST_CASE \("Call's are comparable.", "\[Call\]"\)](#)
- [TEST_CASE \("Call's 'action' method determines what to do with a " "packet/address combination.", "\[Call\]"\)](#)
- [TEST_CASE \("Call's are printable \(without a condition but with a priority\).", "\[Call\]"\)](#)
- [TEST_CASE \("Call's are printable \(with a condition but without a priority\).", "\[Call\]"\)](#)
- [Call call \(chain, -3, If\(\).type\("PING"\).from\("192.168/8"\).to\("172.16/4"\)\)](#)
- [TEST_CASE \("Call's 'clone' method returns a polymorphic copy.", "\[Call\]"\)](#)

Variables

- auto ping = packet_v2::Packet(to_vector(PingV2()))
- Call call (chain)
- Rule & rule = call

16.195.1 Function Documentation

16.195.1.1 call()

```
Call call (
    chain ,
    - 3,
    If().type("PING").from("192.168/8").to("172.16/4") )
```

16.195.1.2 TEST_CASE() [1/6]

```
TEST_CASE (
    "Call's are constructable." ,
    "" [Call] )
```

Definition at line 37 of file [test_Call.cpp](#).

References [mock_shared\(\)](#).

Here is the call graph for this function:



16.195.1.3 TEST_CASE() [2/6]

```
TEST_CASE (
    "Call's are comparable." ,
    "" [Call] )
```

Definition at line 80 of file [test_Call.cpp](#).

References [mock_shared\(\)](#).

Here is the call graph for this function:



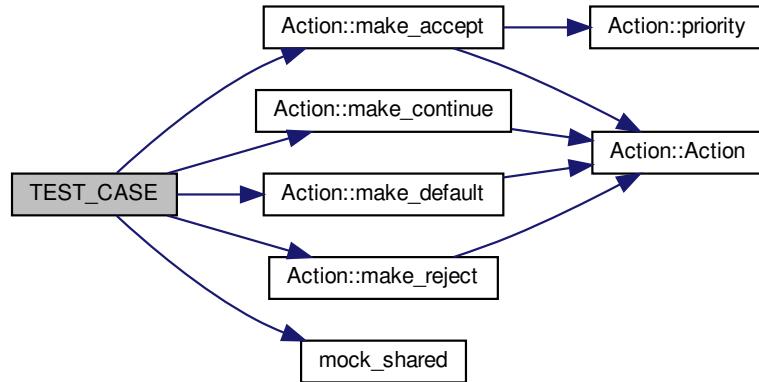
16.195.1.4 TEST_CASE() [3/6]

```
TEST_CASE (
    "Call's 'action' method determines what to do with a \"packet/address combination."
    ,
    "" [Call] )
```

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

References [Action::make_accept\(\)](#), [Action::make_continue\(\)](#), [Action::make_default\(\)](#), [Action::make_reject\(\)](#), [mock_shared\(\)](#), and [ping](#).

Here is the call graph for this function:



16.195.1.5 TEST_CASE() [4/6]

```
TEST_CASE (
    "Call's are printable (without a condition but with a priority).",
    "" [Call] )
```

Definition at line 243 of file [test_Call.cpp](#).

References [call](#), [ping](#), and [rule](#).

16.195.1.6 TEST_CASE() [5/6]

```
TEST_CASE (
    "Call's are printable (with a condition but without a priority).",
    "" [Call] )
```

Definition at line 261 of file [test_Call.cpp](#).

References [call](#), [ping](#), and [rule](#).

16.195.1.7 TEST_CASE() [6/6]

```
TEST_CASE (
    "Call's 'clone' method returns a polymorphic copy.",
    "" [Call] )
```

Definition at line 304 of file [test_Call.cpp](#).

16.195.2 Variable Documentation

16.195.2.1 call

```
Call call(chain, -3, If().type("PING").from("192.168/8").to("172.16/4")) (
    chain )
```

16.195.2.2 ping

```
auto ping = packet_v2::Packet(to_vector(PingV2()))
```

Definition at line 229 of file [test_Call.cpp](#).

16.195.2.3 rule

```
Rule & rule = call
```

Definition at line 231 of file [test_Call.cpp](#).

16.196 test_Call.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <catch.hpp>
00019 #include <fakeit.hpp>
00020
00021 #include "Action.hpp"
00022 #include "Call.hpp"
00023 #include "Chain.hpp"
00024 #include "If.hpp"
00025 #include "MAVAddress.hpp"
00026 #include "Packet.hpp"
00027 #include "PacketVersion1.hpp"
00028 #include "PacketVersion2.hpp"
00029 #include "Rule.hpp"
00030 #include "utility.hpp"
00031
00032 #include "common.hpp"
00033 #include "common_Packet.hpp"
00034 #include "common_Rule.hpp"
00035
00036
00037 TEST_CASE("Call's are constructable.", "[Call]")
00038 {
00039     fakeit::Mock<Chain> mock;
00040     std::shared_ptr<Chain> chain = mock_shared(mock);
00041     SECTION("Without a condition (match all packet/address combinations) or a "
00042             "priority.")
00043     {
00044         REQUIRE_NO_THROW(Call(chain));
00045     }
00046     SECTION("Without a condition (match all packet/address combinations) but "
00047             "with a priority.")
00048     {
00049         REQUIRE_NO_THROW(Call(chain, 3));
00050     }
```

```

00051     SECTION("With a condition and without a priority.")
00052     {
00053         REQUIRE_NO_THROW(Call(chain, If()));
00054         REQUIRE_NO_THROW(Call(chain, If().type("PING")));
00055         REQUIRE_NO_THROW(Call(chain, If().from("192.168")));
00056         REQUIRE_NO_THROW(Call(chain, If().to("172.16")));
00057     }
00058     SECTION("With both a condition and a priority.")
00059     {
00060         REQUIRE_NO_THROW(Call(chain, 3, If()));
00061         REQUIRE_NO_THROW(Call(chain, 3, If().type("PING")));
00062         REQUIRE_NO_THROW(Call(chain, 3, If().from("192.168")));
00063         REQUIRE_NO_THROW(Call(chain, 3, If().to("172.16")));
00064     }
00065     SECTION("Ensures the chain's shared pointer is not null.")
00066     {
00067         REQUIRE_THROWS_AS(Call(nullptr), std::invalid_argument);
00068         REQUIRE_THROWS_AS(Call(nullptr, 3), std::invalid_argument);
00069         REQUIRE_THROWS_AS(
00070             Call(nullptr, 3, If().type("PING")), std::invalid_argument);
00071         REQUIRE_THROWS_WITH(Call(nullptr, "Given chain pointer is null."));
00072         REQUIRE_THROWS_WITH(Call(nullptr, 3), "Given chain pointer is null.");
00073         REQUIRE_THROWS_WITH(
00074             Call(nullptr, 3, If().type("PING")),
00075             "Given chain pointer is null.");
00076     }
00077 }
00078
00079
00080 TEST_CASE("Call's are comparable.", "[Call]")
00081 {
00082     fakeit::Mock<Chain> mock1;
00083     fakeit::Mock<Chain> mock2;
00084     std::shared_ptr<Chain> chain1 = mock_shared(mock1);
00085     std::shared_ptr<Chain> chain2 = mock_shared(mock2);
00086     SECTION("with ==")
00087     {
00088         REQUIRE(Call(chain1) == Call(chain1));
00089         REQUIRE(
00090             Call(chain1, If().type("PING")) == Call(chain1, If().type("PING")));
00091         REQUIRE(Call(chain1, 3) == Call(chain1, 3));
00092         REQUIRE(
00093             Call(chain1, 3, If().type("PING")) ==
00094             Call(chain1, 3, If().type("PING")));
00095         REQUIRE_FALSE(Call(chain1) == Call(chain2));
00096         REQUIRE_FALSE(
00097             Call(chain1, If().type("PING")) ==
00098             Call(chain1, If().type("SET_MODE")));
00099         REQUIRE_FALSE(Call(chain1, If().type("PING")) == Call(chain1,
If()));
00100         REQUIRE_FALSE(Call(chain1, If().type("PING")) == Call(chain1));
00101         REQUIRE_FALSE(Call(chain1, 3) == Call(chain1, -3));
00102         REQUIRE_FALSE(Call(chain1, 3) == Call(chain1));
00103     }
00104     SECTION("with !=")
00105     {
00106         REQUIRE(Call(chain1, If().type("PING")) != Call(chain1));
00107         REQUIRE(Call(chain1, If().type("PING")) != Call(chain1, If()));
00108         REQUIRE(
00109             Call(chain1, If().type("PING")) !=
00110             Call(chain1, If().type("SET_MODE")));
00111         REQUIRE(Call(chain1, 3) != Call(chain1, -3));
00112         REQUIRE(Call(chain1, 3) != Call(chain1));
00113         REQUIRE(Call(chain1) != Call(chain2));
00114         REQUIRE_FALSE(Call(chain1) != Call(chain1));
00115         REQUIRE_FALSE(
00116             Call(chain1, If().type("PING")) != Call(chain1, If().type("PING")));
00117         REQUIRE_FALSE(Call(chain1, 3) != Call(chain1, 3));
00118         REQUIRE_FALSE(
00119             Call(chain1, 3, If().type("PING")) !=
00120             Call(chain1, 3, If().type("PING")));
00121     }
00122 }
00123
00124
00125 TEST_CASE("Call's 'action' method determines what to do with a "
00126             "packet/address combination.", "[Call]")
00127 {
00128     fakeit::Mock<Chain> accept_mock;
00129     fakeit::When(Method(accept_mock, action)).AlwaysReturn(
00130         Action::make_accept());

```

```

00131     std::shared_ptr<Chain> accept_chain = mock_shared(accept_mock);
00132     fakeit::Mock<Chain> reject_mock;
00133     fakeit::When(Method(reject_mock, action)).AlwaysReturn(
00134         Action::make_reject());
00135     std::shared_ptr<Chain> reject_chain = mock_shared(reject_mock);
00136     fakeit::Mock<Chain> continue_mock;
00137     fakeit::When(Method(continue_mock, action)).AlwaysReturn(
00138         Action::make_continue());
00139     std::shared_ptr<Chain> continue_chain = mock_shared(continue_mock);
00140     fakeit::Mock<Chain> default_mock;
00141     fakeit::When(Method(default_mock, action)).AlwaysReturn(
00142         Action::make_default());
00143     std::shared_ptr<Chain> default_chain = mock_shared(default_mock);
00144     fakeit::Mock<Chain> accept10_mock;
00145     fakeit::When(Method(accept10_mock, action)).AlwaysReturn(
00146         Action::make_accept(10));
00147     std::shared_ptr<Chain> accept10_chain = mock_shared(accept10_mock);
00148     auto ping = packet_v2::Packet(to_vector(PingV2()));
00149     SECTION("Check call to chain's action method.")
00150     {
00151         REQUIRE(Call(std::make_shared<TestChain>()).action(
00152             ping, MAVAddress("192.168")) ==
00153             Action::make_accept());
00154             fakeit::Mock<Chain> mock;
00155             fakeit::When(Method(mock, action)).AlwaysReturn(Action::make_accept());
00156             std::shared_ptr<Chain> chain = mock_shared(mock);
00157             MAVAddress address("192.168");
00158             Call(chain).action(ping, address);
00159             fakeit::Verify(
00160                 Method(mock, action).Matching([&](auto & a, auto & b)
00161                 {
00162                     return a == ping && b == MAVAddress("192.168");
00163                 })).Once();
00164     }
00165     SECTION("Delegates to the contained chain if there is no conditional.")
00166     {
00167         // Without priority.
00168         REQUIRE(
00169             Call(accept_chain).action(ping, MAVAddress("192.168")) ==
00170             Action::make_accept());
00171         REQUIRE(
00172             Call(reject_chain).action(ping, MAVAddress("192.168")) ==
00173             Action::make_reject());
00174         REQUIRE(
00175             Call(continue_chain).action(ping, MAVAddress("192.168")) ==
00176             Action::make_continue());
00177         REQUIRE(
00178             Call(default_chain).action(ping, MAVAddress("192.168")) ==
00179             Action::make_default());
00180         // With priority (adds priority).
00181         REQUIRE(
00182             Call(accept_chain, 3).action(ping, MAVAddress("192.168")) ==
00183             Action::make_accept(3));
00184         // Priority already set (no override).
00185         REQUIRE(
00186             Call(accept10_chain, 3).action(ping, MAVAddress("192.168")) ==
00187             Action::make_accept(10));
00188     }
00189     SECTION("Delegates to the contained chain if the conditional is a match.")
00190     {
00191         // Without priority.
00192         REQUIRE(
00193             Call(accept_chain, If().to("192.168")).action(
00194                 ping, MAVAddress("192.168")) ==
00195                 Action::make_accept());
00196         REQUIRE(
00197             Call(reject_chain, If().to("192.168")).action(
00198                 ping, MAVAddress("192.168")) ==
00199                 Action::make_reject());
00200         REQUIRE(
00201             Call(continue_chain, If().to("192.168")).action(
00202                 ping, MAVAddress("192.168")) ==
00203                 Action::make_continue());
00204         REQUIRE(
00205             Call(default_chain, If().to("192.168")).action(
00206                 ping, MAVAddress("192.168")) ==
00207                 Action::make_default());
00208         // With priority (adds priority).
00209         REQUIRE(
00210             Call(accept_chain, 3, If().to("192.168")).action(
00211                 ping, MAVAddress("192.168")) ==
00212                 Action::make_accept(3));
00213         REQUIRE(
00214             Call(accept10_chain, 3, If().to("192.168")).action(
00215                 ping, MAVAddress("192.168")) ==
00216                 Action::make_accept(10));
00217     }

```

```

    Action::make_accept(3));
00207     // Priority already set (no override).
00208     REQUIRE(
00209         Call(accept10_chain, 3, If().to("192.168")).action(
00210             ping, MAVAddress("192.168")) ==
00211     Action::make_accept(10));
00212     SECTION("Returns the continue action if the conditional does not match.")
00213     {
00214         // Without priority.
00215         REQUIRE(
00216             Call(accept_chain, If().to("172.16")).action(
00217                 ping, MAVAddress("192.168")) ==
00218         Action::make_continue());
00219         // With priority.
00220         REQUIRE(
00221             Call(accept_chain, 3, If().to("172.16")).action(
00222                 ping, MAVAddress("192.168")) ==
00223     Action::make_continue());
00224     }
00225
00226 TEST_CASE("Call's are printable (without a condition or a priority).", "[Call]")
00227 {
00228     auto chain = std::make_shared<TestChain>();
00229     auto ping = packet_v2::Packet(to_vector(PingV2()));
00230     Call call(chain);
00231     Rule &rule = call;
00232     SECTION("By direct type.")
00233     {
00234         REQUIRE(str(call) == "call test_chain");
00235     }
00236     SECTION("By polymorphic type.")
00237     {
00238         REQUIRE(str(rule) == "call test_chain");
00239     }
00240 }
00241
00242
00243 TEST_CASE("Call's are printable (without a condition but with a priority).",
00244     "[Call]")
00245 {
00246     auto chain = std::make_shared<TestChain>();
00247     auto ping = packet_v2::Packet(to_vector(PingV2()));
00248     Call call(chain, -3);
00249     Rule &rule = call;
00250     SECTION("By direct type.")
00251     {
00252         REQUIRE(str(call) == "call test_chain with priority -3");
00253     }
00254     SECTION("By polymorphic type.")
00255     {
00256         REQUIRE(str(rule) == "call test_chain with priority -3");
00257     }
00258 }
00259
00260
00261 TEST_CASE("Call's are printable (with a condition but without a priority).",
00262     "[Call]")
00263 {
00264     auto chain = std::make_shared<TestChain>();
00265     auto ping = packet_v2::Packet(to_vector(PingV2()));
00266     Call call(chain, If().type("PING").from("192.168/8").to("172.16/4"));
00267     Rule &rule = call;
00268     SECTION("By direct type.")
00269     {
00270         REQUIRE(
00271             str(call) == "call test_chain if PING from 192.168/8 to 172.16/4");
00272     }
00273     SECTION("By polymorphic type.")
00274     {
00275         REQUIRE(
00276             str(rule) == "call test_chain if PING from 192.168/8 to 172.16/4");
00277     }
00278 }
00279
00280
00281 TEST_CASE("Call's are printable (with a condition and a priority).", "[Call]")
00282 {
00283     auto chain = std::make_shared<TestChain>();

```

```

00284     auto ping = packet_v2::Packet(to_vector(PingV2()));
00285     Call call(chain, -3, If().type("PING").from("192.168/8").to("172.16/4"));
00286     Rule &rule = call;
00287     SECTION("By direct type.")
00288     {
00289         REQUIRE(
00290             str(call) ==
00291             "call test_chain with priority -3 "
00292             "if PING from 192.168/8 to 172.16/4");
00293     }
00294     SECTION("By polymorphic type.")
00295     {
00296         REQUIRE(
00297             str(rule) ==
00298             "call test_chain with priority -3 "
00299             "if PING from 192.168/8 to 172.16/4");
00300     }
00301 }
00302
00303
00304 TEST_CASE("Call's 'clone' method returns a polymorphic copy.", "[Call]")
00305 {
00306     auto chain = std::make_shared<TestChain>();
00307     SECTION("Without a priority.")
00308     {
00309         Call call(chain, If().type("PING"));
00310         Rule &rule = call;
00311         std::unique_ptr<Rule> polymorphic_copy = rule.clone();
00312         REQUIRE(call == *polymorphic_copy);
00313     }
00314     SECTION("With a priority.")
00315     {
00316         Call call(chain, 3, If().type("PING"));
00317         Rule &rule = call;
00318         std::unique_ptr<Rule> polymorphic_copy = rule.clone();
00319         REQUIRE(call == *polymorphic_copy);
00320     }
00321 }

```

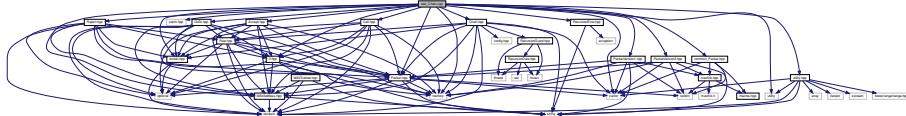
16.197 test_Chain.cpp File Reference

```

#include <memory>
#include <utility>
#include <vector>
#include <catch.hpp>
#include "Accept.hpp"
#include "Action.hpp"
#include "Call.hpp"
#include "Chain.hpp"
#include "GoTo.hpp"
#include "If.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "PacketVersion1.hpp"
#include "PacketVersion2.hpp"
#include "RecursionError.hpp"
#include "Reject.hpp"
#include "utility.hpp"
#include "common_Packet.hpp"

```

Include dependency graph for test_Chain.cpp:



Functions

- `TEST_CASE ("Chain's are constructable.", "[Chain]")`
- `TEST_CASE ("Chain's are comparable.", "[Chain]")`
- `TEST_CASE ("Chain's 'append' method appends a new rule to the filter chain.", "[Chain]")`
- `TEST_CASE ("Chain's 'name' method returns the name of the chain.", "[Chain]")`
- `TEST_CASE ("Chain's are copyable.", "[Chain]")`
- `TEST_CASE ("Chain's are movable.", "[Chain]")`
- `TEST_CASE ("Chain's are assignable.", "[Chain]")`
- `chain_a.append (std::make_unique<Accept>(If().to("192.168")))`
- `chain_a.append (std::make_unique<Reject>())`
- `TEST_CASE ("Chain's 'action' method determines what to do with a packet with \" \" respect to a destination address.", "[Rule]")`
- `TEST_CASE ("Chain's are printable.", "[Chain]")`

Variables

- `Chain chain_a_compare ("test_chain_a")`
- `Chain chain_b ("test_chain_b")`
- `Chain chain_b_compare ("test_chain_b")`
- `chain_a = std::move(chain_b)`

16.197.1 Function Documentation

16.197.1.1 append() [1/2]

```
chain_a_compare.append (
    std::make_unique<Accept>(If().to("192.168")) )
```

16.197.1.2 append() [2/2]

```
chain_b_compare.append (
    std::make_unique<Reject>() )
```

16.197.1.3 TEST_CASE() [1/9]

```
TEST_CASE (
    "Chain's are constructable." ,
    "" [Chain] )
```

Definition at line 41 of file [test_Chain.cpp](#).

16.197.1.4 TEST_CASE() [2/9]

```
TEST_CASE (
    "Chain's are comparable." ,
    "" [Chain] )
```

Definition at line 72 of file [test_Chain.cpp](#).

16.197.1.5 TEST_CASE() [3/9]

```
TEST_CASE (
    "Chain's 'append' method appends a new rule to the filter chain." ,
    "" [Chain] )
```

Definition at line 122 of file [test_Chain.cpp](#).

References [Chain::append\(\)](#).

Here is the call graph for this function:



16.197.1.6 TEST_CASE() [4/9]

```
TEST_CASE (
    "Chain's 'name' method returns the name of the chain." ,
    "" [Chain] )
```

Definition at line 136 of file [test_Chain.cpp](#).

16.197.1.7 TEST_CASE() [5/9]

```
TEST_CASE (
    "Chain's are copyable." ,
    "" [Chain] )
```

Definition at line 142 of file [test_Chain.cpp](#).

References [Chain::append\(\)](#).

Here is the call graph for this function:

**16.197.1.8 TEST_CASE() [6/9]**

```
TEST_CASE (
    "Chain's are movable." ,
    "" [Chain] )
```

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

References [Chain::append\(\)](#).

Here is the call graph for this function:



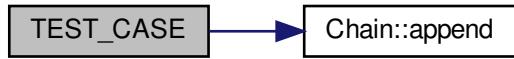
16.197.1.9 TEST_CASE() [7/9]

```
TEST_CASE (
    "Chain's are assignable." ,
    "" [Chain] )
```

Definition at line 168 of file [test_Chain.cpp](#).

References [Chain::append\(\)](#), [chain_a](#), [chain_a_compare](#), [chain_b](#), and [chain_b_compare](#).

Here is the call graph for this function:



16.197.1.10 TEST_CASE() [8/9]

```
TEST_CASE (
    "Chain's 'action' method determines what to do with a packet with \" \" respect to a
destination address." ,
    "" [Rule] )
```

Definition at line 204 of file [test_Chain.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.197.1.11 TEST_CASE() [9/9]

```
TEST_CASE (
    "Chain's are printable." ,
    "" [Chain] )
```

Definition at line 256 of file [test_Chain.cpp](#).

16.197.2 Variable Documentation**16.197.2.1 chain_a**

```
chain_a = std::move(chain\_b)
```

Definition at line 199 of file [test_Chain.cpp](#).

16.197.2.2 chain_a_compare

```
Chain chain_a_compare("test_chain_a")
```

16.197.2.3 chain_b

```
Chain chain_b("test_chain_b")
```

16.197.2.4 chain_b_compare

```
Chain chain_b_compare("test_chain_b")
```

16.198 test_Chain.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <memory>
00019 #include <utility>
00020 #include <vector>
00021
00022 #include <catch.hpp>
00023
00024 #include "Accept.hpp"
00025 #include "Action.hpp"
00026 #include "Call.hpp"
00027 #include "Chain.hpp"
00028 #include "GoTo.hpp"
00029 #include "If.hpp"
00030 #include "MAVAddress.hpp"
00031 #include "Packet.hpp"
00032 #include "PacketVersion1.hpp"
00033 #include "PacketVersion2.hpp"
00034 #include "RecursionError.hpp"
00035 #include "Reject.hpp"
00036 #include "utility.hpp"
00037
00038 #include "common_Packet.hpp"
00039
00040
00041 TEST_CASE("Chain's are constructable.", "[Chain]")
00042 {
00043     SECTION("Without rules.")
00044     {
00045         REQUIRE_NOTHROW(Chain("test_chain"));
00046     }
00047     SECTION("With rules.")
00048     {
00049         std::vector<std::unique_ptr<Rule>> rules;
00050         rules.push_back(std::make_unique<Accept>(If\(\).type("PING")));
00051         rules.push_back(std::make_unique<Reject>());
00052         REQUIRE_NOTHROW(Chain("test_chain", std::move(rules)));
00053     }
00054     SECTION("Ensures the chain name does not contain whitespace.")
00055     {
00056         // Spaces
00057         REQUIRE_THROWS_AS(Chain("test chain"), std::invalid_argument);
00058         REQUIRE_THROWS_WITH(
00059             Chain("test chain"), "Chain names cannot contain whitespace.");
00060         // Tabs
00061         REQUIRE_THROWS_AS(Chain("test\tchain"), std::invalid_argument);
00062         REQUIRE_THROWS_WITH(
00063             Chain("test\tchain"), "Chain names cannot contain whitespace.");
00064         // Newlines
00065         REQUIRE_THROWS_AS(Chain("test\nchain"), std::invalid_argument);
00066         REQUIRE_THROWS_WITH(
00067             Chain("test\nchain"), "Chain names cannot contain whitespace.");
00068     }
00069 }
00070
00071
00072 TEST_CASE("Chain's are comparable.", "[Chain]")
00073 {
00074     SECTION("When there are no rules.")
00075     {
00076         REQUIRE(Chain("test_chain") == Chain("test_chain"));
00077         REQUIRE_FALSE(Chain("test_chain") != Chain("test_chain"));
00078     }
00079 }
```

```

00078     REQUIRE_FALSE(Chain("test_chain1") == Chain("test_chain2"));
00079     REQUIRE(Chain("test_chain1") != Chain("test_chain2"));
00080 }
00081 SECTION("When the number of rules are not the same.")
00082 {
00083     std::vector<std::unique_ptr<Rule>> rules1;
00084     rules1.push_back(std::make_unique<Accept>(If().to("192.168")));
00085     rules1.push_back(std::make_unique<Reject>());
00086     Chain chain1("test_chain", std::move(rules1));
00087     std::vector<std::unique_ptr<Rule>> rules2;
00088     rules2.push_back(std::make_unique<Accept>(If().to("192.168")));
00089     Chain chain2("test_chain", std::move(rules2));
00090     REQUIRE_FALSE(chain1 == chain2);
00091     REQUIRE(chain1 != chain2);
00092 }
00093 SECTION("When the rules are the same.")
00094 {
00095     std::vector<std::unique_ptr<Rule>> rules1;
00096     rules1.push_back(std::make_unique<Accept>(If().to("192.168")));
00097     rules1.push_back(std::make_unique<Reject>());
00098     Chain chain1("test_chain", std::move(rules1));
00099     std::vector<std::unique_ptr<Rule>> rules2;
00100    rules2.push_back(std::make_unique<Accept>(If().to("192.168")));
00101    rules2.push_back(std::make_unique<Reject>());
00102    Chain chain2("test_chain", std::move(rules2));
00103    REQUIRE(chain1 == chain2);
00104    REQUIRE_FALSE(chain1 != chain2);
00105 }
00106 SECTION("When the rules are not the same.")
00107 {
00108     std::vector<std::unique_ptr<Rule>> rules1;
00109     rules1.push_back(std::make_unique<Accept>(If().to("192.168")));
00110     rules1.push_back(std::make_unique<Reject>());
00111     Chain chain1("test_chain", std::move(rules1));
00112     std::vector<std::unique_ptr<Rule>> rules2;
00113     rules2.push_back(std::make_unique<Accept>(If().from("172.16")));
00114     rules2.push_back(std::make_unique<Reject>());
00115     Chain chain2("test_chain", std::move(rules2));
00116     REQUIRE_FALSE(chain1 == chain2);
00117     REQUIRE(chain1 != chain2);
00118 }
00119 }
00120
00121
00122 TEST_CASE("Chain's 'append' method appends a new rule to the filter chain.", "[Chain]")
00123 {
00124     std::vector<std::unique_ptr<Rule>> rules;
00125     rules.push_back(std::make_unique<Accept>(If().to("192.168")));
00126     rules.push_back(std::make_unique<Reject>());
00127     Chain chain1("test_chain", std::move(rules));
00128     Chain chain2("test_chain");
00129     chain2.append(std::make_unique<Accept>(If().to("192.168")));
00130     chain2.append(std::make_unique<Reject>());
00131     REQUIRE(chain1 == chain2);
00132 }
00133
00134
00135
00136 TEST_CASE("Chain's 'name' method returns the name of the chain.", "[Chain]")
00137 {
00138     REQUIRE(Chain("crazy_chain_name").name() == "crazy_chain_name");
00139 }
00140
00141
00142 TEST_CASE("Chain's are copyable.", "[Chain]")
00143 {
00144     Chain original("test_chain");
00145     original.append(std::make_unique<Accept>(If().to("192.168")));
00146     original.append(std::make_unique<Reject>());
00147     Chain for_comparison("test_chain");
00148     for_comparison.append(std::make_unique<Accept>(If().to("192.168")));
00149     for_comparison.append(std::make_unique<Reject>());
00150     Chain copy(original);
00151     REQUIRE(copy == for_comparison);
00152 }
00153
00154
00155 TEST_CASE("Chain's are movable.", "[Chain]")
00156 {
00157     Chain original("test_chain");
00158     original.append(std::make_unique<Accept>(If().to("192.168")));

```

```

00159     original.append(std::make_unique<Reject>());
00160     Chain for_comparison("test_chain");
00161     for_comparison.append(std::make_unique<Accept>(If().to("192.168")));
00162     for_comparison.append(std::make_unique<Reject>());
00163     Chain moved(std::move(original));
00164     REQUIRE(moved == for_comparison);
00165 }
00166
00167
00168 TEST_CASE("Chain's are assignable.", "[Chain]")
00169 {
00170     Chain chain_a("test_chain_a");
00171     chain_a.append(std::make_unique<Accept>(If().to("192.168")));
00172     chain_a.append(std::make_unique<Reject>());
00173     Chain chain_a_compare("test_chain_a");
00174     chain_a_compare.append(std::make_unique<Accept>(If().to("192.168")));
00175     chain_a_compare.append(std::make_unique<Reject>());
00176     Chain chain_b("test_chain_b");
00177     chain_b.append(std::make_unique<Reject>());
00178     Chain chain_b_compare("test_chain_b");
00179     chain_b_compare.append(std::make_unique<Reject>());
00180     REQUIRE(chain_a == chain_a_compare);
00181     chain_a = chain_b;
00182     REQUIRE(chain_a == chain_b_compare);
00183 }
00184
00185
00186 TEST_CASE("Chain's are assignable (by move semantics).", "[Chain]")
00187 {
00188     Chain chain_a("test_chain_a");
00189     chain_a.append(std::make_unique<Accept>(If().to("192.168")));
00190     chain_a.append(std::make_unique<Reject>());
00191     Chain chain_a_compare("test_chain_a");
00192     chain_a_compare.append(std::make_unique<Accept>(If().to("192.168")));
00193     chain_a_compare.append(std::make_unique<Reject>());
00194     Chain chain_b("test_chain_b");
00195     chain_b.append(std::make_unique<Reject>());
00196     Chain chain_b_compare("test_chain_b");
00197     chain_b_compare.append(std::make_unique<Reject>());
00198     REQUIRE(chain_a == chain_a_compare);
00199     chain_a = std::move(chain_b);
00200     REQUIRE(chain_a == chain_b_compare);
00201 }
00202
00203
00204 TEST_CASE("Chain's 'action' method determines what to do with a packet with "
00205             "respect to a destination address.", "[Rule]")
00206 {
00207     auto heartbeat = packet_v2::Packet(to_vector(HeartbeatV2()));
00208     auto ping = packet_v1::Packet(to_vector(PingV1()));
00209     auto set_mode = packet_v2::Packet(to_vector(SetModeV2()));
00210     SECTION("When the chain is well formed.")
00211 {
00212     auto chain = std::make_shared<Chain>("main_chain");
00213     auto subchain = std::make_shared<Chain>("sub_chain");
00214     chain->append(std::make_unique<Accept>(If().to("192.168")));
00215     chain->append(std::make_unique<Accept>(If().type("HEARTBEAT")));
00216     chain->append(std::make_unique<Call>(subchain, If().to("172.0/8")));
00217     chain->append(std::make_unique<Reject>(If().to("10.10")));
00218     subchain->append(std::make_unique<Accept>(If().type("SET_MODE")));
00219     subchain->append(std::make_unique<Accept>(If().to("172.16")));
00220     REQUIRE(
00221         chain->action(ping, MAVAddress("192.168")) ==
00222             Action::make_accept());
00223     REQUIRE(
00224         chain->action(ping, MAVAddress("192.168")) ==
00225             Action::make_accept());
00226     REQUIRE(
00227         chain->action(heartbeat, MAVAddress("192.0")) ==
00228             Action::make_accept());
00229     REQUIRE(
00230         chain->action(set_mode, MAVAddress("172.0")) ==
00231             Action::make_accept());
00232     REQUIRE(
00233         chain->action(ping, MAVAddress("172.16")) ==
00234             Action::make_accept());
00235     REQUIRE(
00236         chain->action(ping, MAVAddress("10.10")) ==
00237             Action::make_reject());
00238     REQUIRE(
00239         chain->action(ping, MAVAddress("172.0")) ==

```

```

00240         Action::make_continue());
00241     }
00242     SECTION("And throws an error when chain recursion is detected.")
00243     {
00244         auto chain = std::make_shared<Chain>("main_chain");
00245         auto subchain = std::make_shared<Chain>("sub_chain");
00246         chain->append(std::make_unique<Call>(subchain));
00247         subchain->append(std::make_unique<Call>(chain));
00248         REQUIRE_THROWS_AS(
00249             chain->action(ping, MAVAddress("192.168")),
00250             RecursionError);
00251         REQUIRE_THROWS_WITH(
00252             chain->action(ping, MAVAddress("192.168")), "Recursion detected.");
00253     }
00254
00255
00256 TEST_CASE("Chain's are printable.", "[Chain]")
00257 {
00258     auto chain = std::make_shared<Chain>("default");
00259     auto ap_in = std::make_shared<Chain>("ap-in");
00260     auto ap_out = std::make_shared<Chain>("ap-out");
00261     chain->append(
00262         std::make_unique<Reject>(If().type("HEARTBEAT").from("10.10")));
00263     chain->append(
00264         std::make_unique<Accept>(-3, If().type("GPS_STATUS").to("172.0/8")));
00265     chain->append(
00266         std::make_unique<Accept>(
00267             If().type("GLOBAL_POSITION_INT").to("172.0/8")));
00268     chain->append(
00269         std::make_unique<GoTo>(ap_in, 3, If().from("192.168")));
00270     chain->append(std::make_unique<Call>(ap_out, If().to("192.168")));
00271     chain->append(std::make_unique<Reject>());
00272     REQUIRE(
00273         str(*chain) ==
00274         "chain default {\n"
00275         "    reject if HEARTBEAT from 10.10;\n"
00276         "    accept with priority -3 if GPS_STATUS to 172.0/8;\n"
00277         "    accept if GLOBAL_POSITION_INT to 172.0/8;\n"
00278         "    goto ap-in with priority 3 if from 192.168;\n"
00279         "    call ap-out if to 192.168;\n"
00280         "    reject;\n"
00281         "}");
00282 }
00283
00284
00285 // Required for complete function coverage.
00286 TEST_CASE("Run dynamic destructors (Chain).", "[Chain]")
00287 {
00288     Chain *chain = nullptr;
00289     REQUIRE_NOTHROW(chain = new Chain("chain"));
00290     REQUIRE_NOTHROW(delete chain);
00291 }

```

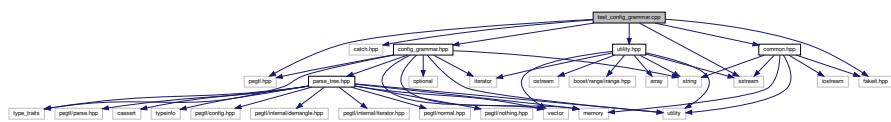
16.199 test_config_grammar.cpp File Reference

```

#include <iostream>
#include <catch.hpp>
#include <fakeit.hpp>
#include <pegtl.hpp>
#include "config_grammar.hpp"
#include "utility.hpp"
#include "common.hpp"

```

Include dependency graph for test_config_grammar.cpp:



Functions

- **TEST_CASE** ("A configuration string must have at least one valid statement " "or block.", "[config]")
 - **TEST_CASE** ("Global statements.", "[config]")
 - **TEST_CASE** ("Parse global 'default_action' statement.", "[config]")
 - **TEST_CASE** ("UDP configuration block.", "[config]")
 - **TEST_CASE** ("UDP port setting.", "[config]")
 - **TEST_CASE** ("UDP IP address setting.", "[config]")
 - **TEST_CASE** ("UDP max_bitrate setting.", "[config]")
 - **TEST_CASE** ("Serial port configuration block.", "[config]")
 - **TEST_CASE** ("Serial port device setting.", "[config]")
 - **TEST_CASE** ("Serial port baud rate setting.", "[config]")
 - **TEST_CASE** ("Serial port flow control setting.", "[config]")
 - **TEST_CASE** ("Serial address preload.", "[config]")
 - **TEST_CASE** ("Chain block.", "[config]")
 - **TEST_CASE** ("Invalid rule.", "[config]")
 - **TEST_CASE** ("Accept rule.", "[config]")
 - **TEST_CASE** ("Reject rule.", "[config]")
 - **TEST_CASE** ("Call rule.", "[config]")
 - **TEST_CASE** ("GoTo rule.", "[config]")
 - **TEST_CASE** ("Condition.", "[config]")
 - **TEST_CASE** ("Packet type condition.", "[config]")
 - **TEST_CASE** ("Source condition.", "[config]")
 - **TEST_CASE** ("Destination condition.", "[config]")
 - **TEST_CASE** ("Priority.", "[config]")
 - **TEST_CASE** ("Rule combinations with 'accept'.", "[config]")
 - **TEST_CASE** ("Rule combinations with 'reject'.", "[config]")
 - **TEST_CASE** ("Rule combinations with 'call'.", "[config]")
 - **TEST_CASE** ("Rule combinations with 'goto'.", "[config]")
 - **TEST_CASE** ("print_node' method prints an abstract syntax tree.", "[config]")

16.199.1 Function Documentation

16.199.1.1 TEST_CASE() [1/28]

```
TEST_CASE (
    "A configuration string must have at least one valid statement or block." ,
    "" [config] )
```

Definition at line 30 of file [test_config_grammar.cpp](#).

References [config::parse\(\)](#).

Here is the call graph for this function:

**16.199.1.2 TEST_CASE() [2/28]**

```
TEST_CASE (
    "Global statements." ,
    "" [config] )
```

Definition at line 41 of file [test_config_grammar.cpp](#).

16.199.1.3 TEST_CASE() [3/28]

```
TEST_CASE (
    "Parse global 'default_action' statement." ,
    "" [config] )
```

Definition at line 61 of file [test_config_grammar.cpp](#).

16.199.1.4 TEST_CASE() [4/28]

```
TEST_CASE (
    "UDP configuration block." ,
    "" [config] )
```

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

16.199.1.5 TEST_CASE() [5/28]

```
TEST_CASE (
    "UDP port setting." ,
    "" [config] )
```

Definition at line 197 of file [test_config_grammar.cpp](#).

16.199.1.6 TEST_CASE() [6/28]

```
TEST_CASE (
    "UDP IP address setting." ,
    "" [config] )
```

Definition at line 259 of file [test_config_grammar.cpp](#).

16.199.1.7 TEST_CASE() [7/28]

```
TEST_CASE (
    "UDP max_bitrate setting." ,
    "" [config] )
```

Definition at line 402 of file [test_config_grammar.cpp](#).

16.199.1.8 TEST_CASE() [8/28]

```
TEST_CASE (
    "Serial port configuration block." ,
    "" [config] )
```

Definition at line 464 of file [test_config_grammar.cpp](#).

16.199.1.9 TEST_CASE() [9/28]

```
TEST_CASE (
    "Serial port device setting." ,
    "" [config] )
```

Definition at line 534 of file [test_config_grammar.cpp](#).

16.199.1.10 TEST_CASE() [10/28]

```
TEST_CASE (
    "Serial port baud rate setting." ,
    "" [config] )
```

Definition at line 624 of file [test_config_grammar.cpp](#).

16.199.1.11 TEST_CASE() [11/28]

```
TEST_CASE (
    "Serial port flow control setting." ,
    "" [config] )
```

Definition at line 688 of file [test_config_grammar.cpp](#).

16.199.1.12 TEST_CASE() [12/28]

```
TEST_CASE (
    "Serial address preload." ,
    "" [config] )
```

Definition at line 778 of file [test_config_grammar.cpp](#).

16.199.1.13 TEST_CASE() [13/28]

```
TEST_CASE (
    "Chain block." ,
    "" [config] )
```

Definition at line 857 of file [test_config_grammar.cpp](#).

16.199.1.14 TEST_CASE() [14/28]

```
TEST_CASE (
    "Invalid rule." ,
    "" [config] )
```

Definition at line 941 of file [test_config_grammar.cpp](#).

16.199.1.15 TEST_CASE() [15/28]

```
TEST_CASE (
    "Accept rule." ,
    "" [config] )
```

Definition at line 957 of file [test_config_grammar.cpp](#).

16.199.1.16 TEST_CASE() [16/28]

```
TEST_CASE (
    "Reject rule." ,
    "" [config] )
```

Definition at line 999 of file [test_config_grammar.cpp](#).

16.199.1.17 TEST_CASE() [17/28]

```
TEST_CASE (
    "Call rule." ,
    "" [config] )
```

Definition at line 1041 of file [test_config_grammar.cpp](#).

16.199.1.18 TEST_CASE() [18/28]

```
TEST_CASE (
    "GoTo rule." ,
    "" [config] )
```

Definition at line 1105 of file [test_config_grammar.cpp](#).

16.199.1.19 TEST_CASE() [19/28]

```
TEST_CASE (
    "Condition." ,
    "" [config] )
```

Definition at line 1169 of file [test_config_grammar.cpp](#).

16.199.1.20 TEST_CASE() [20/28]

```
TEST_CASE (
    "Packet type condition." ,
    "" [config] )
```

Definition at line 1185 of file [test_config_grammar.cpp](#).

16.199.1.21 TEST_CASE() [21/28]

```
TEST_CASE (
    "Source condition." ,
    "" [config] )
```

Definition at line 1257 of file [test_config_grammar.cpp](#).

16.199.1.22 TEST_CASE() [22/28]

```
TEST_CASE (
    "Destination condition." ,
    "" [config] )
```

Definition at line 1426 of file [test_config_grammar.cpp](#).

16.199.1.23 TEST_CASE() [23/28]

```
TEST_CASE (
    "Priority." ,
    "" [config] )
```

Definition at line 1595 of file [test_config_grammar.cpp](#).

16.199.1.24 TEST_CASE() [24/28]

```
TEST_CASE (
    "Rule combinations with 'accept'." ,
    "" [config] )
```

Definition at line 1718 of file [test_config_grammar.cpp](#).

16.199.1.25 TEST_CASE() [25/28]

```
TEST_CASE (
    "Rule combinations with 'reject'." ,
    "" [config] )
```

Definition at line 1979 of file [test_config_grammar.cpp](#).

16.199.1.26 TEST_CASE() [26/28]

```
TEST_CASE (
    "Rule combinations with 'call'." ,
    "" [config] )
```

Definition at line 2240 of file [test_config_grammar.cpp](#).

16.199.1.27 TEST_CASE() [27/28]

```
TEST_CASE (
    "Rule combinations with 'goto'." ,
    "" [config] )
```

Definition at line 2502 of file [test_config_grammar.cpp](#).

16.199.1.28 TEST_CASE() [28/28]

```
TEST_CASE (
    "'print_node' method prints an abstract syntax tree." ,
    "" [config] )
```

Definition at line 2764 of file [test_config_grammar.cpp](#).

16.200 test_config_grammar.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <sstream>
00019
00020 #include <catch.hpp>
00021 #include <fakeit.hpp>
00022 #include <pegtl.hpp>
00023
00024 #include "config_grammar.hpp"
00025 #include "utility.hpp"
00026
00027 #include "common.hpp"
00028
00029
00030 TEST_CASE("A configuration string must have at least one valid statement "
00031           "or block.", "[config]")
00032 {
00033     tao::pegtl::string_input<> in("1337", "");
00034     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00035     REQUIRE_THROWS_WITH(
00036         config::parse(in),
00037         ":1:0(0): expected at least one valid statement or block");
00038 }
00039
00040
00041 TEST_CASE("Global statements.", "[config]")
00042 {
00043     SECTION("Missing end of statement ';;'.")
00044     {
00045         tao::pegtl::string_input<> in("invalid", "");
00046         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00047         REQUIRE_THROWS_WITH(
00048             config::parse(in),
00049             ":1:7(7): expected end of statement ';' character");
00050     }
00051     SECTION("Invalid statement.")
00052     {
00053         tao::pegtl::string_input<> in("invalid;", "");
00054         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00055         REQUIRE_THROWS_WITH(
00056             config::parse(in), ":1:8(8): unsupported statement");
00057     }
00058 }
00059
00060
00061 TEST_CASE("Parse global 'default_action' statement.", "[config]")
00062 {
00063     SECTION("Parses the 'accept' option.")
00064     {
00065         tao::pegtl::string_input<> in("default_action accept;", "");
00066         auto root = config::parse(in);
00067         REQUIRE(root != nullptr);
00068         REQUIRE(
00069             str(*root) ==
00070             ":001: default_action\n"
00071             ":001: | accept\n");
00072     }
00073     SECTION("Parses the 'accept' option (with comments).")
00074     {
00075         tao::pegtl::string_input<> in("default_action accept;# comment", "");
00076         auto root = config::parse(in);
00077         REQUIRE(root != nullptr);

```

```

00078     REQUIRE(
00079         str(*root) ==
00080             ":001: default_action\n"
00081             ":001: | accept\n");
00082     }
00083 SECTION("Parses the 'reject' option.")
00084 {
00085     tao::pegtl::string_input<> in("default_action reject;", "");
00086     auto root = config::parse(in);
00087     REQUIRE(root != nullptr);
00088     REQUIRE(
00089         str(*root) ==
00090             ":001: default_action\n"
00091             ":001: | reject\n");
00092     }
00093 SECTION("Parses the 'reject' option (with comments).")
00094 {
00095     tao::pegtl::string_input<> in("default_action reject;# comment", "");
00096     auto root = config::parse(in);
00097     REQUIRE(root != nullptr);
00098     REQUIRE(
00099         str(*root) ==
00100             ":001: default_action\n"
00101             ":001: | reject\n");
00102     }
00103 SECTION("Missing end of statement.")
00104 {
00105     tao::pegtl::string_input<> in("default_action accept", "");
00106     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00107     REQUIRE_THROWS_WITH(
00108         config::parse(in),
00109             ":1:21(21): expected end of statement ';' character");
00110     }
00111 SECTION("Invalid default action.")
00112 {
00113     tao::pegtl::string_input<> in("default_action invalid;", "");
00114     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00115     REQUIRE_THROWS_WITH(
00116         config::parse(in), ":1:15(15): expected 'accept' or 'reject'");
00117     }
00118 SECTION("Missing default action value.")
00119 {
00120     tao::pegtl::string_input<> in("default_action;", "");
00121     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00122     REQUIRE_THROWS_WITH(
00123         config::parse(in), ":1:14(14): expected 'accept' or 'reject'");
00124     }
00125 }
00126
00127
00128 TEST_CASE("UDP configuration block.", "[config]")
00129 {
00130     SECTION("Empty UDP blocks are allowed (single line).")
00131     {
00132         tao::pegtl::string_input<> in("udp {}", "");
00133         auto root = config::parse(in);
00134         REQUIRE(root != nullptr);
00135         REQUIRE(str(*root) == ":001: udp\n");
00136     }
00137 SECTION("Empty UDP blocks are allowed (single line with comments).")
00138 {
00139     tao::pegtl::string_input<> in("udp {}# comment", "");
00140     auto root = config::parse(in);
00141     REQUIRE(root != nullptr);
00142     REQUIRE(str(*root) == ":001: udp\n");
00143     }
00144 SECTION("Empty UDP blocks are allowed (1TBS style).")
00145 {
00146     tao::pegtl::string_input<> in("udp {\n}", "");
00147     auto root = config::parse(in);
00148     REQUIRE(root != nullptr);
00149     REQUIRE(str(*root) == ":001: udp\n");
00150     }
00151 SECTION("Empty UDP blocks are allowed (1TBS style with comments).")
00152 {
00153     tao::pegtl::string_input<> in("udp { # comment\n}# comment", "");
00154     auto root = config::parse(in);
00155     REQUIRE(root != nullptr);
00156     REQUIRE(str(*root) == ":001: udp\n");
00157     }
00158 SECTION("Empty UDP blocks are allowed (Allman style).")

```

```

00159     {
00160         tao::pegtl::string_input<> in("udp\n{\n", "");
00161         auto root = config::parse(in);
00162         REQUIRE(root != nullptr);
00163         REQUIRE(str(*root) == ":001: udp\n");
00164     }
00165 SECTION("Empty UDP blocks are allowed (Allman style with commentd).")
00166 {
00167     tao::pegtl::string_input<> in(
00168         "udp# comment \n{\# comment\n}# comment", "");
00169     auto root = config::parse(in);
00170     REQUIRE(root != nullptr);
00171     REQUIRE(str(*root) == ":001: udp\n");
00172 }
00173 SECTION("Missing opening brace (closing brace exists).")
00174 {
00175     tao::pegtl::string_input<> in("udp }", "");
00176     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00177     REQUIRE_THROWS_WITH(
00178         config::parse(in), ":1:4(4): expected opening brace '('");
00179 }
00180 SECTION("Missing opening brace (closing brace missing).")
00181 {
00182     tao::pegtl::string_input<> in("udp", "");
00183     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00184     REQUIRE_THROWS_WITH(
00185         config::parse(in), ":1:3(3): expected opening brace '('");
00186 }
00187 SECTION("Missing closing brace.")
00188 {
00189     tao::pegtl::string_input<> in("udp {", "");
00190     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00191     REQUIRE_THROWS_WITH(
00192         config::parse(in), ":1:5(5): expected closing brace ')'");
00193 }
00194 }
00195
00196
00197 TEST_CASE("UDP port setting.", "[config]")
00198 {
00199     SECTION("Parses port number setting.")
00200     {
00201         tao::pegtl::string_input<> in(
00202             "udp {\n"
00203                 "    port 14500;\n"
00204             "}", "");
00205         auto root = config::parse(in);
00206         REQUIRE(root != nullptr);
00207         REQUIRE(
00208             str(*root) ==
00209             ":001: udp\n"
00210             ":002: | port 14500\n");
00211     }
00212     SECTION("Parses port number setting (with comments).")
00213     {
00214         tao::pegtl::string_input<> in(
00215             "udp {# comment\n"
00216                 "    port 14500;# comment\n"
00217                 "# comment", "");
00218         auto root = config::parse(in);
00219         REQUIRE(root != nullptr);
00220         REQUIRE(
00221             str(*root) ==
00222             ":001: udp\n"
00223             ":002: | port 14500\n");
00224     }
00225     SECTION("Missing end of statement.")
00226     {
00227         tao::pegtl::string_input<> in(
00228             "udp {\n"
00229                 "    port 14500\n"
00230             "}", "");
00231         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00232         REQUIRE_THROWS_WITH(
00233             config::parse(in),
00234             ":3:0(21): expected end of statement ';' character");
00235     }
00236     SECTION("Invalid port number.")
00237     {
00238         tao::pegtl::string_input<> in(
00239             "udp {\n"

```

```
00240         "      port a35;\n"
00241         ")", "");
00242     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00243     REQUIRE_THROWS_WITH(
00244         config::parse(in), ":2:9(15): expected a valid port number");
00245 }
00246 SECTION("Missing port number.")
00247 {
00248     tao::pegtl::string_input<> in(
00249         "udp (\n"
00250         "      port;\n"
00251         ")", "");
00252     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00253     REQUIRE_THROWS_WITH(
00254         config::parse(in), ":2:8(14): expected a valid port number");
00255 }
00256 }
00257
00258
00259 TEST_CASE("UDP IP address setting.", "[config]")
00260 {
00261     SECTION("Parses IP address setting.")
00262     {
00263         tao::pegtl::string_input<> in(
00264             "udp (\n"
00265             "      address 127.0.0.1;\n"
00266             ")", "");
00267         auto root = config::parse(in);
00268         REQUIRE(root != nullptr);
00269         REQUIRE(
00270             str(*root) ==
00271             ":001: udp\n"
00272             ":002: | address 127.0.0.1\n");
00273     }
00274     SECTION("Parses IP address setting (with comments).")
00275     {
00276         tao::pegtl::string_input<> in(
00277             "udp (# comment\n"
00278             "      address 127.0.0.1;# comment\n"
00279             "# comment", "");
00280         auto root = config::parse(in);
00281         REQUIRE(root != nullptr);
00282         REQUIRE(
00283             str(*root) ==
00284             ":001: udp\n"
00285             ":002: | address 127.0.0.1\n");
00286     }
00287     SECTION("Missing end of statement.")
00288     {
00289         tao::pegtl::string_input<> in(
00290             "udp (\n"
00291             "      address 127.0.0.1\n"
00292             ")", "");
00293         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00294         REQUIRE_THROWS_WITH(
00295             config::parse(in),
00296             ":3:0(28): expected end of statement ';' character");
00297     }
00298     SECTION("Invalid IP address [1].")
00299     {
00300         tao::pegtl::string_input<> in(
00301             "udp (\n"
00302             "      address 127.0.0.1.;\n"
00303             ")", "");
00304         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00305         REQUIRE_THROWS_WITH(
00306             config::parse(in),
00307             ":2:21(27): expected end of statement ';' character");
00308     }
00309     SECTION("Invalid IP address [2].")
00310     {
00311         tao::pegtl::string_input<> in(
00312             "udp (\n"
00313             "      address 127.0.0.;\n"
00314             ")", "");
00315         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00316         REQUIRE_THROWS_WITH(
00317             config::parse(in), ":2:20(26): expected a valid IP address");
00318     }
00319     SECTION("Invalid IP address [3].")
00320     {
```

```

00321     tao::pegtl::string_input<> in(
00322         "udp (\n"
00323         "    address 127.0.0;\n"
00324         ")", "");
00325     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00326     REQUIRE_THROWS_WITH(
00327         config::parse(in), ":2:19(25): expected a valid IP address");
00328 }
00329 SECTION("Invalid IP address [4].")
00330 {
00331     tao::pegtl::string_input<> in(
00332         "udp (\n"
00333         "    address 127.0.1;\n"
00334         ")", "");
00335     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00336     REQUIRE_THROWS_WITH(
00337         config::parse(in), ":2:18(24): expected a valid IP address");
00338 }
00339 SECTION("Invalid IP address [5].")
00340 {
00341     tao::pegtl::string_input<> in(
00342         "udp (\n"
00343         "    address 127.0;\n"
00344         ")", "");
00345     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00346     REQUIRE_THROWS_WITH(
00347         config::parse(in), ":2:17(23): expected a valid IP address");
00348 }
00349 SECTION("Invalid IP address [6].")
00350 {
00351     tao::pegtl::string_input<> in(
00352         "udp (\n"
00353         "    address 127.1;\n"
00354         ")", "");
00355     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00356     REQUIRE_THROWS_WITH(
00357         config::parse(in), ":2:16(22): expected a valid IP address");
00358 }
00359 SECTION("Invalid IP address [7].")
00360 {
00361     tao::pegtl::string_input<> in(
00362         "udp (\n"
00363         "    address 127;\n"
00364         ")", "");
00365     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00366     REQUIRE_THROWS_WITH(
00367         config::parse(in), ":2:15(21): expected a valid IP address");
00368 }
00369 SECTION("Invalid IP address [8].")
00370 {
00371     tao::pegtl::string_input<> in(
00372         "udp (\n"
00373         "    address 127:0:0:1;\n"
00374         ")", "");
00375     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00376     REQUIRE_THROWS_WITH(
00377         config::parse(in), ":2:15(21): expected a valid IP address");
00378 }
00379 SECTION("Invalid IP address [9].")
00380 {
00381     tao::pegtl::string_input<> in(
00382         "udp (\n"
00383         "    address 127.a.0.1;\n"
00384         ")", "");
00385     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00386     REQUIRE_THROWS_WITH(
00387         config::parse(in), ":2:16(22): expected a valid IP address");
00388 }
00389 SECTION("Missing IP address.")
00390 {
00391     tao::pegtl::string_input<> in(
00392         "udp (\n"
00393         "    address;\n"
00394         ")", "");
00395     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00396     REQUIRE_THROWS_WITH(
00397         config::parse(in), ":2:11(17): expected a valid IP address");
00398 }
00399 }
00400
00401

```

```
00402 TEST_CASE("UDP max_bitrate setting.", "[config]")
00403 {
00404     SECTION("Parses max bitrate setting.")
00405     {
00406         tao:::pegtl:::string_input<> in(
00407             "udp {\n"
00408             "    max_bitrate 8192;\n"
00409             "}\n", "");
00410         auto root = config:::parse(in);
00411         REQUIRE(root != nullptr);
00412         REQUIRE(
00413             str(*root) ==
00414             ":001: udp\n"
00415             ":002: | max_bitrate 8192\n");
00416     }
00417     SECTION("Parses max bitrate setting (with comments).")
00418     {
00419         tao:::pegtl:::string_input<> in(
00420             "udp {# comment\n"
00421             "    max_bitrate 8192;# comment\n"
00422             "# comment", "");
00423         auto root = config:::parse(in);
00424         REQUIRE(root != nullptr);
00425         REQUIRE(
00426             str(*root) ==
00427             ":001: udp\n"
00428             ":002: | max_bitrate 8192\n");
00429     }
00430     SECTION("Missing end of statement.")
00431     {
00432         tao:::pegtl:::string_input<> in(
00433             "udp {\n"
00434             "    max_bitrate 8192\n"
00435             "}\n", "");
00436         REQUIRE_THROWS_AS(config:::parse(in), tao:::pegtl:::parse_error);
00437         REQUIRE_THROWS_WITH(
00438             config:::parse(in),
00439             ":3:0(27): expected end of statement ';' character");
00440     }
00441     SECTION("Invalid bitrate.")
00442     {
00443         tao:::pegtl:::string_input<> in(
00444             "udp {\n"
00445             "    max_bitrate a512;\n"
00446             "}\n", "");
00447         REQUIRE_THROWS_AS(config:::parse(in), tao:::pegtl:::parse_error);
00448         REQUIRE_THROWS_WITH(
00449             config:::parse(in), ":2:16(22): expected a valid bitrate");
00450     }
00451     SECTION("Missing bitrate.")
00452     {
00453         tao:::pegtl:::string_input<> in(
00454             "udp {\n"
00455             "    max_bitrate;\n"
00456             "}\n", "");
00457         REQUIRE_THROWS_AS(config:::parse(in), tao:::pegtl:::parse_error);
00458         REQUIRE_THROWS_WITH(
00459             config:::parse(in), ":2:15(21): expected a valid bitrate");
00460     }
00461 }
00462
00463
00464 TEST_CASE("Serial port configuration block.", "[config]")
00465 {
00466     SECTION("Empty serial port blocks are allowed (single line).")
00467     {
00468         tao:::pegtl:::string_input<> in("serial {}", "");
00469         auto root = config:::parse(in);
00470         REQUIRE(root != nullptr);
00471         REQUIRE(str(*root) == ":001: serial\n");
00472     }
00473     SECTION("Empty serial port blocks are allowed (single line with comments).")
00474     {
00475         tao:::pegtl:::string_input<> in("serial {}# comment", "");
00476         auto root = config:::parse(in);
00477         REQUIRE(root != nullptr);
00478         REQUIRE(str(*root) == ":001: serial\n");
00479     }
00480     SECTION("Empty serial port blocks are allowed (1TBS style).")
00481     {
00482         tao:::pegtl:::string_input<> in("serial {\n}", "");
```

```

00483     auto root = config::parse(in);
00484     REQUIRE(root != nullptr);
00485     REQUIRE(str(*root) == ":001: serial\n");
00486 }
00487 SECTION("Empty serial port blocks are allowed (1TBS style with comments).")
00488 {
00489     tao::pegtl::string_input<> in("serial {# comment\n}# comment", "");
00490     auto root = config::parse(in);
00491     REQUIRE(root != nullptr);
00492     REQUIRE(str(*root) == ":001: serial\n");
00493 }
00494 SECTION("Empty serial port blocks are allowed (Allman style).")
00495 {
00496     tao::pegtl::string_input<> in("serial\n{\n}", "");
00497     auto root = config::parse(in);
00498     REQUIRE(root != nullptr);
00499     REQUIRE(str(*root) == ":001: serial\n");
00500 }
00501 SECTION("Empty serial port blocks are allowed "
00502         "(Allman style with comments).")
00503 {
00504     tao::pegtl::string_input<> in(
00505         "serial# comment\n{# comment\n}# comment", "");
00506     auto root = config::parse(in);
00507     REQUIRE(root != nullptr);
00508     REQUIRE(str(*root) == ":001: serial\n");
00509 }
00510 SECTION("Missing opening brace (closing brace exists).")
00511 {
00512     tao::pegtl::string_input<> in("serial }", "");
00513     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00514     REQUIRE_THROWS_WITH(
00515         config::parse(in), ":1:7(7): expected opening brace '('");
00516 }
00517 SECTION("Missing opening brace (closing brace missing).")
00518 {
00519     tao::pegtl::string_input<> in("serial", "");
00520     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00521     REQUIRE_THROWS_WITH(
00522         config::parse(in), ":1:6(6): expected opening brace '('");
00523 }
00524 SECTION("Missing closing brace.")
00525 {
00526     tao::pegtl::string_input<> in("serial {", "");
00527     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00528     REQUIRE_THROWS_WITH(
00529         config::parse(in), ":1:8(8): expected closing brace ')'");
00530 }
00531 }
00532
00533
00534 TEST_CASE("Serial port device setting.", "[config]")
00535 {
00536     SECTION("Parses device string setting [1].")
00537 {
00538     tao::pegtl::string_input<> in(
00539         "serial {\n"
00540             "    device /dev/tty_USB0;\n"
00541         "}\n", "");
00542     auto root = config::parse(in);
00543     REQUIRE(root != nullptr);
00544     REQUIRE(
00545         str(*root) ==
00546         ":001: serial\n"
00547         ":002: | device /dev/tty_USB0\n");
00548 }
00549 SECTION("Parses device string setting (with comments) [1].")
00550 {
00551     tao::pegtl::string_input<> in(
00552         "serial {# comment\n"
00553             "    device /dev/tty_USB0;# comment\n"
00554             "# comment", "");
00555     auto root = config::parse(in);
00556     REQUIRE(root != nullptr);
00557     REQUIRE(
00558         str(*root) ==
00559         ":001: serial\n"
00560         ":002: | device /dev/tty_USB0\n");
00561 }
00562 SECTION("Parses device string setting [2].")
00563 {

```

```
00564     tao::pegtl::string_input<> in(
00565         "serial {\n"
00566             "    device COM1;\n"
00567             "} ", "");
00568     auto root = config::parse(in);
00569     REQUIRE(root != nullptr);
00570     REQUIRE(
00571         str(*root) ==
00572         ":001: serial\n"
00573         ":002: | device COM1\n");
00574 }
00575 SECTION("Parses device string setting (with comment) [2].")
00576 {
00577     tao::pegtl::string_input<> in(
00578         "serial {# comment\n"
00579             "    device COM1;# comment\n"
00580             "}# comment", "");
00581     auto root = config::parse(in);
00582     REQUIRE(root != nullptr);
00583     REQUIRE(
00584         str(*root) ==
00585         ":001: serial\n"
00586         ":002: | device COM1\n");
00587 }
00588 SECTION("Missing end of statement.")
00589 {
00590     tao::pegtl::string_input<> in(
00591         "serial {\n"
00592             "    device ttyUSB0\n"
00593             "} ", "");
00594     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00595     REQUIRE_THROWS_WITH(
00596         config::parse(in),
00597         ":3:0(28): expected end of statement ';' character");
00598 }
00599 SECTION("Invalid device string.")
00600 {
00601     tao::pegtl::string_input<> in(
00602         "serial {\n"
00603             "    device +ab;\n"
00604             "} ", "");
00605     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00606     REQUIRE_THROWS_WITH(
00607         config::parse(in),
00608         ":2:11(20): expected a valid serial port device name");
00609 }
00610 SECTION("Missing device string.")
00611 {
00612     tao::pegtl::string_input<> in(
00613         "serial {\n"
00614             "    device;\n"
00615             "} ", "");
00616     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00617     REQUIRE_THROWS_WITH(
00618         config::parse(in),
00619         ":2:10(19): expected a valid serial port device name");
00620 }
00621 }
00622
00623
00624 TEST_CASE("Serial port baud rate setting.", "[config]")
00625 {
00626     SECTION("Parses baud rate setting.")
00627 {
00628     tao::pegtl::string_input<> in(
00629         "serial {\n"
00630             "    baudrate 9600;\n"
00631             "} ", "");
00632     auto root = config::parse(in);
00633     REQUIRE(root != nullptr);
00634     REQUIRE(
00635         str(*root) ==
00636         ":001: serial\n"
00637         ":002: | baudrate 9600\n");
00638 }
00639 SECTION("Parses device string setting (with comments).")
00640 {
00641     tao::pegtl::string_input<> in(
00642         "serial {# comment\n"
00643             "    baudrate 9600;# comment\n"
00644             "}# comment", "");
```

```

00645     auto root = config::parse(in);
00646     REQUIRE(root != nullptr);
00647     REQUIRE(
00648         str(*root) ==
00649         ":001: serial\n"
00650         ":002: | baudrate 9600\n");
00651 }
00652 SECTION("Missing end of statement.")
00653 {
00654     tao::pegtl::string_input<> in(
00655         "serial\n"
00656         "    baudrate 9600\n"
00657         ")", "");
00658     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00659     REQUIRE_THROWS_WITH(
00660         config::parse(in),
00661         ":3:0(27): expected end of statement ';' character");
00662 }
00663 SECTION("Invalid baud rate.")
00664 {
00665     tao::pegtl::string_input<> in(
00666         "serial\n"
00667         "    baudrate +9600;\n"
00668         ")", "");
00669     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00670     REQUIRE_THROWS_WITH(
00671         config::parse(in),
00672         ":2:13(22): expected a valid baud rate");
00673 }
00674 SECTION("Missing baud rate value.")
00675 {
00676     tao::pegtl::string_input<> in(
00677         "serial\n"
00678         "    baudrate;\n"
00679         ")", "");
00680     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00681     REQUIRE_THROWS_WITH(
00682         config::parse(in),
00683         ":2:12(21): expected a valid baud rate");
00684 }
00685 }
00686
00687
00688 TEST_CASE("Serial port flow control setting.", "[config]")
00689 {
00690     SECTION("Parses flow control setting (yes).")
00691 {
00692     tao::pegtl::string_input<> in(
00693         "serial\n"
00694         "    flow_control yes;\n"
00695         ")", "");
00696     auto root = config::parse(in);
00697     REQUIRE(root != nullptr);
00698     REQUIRE(
00699         str(*root) ==
00700         ":001: serial\n"
00701         ":002: | flow_control yes\n");
00702 }
00703 SECTION("Parses flow control setting (yes with comments).")
00704 {
00705     tao::pegtl::string_input<> in(
00706         "serial # comments\n"
00707         "    flow_control yes;# comments\n"
00708         "# comments", "");
00709     auto root = config::parse(in);
00710     REQUIRE(root != nullptr);
00711     REQUIRE(
00712         str(*root) ==
00713         ":001: serial\n"
00714         ":002: | flow_control yes\n");
00715 }
00716 SECTION("Parses flow control setting (no).")
00717 {
00718     tao::pegtl::string_input<> in(
00719         "serial\n"
00720         "    flow_control no;\n"
00721         ")", "");
00722     auto root = config::parse(in);
00723     REQUIRE(root != nullptr);
00724     REQUIRE(
00725         str(*root) ==

```

```
00726      ":001: serial\n"
00727      ":002: | flow_control no\n");
00728  }
00729 SECTION("Parses flow control setting (no with comments).")
00730 {
00731     tao::pegtl::string_input<> in(
00732         "serial {# comments\n"
00733         "    flow_control no; # comments\n"
00734         "# comments", "");
00735     auto root = config::parse(in);
00736     REQUIRE(root != nullptr);
00737     REQUIRE(
00738         str(*root) ==
00739         ":001: serial\n"
00740         ":002: | flow_control no\n");
00741 }
00742 SECTION("Missing end of statement.")
00743 {
00744     tao::pegtl::string_input<> in(
00745         "serial {\n"
00746         "    flow_control yes\n"
00747         "}", "");
00748     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00749     REQUIRE_THROWS_WITH(
00750         config::parse(in),
00751         ":3:0(30): expected end of statement ';' character");
00752 }
00753 SECTION("Invalid flow control setting.")
00754 {
00755     tao::pegtl::string_input<> in(
00756         "serial {\n"
00757         "    flow_control maybe;\n"
00758         "}", "");
00759     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00760     REQUIRE_THROWS_WITH(
00761         config::parse(in),
00762         ":2:17(26): expected 'yes' or 'no'");
00763 }
00764 SECTION("Missing flow control setting.")
00765 {
00766     tao::pegtl::string_input<> in(
00767         "serial {\n"
00768         "    flow_control;\n"
00769         "}", "");
00770     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00771     REQUIRE_THROWS_WITH(
00772         config::parse(in),
00773         ":2:16(25): expected 'yes' or 'no'");
00774 }
00775 }
00776
00777
00778 TEST_CASE("Serial address preload.", "[config]")
00779 {
00780     SECTION("Parses a preload address.")
00781 {
00782     tao::pegtl::string_input<> in(
00783         "serial {\n"
00784         "    preload 62.34;\n"
00785         "}", "");
00786     auto root = config::parse(in);
00787     REQUIRE(root != nullptr);
00788     REQUIRE(
00789         str(*root) ==
00790         ":001: serial\n"
00791         ":002: | preload 62.34\n");
00792 }
00793 SECTION("Parses multiple preload address.")
00794 {
00795     tao::pegtl::string_input<> in(
00796         "serial {\n"
00797         "    preload 1.1;\n"
00798         "    preload 62.34;\n"
00799         "}", "");
00800     auto root = config::parse(in);
00801     REQUIRE(root != nullptr);
00802     REQUIRE(
00803         str(*root) ==
00804         ":001: serial\n"
00805         ":002: | preload 1.1\n"
00806         ":003: | preload 62.34\n");
```

```

00807     }
00808     SECTION("Parses a preload address (with comments).")
00809     {
00810         tao::pegtl::string_input<> in(
00811             "serial {# comment\n"
00812             "    preload 62.34;# comment\n"
00813             "#}# comment", "");
00814         auto root = config::parse(in);
00815         REQUIRE(root != nullptr);
00816         REQUIRE(
00817             str(*root) ==
00818             ":001: serial\n"
00819             ":002: | preload 62.34\n");
00820     }
00821     SECTION("Missing end of statement.")
00822     {
00823         tao::pegtl::string_input<> in(
00824             "serial {\n"
00825             "    preload 62.34\n"
00826             "}", "");
00827         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00828         REQUIRE_THROWS_WITH(
00829             config::parse(in),
00830             ":3:0(27): expected end of statement ';' character");
00831     }
00832     SECTION("Invalid MAVLink address.")
00833     {
00834         tao::pegtl::string_input<> in(
00835             "serial {\n"
00836             "    preload 62:34;\n"
00837             "}", "");
00838         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00839         REQUIRE_THROWS_WITH(
00840             config::parse(in),
00841             ":2:14(23): expected a valid MAVLink address");
00842     }
00843     SECTION("Missing MAVLink address.")
00844     {
00845         tao::pegtl::string_input<> in(
00846             "serial {\n"
00847             "    preload;\n"
00848             "}", "");
00849         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00850         REQUIRE_THROWS_WITH(
00851             config::parse(in),
00852             ":2:11(20): expected a valid MAVLink address");
00853     }
00854 }
00855
00856
00857 TEST_CASE("Chain block.", "[config]")
00858 {
00859     SECTION("Empty chain blocks are allowed (single line).")
00860     {
00861         tao::pegtl::string_input<> in("chain some_name10 {}", "");
00862         auto root = config::parse(in);
00863         REQUIRE(root != nullptr);
00864         REQUIRE(str(*root) == ":001: chain some_name10\n");
00865     }
00866     SECTION("Empty chain blocks are allowed (single line with comments).")
00867     {
00868         tao::pegtl::string_input<> in("chain some_name10 {}# comment", "");
00869         auto root = config::parse(in);
00870         REQUIRE(root != nullptr);
00871         REQUIRE(str(*root) == ":001: chain some_name10\n");
00872     }
00873     SECTION("Empty chain blocks are allowed (1TBS style).")
00874     {
00875         tao::pegtl::string_input<> in("chain some_name10 {\n}", "");
00876         auto root = config::parse(in);
00877         REQUIRE(root != nullptr);
00878         REQUIRE(str(*root) == ":001: chain some_name10\n");
00879     }
00880     SECTION("Empty chain blocks are allowed (1TBS style with comments).")
00881     {
00882         tao::pegtl::string_input<> in(
00883             "chain some_name10 (# comment\n{n}# comment", "");
00884         auto root = config::parse(in);
00885         REQUIRE(root != nullptr);
00886         REQUIRE(str(*root) == ":001: chain some_name10\n");
00887     }

```

```

00888     SECTION("Empty chain blocks are allowed (Allman style).")
00889     {
00890         tao::pegtl::string_input<> in("chain some_name10\n{\n}", "");
00891         auto root = config::parse(in);
00892         REQUIRE(root != nullptr);
00893         REQUIRE(str(*root) == ":001: chain some_name10\n");
00894     }
00895     SECTION("Empty chain blocks are allowed (Allman style with comments).")
00896     {
00897         tao::pegtl::string_input<> in(
00898             "chain some_name10# comment\n#{ comment\n}# comment", "");
00899         auto root = config::parse(in);
00900         REQUIRE(root != nullptr);
00901         REQUIRE(str(*root) == ":001: chain some_name10\n");
00902     }
00903     SECTION("Chains must have a name.")
00904     {
00905         tao::pegtl::string_input<> in("chain {}", "");
00906         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00907         REQUIRE_THROWS_WITH(
00908             config::parse(in), ":1:6(6): expected a valid chain name");
00909     }
00910     SECTION("Chain names must be valid.")
00911     {
00912         tao::pegtl::string_input<> in("chain lnv@lld {}", "");
00913         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00914         REQUIRE_THROWS_WITH(
00915             config::parse(in), ":1:6(6): expected a valid chain name");
00916     }
00917     SECTION("Missing opening brace (closing brace exists).")
00918     {
00919         tao::pegtl::string_input<> in("chain some_name10 )", "");
00920         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00921         REQUIRE_THROWS_WITH(
00922             config::parse(in), ":1:18(18): expected opening brace '('");
00923     }
00924     SECTION("Missing opening brace (closing brace missing).")
00925     {
00926         tao::pegtl::string_input<> in("chain some_name10", "");
00927         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00928         REQUIRE_THROWS_WITH(
00929             config::parse(in), ":1:17(17): expected opening brace '('");
00930     }
00931     SECTION("Missing closing brace.")
00932     {
00933         tao::pegtl::string_input<> in("chain some_name10 {", "");
00934         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00935         REQUIRE_THROWS_WITH(
00936             config::parse(in), ":1:19(19): expected closing brace ')'");
00937     }
00938 }
00939
00940
00941 TEST_CASE("Invalid rule.", "[config]")
00942 {
00943     SECTION("Parses 'reject' rule.")
00944     {
00945         tao::pegtl::string_input<> in(
00946             "chain default {\n"
00947             "    invalid;\n"
00948             "}", "");
00949         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00950         REQUIRE_THROWS_WITH(
00951             config::parse(in),
00952             ":3:0(29): expected a valid rule");
00953     }
00954 }
00955
00956
00957 TEST_CASE("Accept rule.", "[config]")
00958 {
00959     SECTION("Parses 'accept' rule.")
00960     {
00961         tao::pegtl::string_input<> in(
00962             "chain default {\n"
00963             "    accept;\n"
00964             "}", "");
00965         auto root = config::parse(in);
00966         REQUIRE(root != nullptr);
00967         REQUIRE(
00968             str(*root) ==

```

```

00969         ":001: chain default\n"
00970         ":002: | accept\n";
00971     }
00972 SECTION("Parses 'accept' rule (with comments).")
00973 {
00974     tao::pegtl::string_input<> in(
00975         "chain default /* comment\n"
00976         "    accept;# comment\n"
00977         "*/# comment", "");
00978     auto root = config::parse(in);
00979     REQUIRE(root != nullptr);
00980     REQUIRE(
00981         str(*root) ==
00982             ":001: chain default\n"
00983             ":002: | accept\n");
00984 }
00985 SECTION("Missing end of statement.")
00986 {
00987     tao::pegtl::string_input<> in(
00988         "chain default {\n"
00989         "    accept\n"
00990         "}", "");
00991     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
00992     REQUIRE_THROWS_WITH(
00993         config::parse(in),
00994         ":2:10(26): expected end of statement ';' character");
00995 }
00996 }
00997
00998
00999 TEST_CASE("Reject rule.", "[config]")
01000 {
01001     SECTION("Parses 'reject' rule.")
01002     {
01003         tao::pegtl::string_input<> in(
01004             "chain default {\n"
01005             "    reject;\n"
01006             "}", "");
01007         auto root = config::parse(in);
01008         REQUIRE(root != nullptr);
01009         REQUIRE(
01010             str(*root) ==
01011                 ":001: chain default\n"
01012                 ":002: | reject\n");
01013     }
01014     SECTION("Parses 'reject' rule (with comments).")
01015     {
01016         tao::pegtl::string_input<> in(
01017             "chain default /* comment\n"
01018             "    reject;# comment\n"
01019             "*/# comment", "");
01020         auto root = config::parse(in);
01021         REQUIRE(root != nullptr);
01022         REQUIRE(
01023             str(*root) ==
01024                 ":001: chain default\n"
01025                 ":002: | reject\n");
01026     }
01027     SECTION("Missing end of statement.")
01028     {
01029         tao::pegtl::string_input<> in(
01030             "chain default {\n"
01031             "    reject\n"
01032             "}", "");
01033         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01034         REQUIRE_THROWS_WITH(
01035             config::parse(in),
01036             ":2:10(26): expected end of statement ';' character");
01037     }
01038 }
01039
01040
01041 TEST_CASE("Call rule.", "[config]")
01042 {
01043     SECTION("Parses 'call' rule.")
01044     {
01045         tao::pegtl::string_input<> in(
01046             "chain default {\n"
01047             "    call some_name10;\n"
01048             "}", "");
01049         auto root = config::parse(in);

```

```

01050     REQUIRE(root != nullptr);
01051     REQUIRE(
01052         str(*root) ==
01053         ":001: chain default\n"
01054         ":002: | call some_name10\\n");
01055     }
01056 SECTION("Parses 'call' rule (with comments).")
01057 {
01058     tao::pegtl::string_input<> in(
01059         "chain default {# comment\\n"
01060         "    call some_name10;# comment\\n"
01061         "}# comment", "");
01062     auto root = config::parse(in);
01063     REQUIRE(root != nullptr);
01064     REQUIRE(
01065         str(*root) ==
01066         ":001: chain default\\n"
01067         ":002: | call some_name10\\n");
01068 }
01069 SECTION("Missing end of statement.")
01070 {
01071     tao::pegtl::string_input<> in(
01072         "chain default {\\n"
01073         "    call some_name10\\n"
01074         "}", "");
01075     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01076     REQUIRE_THROWS_WITH(
01077         config::parse(in),
01078         ":3:0(37): expected end of statement ';' character");
01079 }
01080 SECTION("Invalid chain name.")
01081 {
01082     tao::pegtl::string_input<> in(
01083         "chain default {\\n"
01084         "    call lnv@lld;\\n"
01085         "}", "");
01086     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01087     REQUIRE_THROWS_WITH(
01088         config::parse(in),
01089         ":2:9(25): expected a valid chain name");
01090 }
01091 SECTION("Missing chain name.")
01092 {
01093     tao::pegtl::string_input<> in(
01094         "chain default {\\n"
01095         "    call;\\n"
01096         "}", "");
01097     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01098     REQUIRE_THROWS_WITH(
01099         config::parse(in),
01100         ":2:8(24): expected a valid chain name");
01101 }
01102 }
01103
01104
01105 TEST_CASE("GoTo rule.", "[config]")
01106 {
01107     SECTION("Parses 'goto' rule.")
01108     {
01109         tao::pegtl::string_input<> in(
01110             "chain default {\\n"
01111             "    goto some_name10;\\n"
01112             "}", "");
01113         auto root = config::parse(in);
01114         REQUIRE(root != nullptr);
01115         REQUIRE(
01116             str(*root) ==
01117             ":001: chain default\\n"
01118             ":002: | goto some_name10\\n");
01119     }
01120     SECTION("Parses 'goto' rule (with comments).")
01121     {
01122         tao::pegtl::string_input<> in(
01123             "chain default {# comment\\n"
01124             "    goto some_name10;# comment\\n"
01125             "}# comment", "");
01126         auto root = config::parse(in);
01127         REQUIRE(root != nullptr);
01128         REQUIRE(
01129             str(*root) ==
01130             ":001: chain default\\n"

```

```

01131         ":002: | goto some_name10\n");
01132     }
01133 SECTION("Missing end of statement.")
01134 {
01135     tao::pegtl::string_input<> in(
01136         "chain default {\n"
01137         "    goto some_name10\n"
01138         "}\n", "");
01139     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01140     REQUIRE_THROWS_WITH(
01141         config::parse(in),
01142         ":3:0(37): expected end of statement ';' character");
01143 }
01144 SECTION("Invalid chain name.")
01145 {
01146     tao::pegtl::string_input<> in(
01147         "chain default {\n"
01148         "    goto Inv@lld;\n"
01149         "}\n", "");
01150     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01151     REQUIRE_THROWS_WITH(
01152         config::parse(in),
01153         ":2:9(25): expected a valid chain name");
01154 }
01155 SECTION("Missing chain name.")
01156 {
01157     tao::pegtl::string_input<> in(
01158         "chain default {\n"
01159         "    goto;\n"
01160         "}\n", "");
01161     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01162     REQUIRE_THROWS_WITH(
01163         config::parse(in),
01164         ":2:8(24): expected a valid chain name");
01165 }
01166 }
01167
01168
01169 TEST_CASE("Condition.", "[config]")
01170 {
01171     SECTION("Conditions must have at least one restriction.")
01172     {
01173         tao::pegtl::string_input<> in(
01174             "chain default {\n"
01175             "    accept if;\n"
01176             "}\n", "");
01177         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01178         REQUIRE_THROWS_WITH(
01179             config::parse(in),
01180             ":2:13(29): condition is empty or invalid");
01181     }
01182 }
01183
01184
01185 TEST_CASE("Packet type condition.", "[config]")
01186 {
01187     SECTION("Parses packet type condition.")
01188     {
01189         tao::pegtl::string_input<> in(
01190             "chain default {\n"
01191             "    accept if PING;\n"
01192             "}\n", "");
01193         auto root = config::parse(in);
01194         REQUIRE(root != nullptr);
01195         REQUIRE(
01196             str(*root) ==
01197             ":001: chain default\n"
01198             ":002: | accept\n"
01199             ":002: | | condition\n"
01200             ":002: | | | packet_type PING\n";
01201     }
01202     SECTION("Parses packet type condition (with number).")
01203     {
01204         tao::pegtl::string_input<> in(
01205             "chain default {\n"
01206             "    accept if SCALED_PRESSURE3;\n"
01207             "}\n", "");
01208         auto root = config::parse(in);
01209         REQUIRE(root != nullptr);
01210         REQUIRE(
01211             str(*root) ==

```

```

01212         ":001: chain default\n"
01213         ":002: | accept\n"
01214         ":002: | | condition\n"
01215         ":002: | | | packet_type SCALED_PRESSURE3\n";
01216     }
01217     SECTION("Parses packet type condition (with comments).")
01218     {
01219         tao::pegtl::string_input<> in(
01220             "chain default {# comment\n"
01221             "    accept if PING;# comment\n"
01222             "#}# comment", "");
01223         auto root = config::parse(in);
01224         REQUIRE(root != nullptr);
01225         REQUIRE(
01226             str(*root) ==
01227             ":001: chain default\n"
01228             ":002: | accept\n"
01229             ":002: | | condition\n"
01230             ":002: | | | packet_type PING\n";
01231         }
01232     SECTION("Invalid packet type [1].")
01233     {
01234         tao::pegtl::string_input<> in(
01235             "chain default {\n"
01236             "    accept if ping;\n"
01237             "}", "");
01238         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01239         REQUIRE_THROWS_WITH(
01240             config::parse(in),
01241             ":2:14(30): condition is empty or invalid");
01242     }
01243     SECTION("Invalid packet type [2].")
01244     {
01245         tao::pegtl::string_input<> in(
01246             "chain default {\n"
01247             "    accept if @;\n"
01248             "}", "");
01249         REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01250         REQUIRE_THROWS_WITH(
01251             config::parse(in),
01252             ":2:14(30): condition is empty or invalid");
01253     }
01254 }
01255
01256
01257 TEST_CASE("Source condition.", "[config]")
01258 {
01259     SECTION("Parses source condition [1].")
01260     {
01261         tao::pegtl::string_input<> in(
01262             "chain default {\n"
01263             "    accept if from 192.0;\n"
01264             "}", "");
01265         auto root = config::parse(in);
01266         REQUIRE(root != nullptr);
01267         REQUIRE(
01268             str(*root) ==
01269             ":001: chain default\n"
01270             ":002: | accept\n"
01271             ":002: | | condition\n"
01272             ":002: | | | source 192.0\n");
01273     }
01274     SECTION("Parses source condition (with comments) [1].")
01275     {
01276         tao::pegtl::string_input<> in(
01277             "chain default {# comment\n"
01278             "    accept if from 192.0;# comment\n"
01279             "#}# comment", "");
01280         auto root = config::parse(in);
01281         REQUIRE(root != nullptr);
01282         REQUIRE(
01283             str(*root) ==
01284             ":001: chain default\n"
01285             ":002: | accept\n"
01286             ":002: | | condition\n"
01287             ":002: | | | source 192.0\n");
01288     }
01289     SECTION("Parses source condition [2].")
01290     {
01291         tao::pegtl::string_input<> in(
01292             "chain default {\n"

```

```

01293         "      accept if from 192.0/8;\n"
01294         ")");
01295     auto root = config::parse(in);
01296     REQUIRE(root != nullptr);
01297     REQUIRE(
01298         str(*root) ==
01299         ":001: chain default\n"
01300         ":002: | accept\n"
01301         ":002: | | condition\n"
01302         ":002: | | | source 192.0/8\n");
01303     }
01304 SECTION("Parses source condition (with comments) [2].")
01305 {
01306     tao::pegtl::string_input<> in(
01307         "chain default {# comment\n"
01308         "    accept if from 192.0/8;# comment\n"
01309         "# comment", "");
01310     auto root = config::parse(in);
01311     REQUIRE(root != nullptr);
01312     REQUIRE(
01313         str(*root) ==
01314         ":001: chain default\n"
01315         ":002: | accept\n"
01316         ":002: | | condition\n"
01317         ":002: | | | source 192.0/8\n");
01318     }
01319 SECTION("Parses source condition [3].")
01320 {
01321     tao::pegtl::string_input<> in(
01322         "chain default (\n"
01323         "    accept if from 192.0\\4;\n"
01324         ") ", "");
01325     auto root = config::parse(in);
01326     REQUIRE(root != nullptr);
01327     REQUIRE(
01328         str(*root) ==
01329         ":001: chain default\n"
01330         ":002: | accept\n"
01331         ":002: | | condition\n"
01332         ":002: | | | source 192.0\\4\n");
01333     }
01334 SECTION("Parses source condition (with comments) [3].")
01335 {
01336     tao::pegtl::string_input<> in(
01337         "chain default {# comment\n"
01338         "    accept if from 192.0\\4;# comment\n"
01339         "# comment", "");
01340     auto root = config::parse(in);
01341     REQUIRE(root != nullptr);
01342     REQUIRE(
01343         str(*root) ==
01344         ":001: chain default\n"
01345         ":002: | accept\n"
01346         ":002: | | condition\n"
01347         ":002: | | | source 192.0\\4\n");
01348     }
01349 SECTION("Parses source condition [4].")
01350 {
01351     tao::pegtl::string_input<> in(
01352         "chain default (\n"
01353         "    accept if from 192.0:255.255;\n"
01354         ") ", "");
01355     auto root = config::parse(in);
01356     REQUIRE(root != nullptr);
01357     REQUIRE(
01358         str(*root) ==
01359         ":001: chain default\n"
01360         ":002: | accept\n"
01361         ":002: | | condition\n"
01362         ":002: | | | source 192.0:255.255\n");
01363     }
01364 SECTION("Parses source condition (with comments) [4].")
01365 {
01366     tao::pegtl::string_input<> in(
01367         "chain default {# comment\n"
01368         "    accept if from 192.0:255.255;# comment\n"
01369         "# comment", "");
01370     auto root = config::parse(in);
01371     REQUIRE(root != nullptr);
01372     REQUIRE(
01373         str(*root) ==

```

```
01374      ":001: chain default\n"
01375      ":002: | accept\n"
01376      ":002: | | condition\n"
01377      ":002: | | | source 192.0:255.255\n");
01378  }
01379 SECTION("Invalid source condition [1].")
01380 {
01381     tao::pegtl::string_input<> in(
01382         "chain default {\n"
01383         "    accept if from 192;\n"
01384         "}", "");
01385     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01386     REQUIRE_THROWS_WITH(
01387         config::parse(in),
01388         ":2:22(38): expected a valid MAVLink subnet");
01389 }
01390 SECTION("Invalid source condition [2].")
01391 {
01392     tao::pegtl::string_input<> in(
01393         "chain default {\n"
01394         "    accept if from 192/8;\n"
01395         "}", "");
01396     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01397     REQUIRE_THROWS_WITH(
01398         config::parse(in),
01399         ":2:22(38): expected a valid MAVLink subnet");
01400 }
01401 SECTION("Invalid source condition [3].")
01402 {
01403     tao::pegtl::string_input<> in(
01404         "chain default {\n"
01405         "    accept if from 192\\8;\n"
01406         "}", "");
01407     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01408     REQUIRE_THROWS_WITH(
01409         config::parse(in),
01410         ":2:22(38): expected a valid MAVLink subnet");
01411 }
01412 SECTION("Invalid source condition [4].")
01413 {
01414     tao::pegtl::string_input<> in(
01415         "chain default {\n"
01416         "    accept if from 192:255.255;\n"
01417         "}", "");
01418     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01419     REQUIRE_THROWS_WITH(
01420         config::parse(in),
01421         ":2:22(38): expected a valid MAVLink subnet");
01422 }
01423 }
01424
01425
01426 TEST_CASE("Destination condition.", "[config]")
01427 {
01428     SECTION("Parses destination condition [1].")
01429 {
01430     tao::pegtl::string_input<> in(
01431         "chain default {\n"
01432         "    accept if to 192.0;\n"
01433         "}", "");
01434     auto root = config::parse(in);
01435     REQUIRE(root != nullptr);
01436     REQUIRE(
01437         str(*root) ==
01438         ":001: chain default\n"
01439         ":002: | accept\n"
01440         ":002: | | condition\n"
01441         ":002: | | | dest 192.0\n");
01442 }
01443 SECTION("Parses destination condition (with comments) [1].")
01444 {
01445     tao::pegtl::string_input<> in(
01446         "chain default {# comment\n"
01447         "    accept if to 192.0;# comment\n"
01448         "# comment", "");
01449     auto root = config::parse(in);
01450     REQUIRE(root != nullptr);
01451     REQUIRE(
01452         str(*root) ==
01453         ":001: chain default\n"
01454         ":002: | accept\n"
```

```

01455         ":002: | | condition\n"
01456         ":002: | | | dest 192.0\n";
01457     }
01458 SECTION("Parses destination condition [2].")
01459 {
01460     tao::pegtl::string_input<> in(
01461         "chain default (\n"
01462         "    accept if to 192.0/8;\n"
01463         ")", "");
01464     auto root = config::parse(in);
01465     REQUIRE(root != nullptr);
01466     REQUIRE(
01467         str(*root) ==
01468         ":001: chain default\n"
01469         ":002: | accept\n"
01470         ":002: | | condition\n"
01471         ":002: | | | dest 192.0/8\n");
01472 }
01473 SECTION("Parses destination condition (with comments) [2].")
01474 {
01475     tao::pegtl::string_input<> in(
01476         "chain default (# comment\n"
01477         "    accept if to 192.0/8;# comment\n"
01478         "# comment", "");
01479     auto root = config::parse(in);
01480     REQUIRE(root != nullptr);
01481     REQUIRE(
01482         str(*root) ==
01483         ":001: chain default\n"
01484         ":002: | accept\n"
01485         ":002: | | condition\n"
01486         ":002: | | | dest 192.0/8\n");
01487 }
01488 SECTION("Parses destination condition [3].")
01489 {
01490     tao::pegtl::string_input<> in(
01491         "chain default (\n"
01492         "    accept if to 192.0\\4;\n"
01493         ")", "");
01494     auto root = config::parse(in);
01495     REQUIRE(root != nullptr);
01496     REQUIRE(
01497         str(*root) ==
01498         ":001: chain default\n"
01499         ":002: | accept\n"
01500         ":002: | | condition\n"
01501         ":002: | | | dest 192.0\\4\n");
01502 }
01503 SECTION("Parses destination condition (with comments) [3].")
01504 {
01505     tao::pegtl::string_input<> in(
01506         "chain default (# comment\n"
01507         "    accept if to 192.0\\4;# comment\n"
01508         "# comment", "");
01509     auto root = config::parse(in);
01510     REQUIRE(root != nullptr);
01511     REQUIRE(
01512         str(*root) ==
01513         ":001: chain default\n"
01514         ":002: | accept\n"
01515         ":002: | | condition\n"
01516         ":002: | | | dest 192.0\\4\n");
01517 }
01518 SECTION("Parses destination condition [4].")
01519 {
01520     tao::pegtl::string_input<> in(
01521         "chain default (\n"
01522         "    accept if to 192.0:255.255;\n"
01523         ")", "");
01524     auto root = config::parse(in);
01525     REQUIRE(root != nullptr);
01526     REQUIRE(
01527         str(*root) ==
01528         ":001: chain default\n"
01529         ":002: | accept\n"
01530         ":002: | | condition\n"
01531         ":002: | | | dest 192.0:255.255\n");
01532 }
01533 SECTION("Parses destination condition (with comments) [4].")
01534 {
01535     tao::pegtl::string_input<> in(

```

```
01536     "chain default {# comment\n"
01537     "    accept if to 192.0:255.255;# comment\n"
01538     "#}# comment", "");
01539     auto root = config::parse(in);
01540     REQUIRE(root != nullptr);
01541     REQUIRE(
01542         str(*root) ==
01543         ":001: chain default\n"
01544         ":002: | accept\n"
01545         ":002: | | condition\n"
01546         ":002: | | | dest 192.0:255.255\n");
01547     }
01548 SECTION("Invalid destination condition [1].")
01549 {
01550     tao::pegtl::string_input<> in(
01551         "chain default {\n"
01552         "    accept if to 192;\n"
01553         "}", "");
01554     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01555     REQUIRE_THROWS_WITH(
01556         config::parse(in),
01557         ":2:20(36): expected a valid MAVLink subnet");
01558 }
01559 SECTION("Invalid destination condition [2].")
01560 {
01561     tao::pegtl::string_input<> in(
01562         "chain default {\n"
01563         "    accept if to 192/8;\n"
01564         "}", "");
01565     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01566     REQUIRE_THROWS_WITH(
01567         config::parse(in),
01568         ":2:20(36): expected a valid MAVLink subnet");
01569 }
01570 SECTION("Invalid destination condition [3].")
01571 {
01572     tao::pegtl::string_input<> in(
01573         "chain default {\n"
01574         "    accept if to 192\\8;\n"
01575         "}", "");
01576     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01577     REQUIRE_THROWS_WITH(
01578         config::parse(in),
01579         ":2:20(36): expected a valid MAVLink subnet");
01580 }
01581 SECTION("Invalid destination condition [4].")
01582 {
01583     tao::pegtl::string_input<> in(
01584         "chain default {\n"
01585         "    accept if to 192:255.255;\n"
01586         "}", "");
01587     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01588     REQUIRE_THROWS_WITH(
01589         config::parse(in),
01590         ":2:20(36): expected a valid MAVLink subnet");
01591 }
01592 }
01593
01594
01595 TEST_CASE("Priority.", "[config]")
01596 {
01597     SECTION("Parses priority (no sign).")
01598     {
01599         tao::pegtl::string_input<> in(
01600             "chain default {\n"
01601             "    accept with priority 99;\n"
01602             "}", "");
01603         auto root = config::parse(in);
01604         REQUIRE(root != nullptr);
01605         REQUIRE(
01606             str(*root) ==
01607             ":001: chain default\n"
01608             ":002: | accept\n"
01609             ":002: | | priority 99\n");
01610     }
01611     SECTION("Parses priority (no sign with comments).")
01612     {
01613         tao::pegtl::string_input<> in(
01614             "chain default {# comment\n"
01615             "    accept with priority 99;# comment\n"
01616             "# comment", "");
01617 }
```

```

01617     auto root = config::parse(in);
01618     REQUIRE(root != nullptr);
01619     REQUIRE(
01620         str(*root) ==
01621             ":001: chain default\n"
01622             ":002: | accept\n"
01623             ":002: | | priority 99\\n");
01624     }
01625 SECTION("Parses priority (positive).")
01626 {
01627     tao::pegtl::string_input<> in(
01628         "chain default \\n"
01629             "    accept with priority +99;\\n"
01630             "\\}", "");
01631     auto root = config::parse(in);
01632     REQUIRE(root != nullptr);
01633     REQUIRE(
01634         str(*root) ==
01635             ":001: chain default\\n"
01636             ":002: | accept\\n"
01637             ":002: | | priority +99\\n");
01638     }
01639 SECTION("Parses priority (positive with comments).")
01640 {
01641     tao::pegtl::string_input<> in(
01642         "chain default # comment\\n"
01643             "    accept with priority +99;# comment\\n"
01644             "\\# comment", "");
01645     auto root = config::parse(in);
01646     REQUIRE(root != nullptr);
01647     REQUIRE(
01648         str(*root) ==
01649             ":001: chain default\\n"
01650             ":002: | accept\\n"
01651             ":002: | | priority +99\\n");
01652     }
01653 SECTION("Parses priority (negative).")
01654 {
01655     tao::pegtl::string_input<> in(
01656         "chain default \\n"
01657             "    accept with priority -99;\\n"
01658             "\\}", "");
01659     auto root = config::parse(in);
01660     REQUIRE(root != nullptr);
01661     REQUIRE(
01662         str(*root) ==
01663             ":001: chain default\\n"
01664             ":002: | accept\\n"
01665             ":002: | | priority -99\\n");
01666     }
01667 SECTION("Parses priority (negative with comments).")
01668 {
01669     tao::pegtl::string_input<> in(
01670         "chain default # comment\\n"
01671             "    accept with priority -99;# comment\\n"
01672             "\\# comment", "");
01673     auto root = config::parse(in);
01674     REQUIRE(root != nullptr);
01675     REQUIRE(
01676         str(*root) ==
01677             ":001: chain default\\n"
01678             ":002: | accept\\n"
01679             ":002: | | priority -99\\n");
01680     }
01681 SECTION("'with priority' must be followed by a valid priority.")
01682 {
01683     tao::pegtl::string_input<> in(
01684         "chain default \\n"
01685             "    accept with priority *99;\\n"
01686             "\\}", "");
01687     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01688     REQUIRE_THROWS_WITH(
01689         config::parse(in),
01690         ":2:25(41): expected priority level");
01691     }
01692 SECTION("'with priority' must be followed by a priority.")
01693 {
01694     tao::pegtl::string_input<> in(
01695         "chain default \\n"
01696             "    accept with priority;\\n"
01697             "\\}", "");

```

```
01698     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01699     REQUIRE_THROWS_WITH(
01700         config::parse(in),
01701         ":2:24(40): expected priority level");
01702 }
01703 SECTION("'with' must be followed by 'priority'.")
01704 {
01705     tao::pegtl::string_input<> in(
01706         "chain default {\n"
01707         "    accept with;\n"
01708         "}", "");
01709     REQUIRE_THROWS_AS(config::parse(in), tao::pegtl::parse_error);
01710     REQUIRE_THROWS_WITH(
01711         config::parse(in),
01712         ":2:15(31): 'with' keyword must be followed by the "
01713         "'priority' keyword");
01714 }
01715 }
01716
01717
01718 TEST_CASE("Rule combinations with 'accept'.", "[config]")
01719 {
01720     SECTION("accept")
01721 {
01722     tao::pegtl::string_input<> in(
01723         "chain default {\n"
01724         "    accept;\n"
01725         "}", "");
01726     auto root = config::parse(in);
01727     REQUIRE(root != nullptr);
01728     REQUIRE(
01729         str(*root) ==
01730         ":001: chain default\n"
01731         ":002: | accept\n");
01732 }
01733 SECTION("accept if PING")
01734 {
01735     tao::pegtl::string_input<> in(
01736         "chain default {\n"
01737         "    accept if PING;\n"
01738         "}", "");
01739     auto root = config::parse(in);
01740     REQUIRE(root != nullptr);
01741     REQUIRE(
01742         str(*root) ==
01743         ":001: chain default\n"
01744         ":002: | accept\n"
01745         ":002: | | condition\n"
01746         ":002: | | | packet_type PING\n");
01747 }
01748 SECTION("accept if PING from 127.1")
01749 {
01750     tao::pegtl::string_input<> in(
01751         "chain default {\n"
01752         "    accept if PING from 127.1;\n"
01753         "}", "");
01754     auto root = config::parse(in);
01755     REQUIRE(root != nullptr);
01756     REQUIRE(
01757         str(*root) ==
01758         ":001: chain default\n"
01759         ":002: | accept\n"
01760         ":002: | | condition\n"
01761         ":002: | | | packet_type PING\n"
01762         ":002: | | | source 127.1\n");
01763 }
01764 SECTION("accept if PING to 192.0")
01765 {
01766     tao::pegtl::string_input<> in(
01767         "chain default {\n"
01768         "    accept if PING to 192.0;\n"
01769         "}", "");
01770     auto root = config::parse(in);
01771     REQUIRE(root != nullptr);
01772     REQUIRE(
01773         str(*root) ==
01774         ":001: chain default\n"
01775         ":002: | accept\n"
01776         ":002: | | condition\n"
01777         ":002: | | | packet_type PING\n"
01778         ":002: | | | dest 192.0\n");
```

```

01779     }
01780     SECTION("accept if PING from 127.1 to 192.0")
01781     {
01782         tao::pegtl::string_input<> in(
01783             "chain default \n"
01784             "    accept if PING from 127.1 to 192.0;\n"
01785             ")", "");
01786         auto root = config::parse(in);
01787         REQUIRE(root != nullptr);
01788         REQUIRE(
01789             str(*root) ==
01790             ":001: chain default\n"
01791             ":002: | accept\n"
01792             ":002: | | condition\n"
01793             ":002: | | | packet_type PING\n"
01794             ":002: | | | source 127.1\n"
01795             ":002: | | | dest 192.0\n");
01796     }
01797     SECTION("accept if from 127.1")
01798     {
01799         tao::pegtl::string_input<> in(
01800             "chain default \n"
01801             "    accept if from 127.1;\n"
01802             ")", "");
01803         auto root = config::parse(in);
01804         REQUIRE(root != nullptr);
01805         REQUIRE(
01806             str(*root) ==
01807             ":001: chain default\n"
01808             ":002: | accept\n"
01809             ":002: | | condition\n"
01810             ":002: | | | source 127.1\n");
01811     }
01812     SECTION("accept if to 192.0")
01813     {
01814         tao::pegtl::string_input<> in(
01815             "chain default \n"
01816             "    accept if to 192.0;\n"
01817             ")", "");
01818         auto root = config::parse(in);
01819         REQUIRE(root != nullptr);
01820         REQUIRE(
01821             str(*root) ==
01822             ":001: chain default\n"
01823             ":002: | accept\n"
01824             ":002: | | condition\n"
01825             ":002: | | | dest 192.0\n");
01826     }
01827     SECTION("accept if from 127.1 to 192.0")
01828     {
01829         tao::pegtl::string_input<> in(
01830             "chain default \n"
01831             "    accept if from 127.1 to 192.0;\n"
01832             ")", "");
01833         auto root = config::parse(in);
01834         REQUIRE(root != nullptr);
01835         REQUIRE(
01836             str(*root) ==
01837             ":001: chain default\n"
01838             ":002: | accept\n"
01839             ":002: | | condition\n"
01840             ":002: | | | source 127.1\n"
01841             ":002: | | | dest 192.0\n");
01842     }
01843     SECTION("accept with priority 99")
01844     {
01845         tao::pegtl::string_input<> in(
01846             "chain default \n"
01847             "    accept with priority 99 if to 192.0;\n"
01848             ")", "");
01849         auto root = config::parse(in);
01850         REQUIRE(root != nullptr);
01851         REQUIRE(
01852             str(*root) ==
01853             ":001: chain default\n"
01854             ":002: | accept\n"
01855             ":002: | | priority 99\n"
01856             ":002: | | condition\n"
01857             ":002: | | | dest 192.0\n");
01858     }
01859     SECTION("accept with priority 99 if PING")

```

```
01860  {
01861      tao::pegtl::string_input<> in(
01862          "chain default {\n"
01863          "    accept with priority 99 if PING;\n"
01864          "}", "");
01865      auto root = config::parse(in);
01866      REQUIRE(root != nullptr);
01867      REQUIRE(
01868          str(*root) ==
01869          ":001: chain default\n"
01870          ":002: | accept\n"
01871          ":002: | | priority 99\n"
01872          ":002: | | condition\n"
01873          ":002: | | | packet_type PING\n";
01874      }
01875  SECTION("accept with priority 99 if PING from 127.1")
01876  {
01877      tao::pegtl::string_input<> in(
01878          "chain default {\n"
01879          "    accept with priority 99 if PING from 127.1;\n"
01880          "}", "");
01881      auto root = config::parse(in);
01882      REQUIRE(root != nullptr);
01883      REQUIRE(
01884          str(*root) ==
01885          ":001: chain default\n"
01886          ":002: | accept\n"
01887          ":002: | | priority 99\n"
01888          ":002: | | condition\n"
01889          ":002: | | | packet_type PING\n"
01890          ":002: | | | source 127.1\n");
01891  }
01892  SECTION("accept with priority 99 if PING to 192.0")
01893  {
01894      tao::pegtl::string_input<> in(
01895          "chain default {\n"
01896          "    accept with priority 99 if PING to 192.0;\n"
01897          "}", "");
01898      auto root = config::parse(in);
01899      REQUIRE(root != nullptr);
01900      REQUIRE(
01901          str(*root) ==
01902          ":001: chain default\n"
01903          ":002: | accept\n"
01904          ":002: | | priority 99\n"
01905          ":002: | | condition\n"
01906          ":002: | | | packet_type PING\n"
01907          ":002: | | | dest 192.0\n");
01908  }
01909  SECTION("accept with priority 99 if PING from 127.1 to 192.0")
01910  {
01911      tao::pegtl::string_input<> in(
01912          "chain default {\n"
01913          "    accept with priority 99 if PING from 127.1 to 192.0;\n"
01914          "}", "");
01915      auto root = config::parse(in);
01916      REQUIRE(root != nullptr);
01917      REQUIRE(
01918          str(*root) ==
01919          ":001: chain default\n"
01920          ":002: | accept\n"
01921          ":002: | | priority 99\n"
01922          ":002: | | condition\n"
01923          ":002: | | | packet_type PING\n"
01924          ":002: | | | source 127.1\n"
01925          ":002: | | | dest 192.0\n");
01926  }
01927  SECTION("accept with priority 99 if from 127.1")
01928  {
01929      tao::pegtl::string_input<> in(
01930          "chain default {\n"
01931          "    accept with priority 99 if from 127.1;\n"
01932          "}", "");
01933      auto root = config::parse(in);
01934      REQUIRE(root != nullptr);
01935      REQUIRE(
01936          str(*root) ==
01937          ":001: chain default\n"
01938          ":002: | accept\n"
01939          ":002: | | priority 99\n"
01940          ":002: | | condition\n"
```

```

01941         ":002: | | | source 127.1\n");
01942     }
01943 SECTION("accept with priority 99 if to 192.0")
01944 {
01945     tao::pegtl::string_input<> in(
01946         "chain default {\n"
01947         "    accept with priority 99 if to 192.0;\n"
01948         "}"", "");
01949     auto root = config::parse(in);
01950     REQUIRE(root != nullptr);
01951     REQUIRE(
01952         str(*root) ==
01953         ":001: chain default\n"
01954         ":002: | accept\n"
01955         ":002: | | priority 99\n"
01956         ":002: | | condition\n"
01957         ":002: | | | dest 192.0\n");
01958     }
01959 SECTION("accept with priority 99 if from 127.1 to 192.0")
01960 {
01961     tao::pegtl::string_input<> in(
01962         "chain default {\n"
01963         "    accept with priority 99 if from 127.1 to 192.0;\n"
01964         "}"", "");
01965     auto root = config::parse(in);
01966     REQUIRE(root != nullptr);
01967     REQUIRE(
01968         str(*root) ==
01969         ":001: chain default\n"
01970         ":002: | accept\n"
01971         ":002: | | priority 99\n"
01972         ":002: | | condition\n"
01973         ":002: | | | source 127.1\n"
01974         ":002: | | | dest 192.0\n");
01975     }
01976 }
01977
01978
01979 TEST_CASE("Rule combinations with 'reject'.", "[config]")
01980 {
01981     SECTION("reject")
01982     {
01983         tao::pegtl::string_input<> in(
01984             "chain default {\n"
01985             "    reject;\n"
01986             "}"", "");
01987         auto root = config::parse(in);
01988         REQUIRE(root != nullptr);
01989         REQUIRE(
01990             str(*root) ==
01991             ":001: chain default\n"
01992             ":002: | reject\n");
01993     }
01994     SECTION("reject if PING.")
01995     {
01996         tao::pegtl::string_input<> in(
01997             "chain default {\n"
01998             "    reject if PING;\n"
01999             "}"", "");
02000         auto root = config::parse(in);
02001         REQUIRE(root != nullptr);
02002         REQUIRE(
02003             str(*root) ==
02004             ":001: chain default\n"
02005             ":002: | reject\n"
02006             ":002: | | condition\n"
02007             ":002: | | | packet_type PING\n");
02008     }
02009     SECTION("reject if PING from 127.1")
02010     {
02011         tao::pegtl::string_input<> in(
02012             "chain default {\n"
02013             "    reject if PING from 127.1;\n"
02014             "}"", "");
02015         auto root = config::parse(in);
02016         REQUIRE(root != nullptr);
02017         REQUIRE(
02018             str(*root) ==
02019             ":001: chain default\n"
02020             ":002: | reject\n"
02021             ":002: | | condition\n"

```

```
02022      ":002: | | | packet_type PING\n"
02023      ":002: | | | source 127.1\n";
02024  }
02025 SECTION("reject if PING to 192.0")
02026 {
02027     tao::pegtl::string_input<> in(
02028         "chain default (\n"
02029         "    reject if PING to 192.0;\n"
02030         ")", "");
02031     auto root = config::parse(in);
02032     REQUIRE(root != nullptr);
02033     REQUIRE(
02034         str(*root) ==
02035         ":001: chain default\n"
02036         ":002: | reject\n"
02037         ":002: | | condition\n"
02038         ":002: | | | packet_type PING\n"
02039         ":002: | | | dest 192.0\n");
02040 }
02041 SECTION("reject if PING from 127.1 to 192.0")
02042 {
02043     tao::pegtl::string_input<> in(
02044         "chain default (\n"
02045         "    reject if PING from 127.1 to 192.0;\n"
02046         ")", "");
02047     auto root = config::parse(in);
02048     REQUIRE(root != nullptr);
02049     REQUIRE(
02050         str(*root) ==
02051         ":001: chain default\n"
02052         ":002: | reject\n"
02053         ":002: | | condition\n"
02054         ":002: | | | packet_type PING\n"
02055         ":002: | | | source 127.1\n"
02056         ":002: | | | dest 192.0\n");
02057 }
02058 SECTION("reject if from 127.1")
02059 {
02060     tao::pegtl::string_input<> in(
02061         "chain default (\n"
02062         "    reject if from 127.1;\n"
02063         ")", "");
02064     auto root = config::parse(in);
02065     REQUIRE(root != nullptr);
02066     REQUIRE(
02067         str(*root) ==
02068         ":001: chain default\n"
02069         ":002: | reject\n"
02070         ":002: | | condition\n"
02071         ":002: | | | source 127.1\n");
02072 }
02073 SECTION("reject if to 192.0")
02074 {
02075     tao::pegtl::string_input<> in(
02076         "chain default (\n"
02077         "    reject if to 192.0;\n"
02078         ")", "");
02079     auto root = config::parse(in);
02080     REQUIRE(root != nullptr);
02081     REQUIRE(
02082         str(*root) ==
02083         ":001: chain default\n"
02084         ":002: | reject\n"
02085         ":002: | | condition\n"
02086         ":002: | | | dest 192.0\n");
02087 }
02088 SECTION("reject if from 127.1 to 192.0")
02089 {
02090     tao::pegtl::string_input<> in(
02091         "chain default (\n"
02092         "    reject if from 127.1 to 192.0;\n"
02093         ")", "");
02094     auto root = config::parse(in);
02095     REQUIRE(root != nullptr);
02096     REQUIRE(
02097         str(*root) ==
02098         ":001: chain default\n"
02099         ":002: | reject\n"
02100         ":002: | | condition\n"
02101         ":002: | | | source 127.1\n"
02102         ":002: | | | dest 192.0\n");
```

```

02103     }
02104     SECTION("reject with priority 99")
02105     {
02106         tao::pegtl::string_input<> in(
02107             "chain default \n"
02108             "    reject with priority 99 if to 192.0;\n"
02109             ")");
02110         auto root = config::parse(in);
02111         REQUIRE(root != nullptr);
02112         REQUIRE(
02113             str(*root) ==
02114             ":001: chain default\n"
02115             ":002: | reject\n"
02116             ":002: | | priority 99\n"
02117             ":002: | | condition\n"
02118             ":002: | | | dest 192.0\n");
02119     }
02120     SECTION("reject with priority 99 if PING")
02121     {
02122         tao::pegtl::string_input<> in(
02123             "chain default \n"
02124             "    reject with priority 99 if PING;\n"
02125             ")");
02126         auto root = config::parse(in);
02127         REQUIRE(root != nullptr);
02128         REQUIRE(
02129             str(*root) ==
02130             ":001: chain default\n"
02131             ":002: | reject\n"
02132             ":002: | | priority 99\n"
02133             ":002: | | condition\n"
02134             ":002: | | | packet_type PING\n");
02135     }
02136     SECTION("reject with priority 99 if PING from 127.1")
02137     {
02138         tao::pegtl::string_input<> in(
02139             "chain default \n"
02140             "    reject with priority 99 if PING from 127.1;\n"
02141             ")");
02142         auto root = config::parse(in);
02143         REQUIRE(root != nullptr);
02144         REQUIRE(
02145             str(*root) ==
02146             ":001: chain default\n"
02147             ":002: | reject\n"
02148             ":002: | | priority 99\n"
02149             ":002: | | condition\n"
02150             ":002: | | | packet_type PING\n"
02151             ":002: | | | source 127.1\n");
02152     }
02153     SECTION("reject with priority 99 if PING to 192.0")
02154     {
02155         tao::pegtl::string_input<> in(
02156             "chain default \n"
02157             "    reject with priority 99 if PING to 192.0;\n"
02158             ")");
02159         auto root = config::parse(in);
02160         REQUIRE(root != nullptr);
02161         REQUIRE(
02162             str(*root) ==
02163             ":001: chain default\n"
02164             ":002: | reject\n"
02165             ":002: | | priority 99\n"
02166             ":002: | | condition\n"
02167             ":002: | | | packet_type PING\n"
02168             ":002: | | | dest 192.0\n");
02169     }
02170     SECTION("reject with priority 99 if PING from 127.1 to 192.0")
02171     {
02172         tao::pegtl::string_input<> in(
02173             "chain default \n"
02174             "    reject with priority 99 if PING from 127.1 to 192.0;\n"
02175             ")");
02176         auto root = config::parse(in);
02177         REQUIRE(root != nullptr);
02178         REQUIRE(
02179             str(*root) ==
02180             ":001: chain default\n"
02181             ":002: | reject\n"
02182             ":002: | | priority 99\n"
02183             ":002: | | condition\n"

```

```

02184         ":002: | | | packet_type PING\n"
02185         ":002: | | | source 127.1\n"
02186         ":002: | | | dest 192.0\n");
02187     }
02188     SECTION("reject with priority 99 if from 127.1")
02189     {
02190         tao::pegtl::string_input<> in(
02191             "chain default {\n"
02192             "    reject with priority 99 if from 127.1;\n"
02193             "}", "");
02194         auto root = config::parse(in);
02195         REQUIRE(root != nullptr);
02196         REQUIRE(
02197             str(*root) ==
02198             ":001: chain default\n"
02199             ":002: | reject\n"
02200             ":002: | | priority 99\n"
02201             ":002: | | condition\n"
02202             ":002: | | | source 127.1\n");
02203     }
02204     SECTION("reject with priority 99 if to 192.0")
02205     {
02206         tao::pegtl::string_input<> in(
02207             "chain default {\n"
02208             "    reject with priority 99 if to 192.0;\n"
02209             "}", "");
02210         auto root = config::parse(in);
02211         REQUIRE(root != nullptr);
02212         REQUIRE(
02213             str(*root) ==
02214             ":001: chain default\n"
02215             ":002: | reject\n"
02216             ":002: | | priority 99\n"
02217             ":002: | | condition\n"
02218             ":002: | | | dest 192.0\n");
02219     }
02220     SECTION("reject with priority 99 if from 127.1 to 192.0")
02221     {
02222         tao::pegtl::string_input<> in(
02223             "chain default {\n"
02224             "    reject with priority 99 if from 127.1 to 192.0;\n"
02225             "}", "");
02226         auto root = config::parse(in);
02227         REQUIRE(root != nullptr);
02228         REQUIRE(
02229             str(*root) ==
02230             ":001: chain default\n"
02231             ":002: | reject\n"
02232             ":002: | | priority 99\n"
02233             ":002: | | condition\n"
02234             ":002: | | | source 127.1\n"
02235             ":002: | | | dest 192.0\n");
02236     }
02237 }
02238
02239
02240 TEST_CASE("Rule combinations with 'call'.", "[config]")
02241 {
02242     SECTION("call some_name10")
02243     {
02244         tao::pegtl::string_input<> in(
02245             "chain default {\n"
02246             "    call some_name10;\n"
02247             "}", "");
02248         auto root = config::parse(in);
02249         REQUIRE(root != nullptr);
02250         REQUIRE(
02251             str(*root) ==
02252             ":001: chain default\n"
02253             ":002: | call some_name10\n");
02254     }
02255     SECTION("call some_name10 if PING.")
02256     {
02257         tao::pegtl::string_input<> in(
02258             "chain default {\n"
02259             "    call some_name10 if PING;\n"
02260             "}", "");
02261         auto root = config::parse(in);
02262         REQUIRE(root != nullptr);
02263         REQUIRE(
02264             str(*root) ==

```

```

02265      ":001: chain default\n"
02266      ":002: | call some_name10\n"
02267      ":002: | | condition\n"
02268      ":002: | | | packet_type PING\n";
02269  }
02270 SECTION("call some_name10 if PING from 127.1")
02271 {
02272     tao::pegtl::string_input<> in(
02273         "chain default {\n"
02274         "    call some_name10 if PING from 127.1;\n"
02275         "}", "");
02276     auto root = config::parse(in);
02277     REQUIRE(root != nullptr);
02278     REQUIRE(
02279         str(*root) ==
02280         ":001: chain default\n"
02281         ":002: | call some_name10\n"
02282         ":002: | | condition\n"
02283         ":002: | | | packet_type PING\n"
02284         ":002: | | | source 127.1\n");
02285     }
02286 SECTION("call some_name10 if PING to 192.0")
02287 {
02288     tao::pegtl::string_input<> in(
02289         "chain default {\n"
02290         "    call some_name10 if PING to 192.0;\n"
02291         "}", "");
02292     auto root = config::parse(in);
02293     REQUIRE(root != nullptr);
02294     REQUIRE(
02295         str(*root) ==
02296         ":001: chain default\n"
02297         ":002: | call some_name10\n"
02298         ":002: | | condition\n"
02299         ":002: | | | packet_type PING\n"
02300         ":002: | | | dest 192.0\n");
02301 }
02302 SECTION("call some_name10 if PING from 127.1 to 192.0")
02303 {
02304     tao::pegtl::string_input<> in(
02305         "chain default {\n"
02306         "    call some_name10 if PING from 127.1 to 192.0;\n"
02307         "}", "");
02308     auto root = config::parse(in);
02309     REQUIRE(root != nullptr);
02310     REQUIRE(
02311         str(*root) ==
02312         ":001: chain default\n"
02313         ":002: | call some_name10\n"
02314         ":002: | | condition\n"
02315         ":002: | | | packet_type PING\n"
02316         ":002: | | | source 127.1\n"
02317         ":002: | | | dest 192.0\n");
02318 }
02319 SECTION("call some_name10 if from 127.1")
02320 {
02321     tao::pegtl::string_input<> in(
02322         "chain default {\n"
02323         "    call some_name10 if from 127.1;\n"
02324         "}", "");
02325     auto root = config::parse(in);
02326     REQUIRE(root != nullptr);
02327     REQUIRE(
02328         str(*root) ==
02329         ":001: chain default\n"
02330         ":002: | call some_name10\n"
02331         ":002: | | condition\n"
02332         ":002: | | | source 127.1\n");
02333 }
02334 SECTION("call some_name10 if to 192.0")
02335 {
02336     tao::pegtl::string_input<> in(
02337         "chain default {\n"
02338         "    call some_name10 if to 192.0;\n"
02339         "}", "");
02340     auto root = config::parse(in);
02341     REQUIRE(root != nullptr);
02342     REQUIRE(
02343         str(*root) ==
02344         ":001: chain default\n"
02345         ":002: | call some_name10\n"

```

```
02346      ":002: | | condition\n"
02347      ":002: | | | dest 192.0\n";
02348  }
02349 SECTION("call some_name10 if from 127.1 to 192.0")
02350 {
02351     tao::pegtl::string_input<> in(
02352         "chain default (\n"
02353         "    call some_name10 if from 127.1 to 192.0;\n"
02354         ")", "");
02355     auto root = config::parse(in);
02356     REQUIRE(root != nullptr);
02357     REQUIRE(
02358         str(*root) ==
02359         ":001: chain default\n"
02360         ":002: | call some_name10\n"
02361         ":002: | | condition\n"
02362         ":002: | | | source 127.1\n"
02363         ":002: | | | dest 192.0\n");
02364 }
02365 SECTION("call some_name10 with priority 99")
02366 {
02367     tao::pegtl::string_input<> in(
02368         "chain default (\n"
02369         "    call some_name10 with priority 99 if to 192.0;\n"
02370         ")", "");
02371     auto root = config::parse(in);
02372     REQUIRE(root != nullptr);
02373     REQUIRE(
02374         str(*root) ==
02375         ":001: chain default\n"
02376         ":002: | call some_name10\n"
02377         ":002: | | priority 99\n"
02378         ":002: | | condition\n"
02379         ":002: | | | dest 192.0\n");
02380 }
02381 SECTION("call some_name10 with priority 99 if PING")
02382 {
02383     tao::pegtl::string_input<> in(
02384         "chain default (\n"
02385         "    call some_name10 with priority 99 if PING;\n"
02386         ")", "");
02387     auto root = config::parse(in);
02388     REQUIRE(root != nullptr);
02389     REQUIRE(
02390         str(*root) ==
02391         ":001: chain default\n"
02392         ":002: | call some_name10\n"
02393         ":002: | | priority 99\n"
02394         ":002: | | condition\n"
02395         ":002: | | | packet_type PING\n");
02396 }
02397 SECTION("call some_name10 with priority 99 if PING from 127.1")
02398 {
02399     tao::pegtl::string_input<> in(
02400         "chain default (\n"
02401         "    call some_name10 with priority 99 if PING from 127.1;\n"
02402         ")", "");
02403     auto root = config::parse(in);
02404     REQUIRE(root != nullptr);
02405     REQUIRE(
02406         str(*root) ==
02407         ":001: chain default\n"
02408         ":002: | call some_name10\n"
02409         ":002: | | priority 99\n"
02410         ":002: | | condition\n"
02411         ":002: | | | packet_type PING\n"
02412         ":002: | | | source 127.1\n");
02413 }
02414 SECTION("call some_name10 with priority 99 if PING to 192.0")
02415 {
02416     tao::pegtl::string_input<> in(
02417         "chain default (\n"
02418         "    call some_name10 with priority 99 if PING to 192.0;\n"
02419         ")", "");
02420     auto root = config::parse(in);
02421     REQUIRE(root != nullptr);
02422     REQUIRE(
02423         str(*root) ==
02424         ":001: chain default\n"
02425         ":002: | call some_name10\n"
02426         ":002: | | priority 99\n"
```

```

02427      ":002: | | condition\n"
02428      ":002: | | | packet_type PING\n"
02429      ":002: | | | dest 192.0\n");
02430 }
02431 SECTION("call some_name10 with priority 99 if PING from 127.1 to 192.0")
02432 {
02433     tao::pegtl::string_input<> in(
02434         "chain default {\n"
02435         "    call some_name10 with priority 99 "
02436         "if PING from 127.1 to 192.0;\n"
02437         "}", "");
02438     auto root = config::parse(in);
02439     REQUIRE(root != nullptr);
02440     REQUIRE(
02441         str(*root) ==
02442         ":001: chain default\n"
02443         ":002: | call some_name10\n"
02444         ":002: | | priority 99\n"
02445         ":002: | | condition\n"
02446         ":002: | | | packet_type PING\n"
02447         ":002: | | | source 127.1\n"
02448         ":002: | | | dest 192.0\n");
02449 }
02450 SECTION("call some_name10 with priority 99 if from 127.1")
02451 {
02452     tao::pegtl::string_input<> in(
02453         "chain default {\n"
02454         "    call some_name10 with priority 99 if from 127.1;\n"
02455         "}", "");
02456     auto root = config::parse(in);
02457     REQUIRE(root != nullptr);
02458     REQUIRE(
02459         str(*root) ==
02460         ":001: chain default\n"
02461         ":002: | call some_name10\n"
02462         ":002: | | priority 99\n"
02463         ":002: | | condition\n"
02464         ":002: | | | source 127.1\n");
02465 }
02466 SECTION("call some_name10 with priority 99 if to 192.0")
02467 {
02468     tao::pegtl::string_input<> in(
02469         "chain default {\n"
02470         "    call some_name10 with priority 99 if to 192.0;\n"
02471         "}", "");
02472     auto root = config::parse(in);
02473     REQUIRE(root != nullptr);
02474     REQUIRE(
02475         str(*root) ==
02476         ":001: chain default\n"
02477         ":002: | call some_name10\n"
02478         ":002: | | priority 99\n"
02479         ":002: | | condition\n"
02480         ":002: | | | dest 192.0\n");
02481 }
02482 SECTION("call some_name10 with priority 99 if from 127.1 to 192.0")
02483 {
02484     tao::pegtl::string_input<> in(
02485         "chain default {\n"
02486         "    call some_name10 with priority 99 if from 127.1 to 192.0;\n"
02487         "}", "");
02488     auto root = config::parse(in);
02489     REQUIRE(root != nullptr);
02490     REQUIRE(
02491         str(*root) ==
02492         ":001: chain default\n"
02493         ":002: | call some_name10\n"
02494         ":002: | | priority 99\n"
02495         ":002: | | condition\n"
02496         ":002: | | | source 127.1\n"
02497         ":002: | | | dest 192.0\n");
02498 }
02499 }
02500
02501
02502 TEST_CASE("Rule combinations with 'goto'.", "[config]")
02503 {
02504     SECTION("goto some_name10")
02505     {
02506         tao::pegtl::string_input<> in(
02507             "chain default {\n"

```

```

02508         "      goto some_name10;\n"
02509         ")", "");
02510     auto root = config::parse(in);
02511     REQUIRE(root != nullptr);
02512     REQUIRE(
02513         str(*root) ==
02514         ":001: chain default\n"
02515         ":002: | goto some_name10\n");
02516 }
02517 SECTION("goto some_name10 if PING.")
02518 {
02519     tao::pegtl::string_input<> in(
02520         "chain default {\n"
02521         "    goto some_name10 if PING;\n"
02522         "}", "");
02523     auto root = config::parse(in);
02524     REQUIRE(root != nullptr);
02525     REQUIRE(
02526         str(*root) ==
02527         ":001: chain default\n"
02528         ":002: | goto some_name10\n"
02529         ":002: | | condition\n"
02530         ":002: | | | packet_type PING\n");
02531 }
02532 SECTION("goto some_name10 if PING from 127.1")
02533 {
02534     tao::pegtl::string_input<> in(
02535         "chain default {\n"
02536         "    goto some_name10 if PING from 127.1;\n"
02537         "}", "");
02538     auto root = config::parse(in);
02539     REQUIRE(root != nullptr);
02540     REQUIRE(
02541         str(*root) ==
02542         ":001: chain default\n"
02543         ":002: | goto some_name10\n"
02544         ":002: | | condition\n"
02545         ":002: | | | packet_type PING\n"
02546         ":002: | | | source 127.1\n");
02547 }
02548 SECTION("goto some_name10 if PING to 192.0")
02549 {
02550     tao::pegtl::string_input<> in(
02551         "chain default {\n"
02552         "    goto some_name10 if PING to 192.0;\n"
02553         "}", "");
02554     auto root = config::parse(in);
02555     REQUIRE(root != nullptr);
02556     REQUIRE(
02557         str(*root) ==
02558         ":001: chain default\n"
02559         ":002: | goto some_name10\n"
02560         ":002: | | condition\n"
02561         ":002: | | | packet_type PING\n"
02562         ":002: | | | dest 192.0\n");
02563 }
02564 SECTION("goto some_name10 if PING from 127.1 to 192.0")
02565 {
02566     tao::pegtl::string_input<> in(
02567         "chain default {\n"
02568         "    goto some_name10 if PING from 127.1 to 192.0;\n"
02569         "}", "");
02570     auto root = config::parse(in);
02571     REQUIRE(root != nullptr);
02572     REQUIRE(
02573         str(*root) ==
02574         ":001: chain default\n"
02575         ":002: | goto some_name10\n"
02576         ":002: | | condition\n"
02577         ":002: | | | packet_type PING\n"
02578         ":002: | | | source 127.1\n"
02579         ":002: | | | dest 192.0\n");
02580 }
02581 SECTION("goto some_name10 if from 127.1")
02582 {
02583     tao::pegtl::string_input<> in(
02584         "chain default {\n"
02585         "    goto some_name10 if from 127.1;\n"
02586         "}", "");
02587     auto root = config::parse(in);
02588     REQUIRE(root != nullptr);

```

```

02589     REQUIRE(
02590         str(*root) ==
02591             ":001: chain default\n"
02592             "| goto some_name10\n"
02593             "| | condition\n"
02594             "| | | source 127.1\n");
02595     }
02596 SECTION("goto some_name10 if to 192.0")
02597 {
02598     tao::pegtl::string_input<> in(
02599         "chain default {\n"
02600             "    goto some_name10 if to 192.0;\n"
02601             "}\n", "");
02602     auto root = config::parse(in);
02603     REQUIRE(root != nullptr);
02604     REQUIRE(
02605         str(*root) ==
02606             ":001: chain default\n"
02607             "| goto some_name10\n"
02608             "| | condition\n"
02609             "| | | dest 192.0\n");
02610     }
02611 SECTION("goto some_name10 if from 127.1 to 192.0")
02612 {
02613     tao::pegtl::string_input<> in(
02614         "chain default {\n"
02615             "    goto some_name10 if from 127.1 to 192.0;\n"
02616             "}\n", "");
02617     auto root = config::parse(in);
02618     REQUIRE(root != nullptr);
02619     REQUIRE(
02620         str(*root) ==
02621             ":001: chain default\n"
02622             "| goto some_name10\n"
02623             "| | condition\n"
02624             "| | | source 127.1\n"
02625             "| | | dest 192.0\n");
02626     }
02627 SECTION("goto some_name10 with priority 99")
02628 {
02629     tao::pegtl::string_input<> in(
02630         "chain default {\n"
02631             "    goto some_name10 with priority 99 if to 192.0;\n"
02632             "}\n", "");
02633     auto root = config::parse(in);
02634     REQUIRE(root != nullptr);
02635     REQUIRE(
02636         str(*root) ==
02637             ":001: chain default\n"
02638             "| goto some_name10\n"
02639             "| | priority 99\n"
02640             "| | condition\n"
02641             "| | | dest 192.0\n");
02642     }
02643 SECTION("goto some_name10 with priority 99 if PING")
02644 {
02645     tao::pegtl::string_input<> in(
02646         "chain default {\n"
02647             "    goto some_name10 with priority 99 if PING;\n"
02648             "}\n", "");
02649     auto root = config::parse(in);
02650     REQUIRE(root != nullptr);
02651     REQUIRE(
02652         str(*root) ==
02653             ":001: chain default\n"
02654             "| goto some_name10\n"
02655             "| | priority 99\n"
02656             "| | condition\n"
02657             "| | | packet_type PING\n");
02658     }
02659 SECTION("goto some_name10 with priority 99 if PING from 127.1")
02660 {
02661     tao::pegtl::string_input<> in(
02662         "chain default {\n"
02663             "    goto some_name10 with priority 99 if PING from 127.1;\n"
02664             "}\n", "");
02665     auto root = config::parse(in);
02666     REQUIRE(root != nullptr);
02667     REQUIRE(
02668         str(*root) ==
02669             ":001: chain default\n"

```

```
02670      ":002: | goto some_name10\n"
02671      ":002: | | priority 99\n"
02672      ":002: | | condition\n"
02673      ":002: | | | packet_type PING\n"
02674      ":002: | | | source 127.1\n";
02675  }
02676 SECTION("goto some_name10 with priority 99 if PING to 192.0")
02677 {
02678     tao::pegtl::string_input<> in(
02679         "chain default {\n"
02680         "    goto some_name10 with priority 99 if PING to 192.0;\n"
02681         "}", "");
02682     auto root = config::parse(in);
02683     REQUIRE(root != nullptr);
02684     REQUIRE(
02685         str(*root) ==
02686         ":001: chain default\n"
02687         ":002: | goto some_name10\n"
02688         ":002: | | priority 99\n"
02689         ":002: | | condition\n"
02690         ":002: | | | packet_type PING\n"
02691         ":002: | | | dest 192.0\n");
02692     }
02693 SECTION("goto some_name10 with priority 99 if PING from 127.1 to 192.0")
02694 {
02695     tao::pegtl::string_input<> in(
02696         "chain default {\n"
02697         "    goto some_name10 with priority 99 "
02698         "if PING from 127.1 to 192.0;\n"
02699         "}", "");
02700     auto root = config::parse(in);
02701     REQUIRE(root != nullptr);
02702     REQUIRE(
02703         str(*root) ==
02704         ":001: chain default\n"
02705         ":002: | goto some_name10\n"
02706         ":002: | | priority 99\n"
02707         ":002: | | condition\n"
02708         ":002: | | | packet_type PING\n"
02709         ":002: | | | source 127.1\n"
02710         ":002: | | | dest 192.0\n");
02711     }
02712 SECTION("goto some_name10 with priority 99 if from 127.1")
02713 {
02714     tao::pegtl::string_input<> in(
02715         "chain default {\n"
02716         "    goto some_name10 with priority 99 if from 127.1;\n"
02717         "}", "");
02718     auto root = config::parse(in);
02719     REQUIRE(root != nullptr);
02720     REQUIRE(
02721         str(*root) ==
02722         ":001: chain default\n"
02723         ":002: | goto some_name10\n"
02724         ":002: | | priority 99\n"
02725         ":002: | | condition\n"
02726         ":002: | | | source 127.1\n");
02727     }
02728 SECTION("goto some_name10 with priority 99 if to 192.0")
02729 {
02730     tao::pegtl::string_input<> in(
02731         "chain default {\n"
02732         "    goto some_name10 with priority 99 if to 192.0;\n"
02733         "}", "");
02734     auto root = config::parse(in);
02735     REQUIRE(root != nullptr);
02736     REQUIRE(
02737         str(*root) ==
02738         ":001: chain default\n"
02739         ":002: | goto some_name10\n"
02740         ":002: | | priority 99\n"
02741         ":002: | | condition\n"
02742         ":002: | | | dest 192.0\n");
02743     }
02744 SECTION("goto some_name10 with priority 99 if from 127.1 to 192.0")
02745 {
02746     tao::pegtl::string_input<> in(
02747         "chain default {\n"
02748         "    goto some_name10 with priority 99 if from 127.1 to 192.0;\n"
02749         "}", "");
02750     auto root = config::parse(in);
```

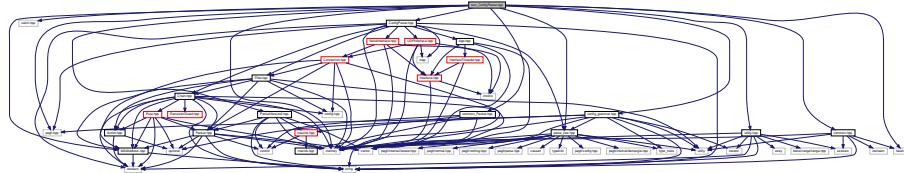
```

02751     REQUIRE(root != nullptr);
02752     REQUIRE(
02753         str(*root) ==
02754             ":001: chain default\n"
02755             ":002: | goto some_name10\n"
02756             ":002: | | priority 99\n"
02757             ":002: | | condition\n"
02758             ":002: | | | source 127.1\n"
02759             ":002: | | | dest 192.0\n";
02760     )
02761 }
02762
02763
02764 TEST_CASE("'print_node' method prints an abstract syntax tree.", "[config]")
02765 {
02766     SECTION("Print without line information.")
02767     {
02768         tao::pegtl::string_input<> in(
02769             "chain default {\n"
02770             "    call some_name10 with priority 99 "
02771             "if PING from 127.1 to 192.0;\n"
02772             "}", "");
02773         auto root = config::parse(in);
02774         REQUIRE(root != nullptr);
02775         std::stringstream ss;
02776         config::print_node(ss, *root, false);
02777         REQUIRE(
02778             ss.str() ==
02779                 "chain default\n"
02780                 "| call some_name10\n"
02781                 "| | priority 99\n"
02782                 "| | condition\n"
02783                 "| | | packet_type PING\n"
02784                 "| | | source 127.1\n"
02785                 "| | | dest 192.0\n");
02786     }
02787     SECTION("Print with line information.")
02788     {
02789         tao::pegtl::string_input<> in(
02790             "chain default {\n"
02791             "    call some_name10 with priority 99 "
02792             "if PING from 127.1 to 192.0;\n"
02793             "}", "");
02794         auto root = config::parse(in);
02795         REQUIRE(root != nullptr);
02796         std::stringstream ss;
02797         config::print_node(ss, *root, true);
02798         REQUIRE(
02799             ss.str() ==
02800                 ":001: chain default\n"
02801                 ":002: | call some_name10\n"
02802                 ":002: | | priority 99\n"
02803                 ":002: | | condition\n"
02804                 ":002: | | | packet_type PING\n"
02805                 ":002: | | | source 127.1\n"
02806                 ":002: | | | dest 192.0\n");
02807     }
02808 }
```

16.201 test_ConfigParser.cpp File Reference

```
#include <memory>
#include <catch.hpp>
#include <fakeit.hpp>
#include <pegtl.hpp>
#include "config_grammar.hpp"
#include "ConfigParser.hpp"
#include "MAVAddress.hpp"
#include "PacketVersion2.hpp"
#include "parse_tree.hpp"
#include "utility.hpp"
```

```
#include "common.hpp"
#include "common_Packet.hpp"
Include dependency graph for test_ConfigParser.cpp:
```



Functions

- [TEST_CASE \("init_chains' initializes a map of chain names to empty chains" "\[ConfigParser\]"\)](#)
- [TEST_CASE \("parse_action' parses an action from the given AST node.", "\[ConfigParser\]"\)](#)
- [TEST_CASE \("parse_condition' parses a condition from the given AST node.", "\[ConfigParser\]"\)](#)
- [TEST_CASE \("parse_chain' parses an action from the given AST node.", "\[ConfigParser\]"\)](#)
- [TEST_CASE \("parse_filter' parses the filter from the given AST root node.", "\[ConfigParser\]"\)](#)
- [TEST_CASE \("parse_serial' parses a serial interface from a serial interface " "AST node.", "\[ConfigParser\]"\)](#)
- [TEST_CASE \("parse_udp' parses a UDP interface from a UDP interface AST node.", "\[ConfigParser\]"\)](#)
- [TEST_CASE \("parse_interfaces' parses serial port and UDP interfaces from " "the root node.", "\[ConfigParser\]"\)](#)
- [TEST_CASE \("ConfigParser can parse a file.", "\[ConfigParser\]"\)](#)
- [TEST_CASE \("ConfigParser are printable.", "\[ConfigParser\]"\)](#)
- [TEST_CASE \("ConfigParser's 'make_app' method returns an application object.", "\[ConfigParser\]"\)](#)

16.201.1 Function Documentation

16.201.1.1 TEST_CASE() [1/11]

```
TEST_CASE (
    "'init_chains' initializes a map of chain names to empty chains" "" [ConfigParser] )
```

Definition at line 35 of file [test_ConfigParser.cpp](#).

16.201.1.2 TEST_CASE() [2/11]

```
TEST_CASE (
    "'parse_action' parses an action from the given AST node." ,
    "" [ConfigParser] )
```

Definition at line 101 of file [test_ConfigParser.cpp](#).

16.201.1.3 TEST_CASE() [3/11]

```
TEST_CASE (
    "'parse_condition' parses a condition from the given AST node." ,
    "" [ConfigParser] )
```

Definition at line 432 of file [test_ConfigParser.cpp](#).

16.201.1.4 TEST_CASE() [4/11]

```
TEST_CASE (
    "'parse_chain' parses an action from the given AST node." ,
    "" [ConfigParser] )
```

Definition at line 564 of file [test_ConfigParser.cpp](#).

16.201.1.5 TEST_CASE() [5/11]

```
TEST_CASE (
    "'parse_filter' parses the filter from the given AST root node." ,
    "" [ConfigParser] )
```

Definition at line 661 of file [test_ConfigParser.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.201.1.6 TEST_CASE() [6/11]

```
TEST_CASE (
    "'parse_serial' parses a serial interface from a serial interface " "AST node." ,
    "" [ConfigParser] )
```

Definition at line 741 of file [test_ConfigParser.cpp](#).

16.201.1.7 TEST_CASE() [7/11]

```
TEST_CASE (
    "'parse_udp' parses a UDP interface from a UDP interface AST node." ,
    "" [ConfigParser] )
```

Definition at line 903 of file [test_ConfigParser.cpp](#).

16.201.1.8 TEST_CASE() [8/11]

```
TEST_CASE (
    "'parse_interfaces' parses serial port and UDP interfaces from " "the root node." ,
    "" [ConfigParser] )
```

Definition at line 997 of file [test_ConfigParser.cpp](#).

References [config::parse\(\)](#).

Here is the call graph for this function:



16.201.1.9 TEST_CASE() [9/11]

```
TEST_CASE (
    "ConfigParser can parse a file." ,
    "" [ConfigParser] )
```

Definition at line 1084 of file [test_ConfigParser.cpp](#).

16.201.1.10 TEST_CASE() [10/11]

```
TEST_CASE (
    "ConfigParser are printable." ,
    "" [ConfigParser] )
```

Definition at line 1098 of file [test_ConfigParser.cpp](#).

16.201.1.11 TEST_CASE() [11/11]

```
TEST_CASE (
    "ConfigParser's 'make_app' method returns an application object." ,
    "" [ConfigParser] )
```

Definition at line 1133 of file [test_ConfigParser.cpp](#).

16.202 test_ConfigParser.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <memory>
00019
00020 #include <catch.hpp>
00021 #include <fakeit.hpp>
00022 #include <pegtl.hpp>
00023
00024 #include "config_grammar.hpp"
00025 #include "ConfigParser.hpp"
00026 #include "MAVAddress.hpp"
00027 #include "PacketVersion2.hpp"
```

```
00028 #include "parse_tree.hpp"
00029 #include "utility.hpp"
00030
00031 #include "common.hpp"
00032 #include "common_Packet.hpp"
00033
00034
00035 TEST_CASE("'init_chains' initializes a map of chain names to empty chains"
00036     "[ConfigParser]")
00037 {
00038     SECTION("default chain listed first")
00039     {
00040         tao::pegtl::string_input<> in(
00041             "chain default {}\n"
00042             "chain first_chain {}\n"
00043             "chain second_chain {}\n"
00044             "chain third_chain {}\n", "");
00045         auto root = config::parse(in);
00046         REQUIRE(root != nullptr);
00047         auto chains = init_chains(*root);
00048         REQUIRE(chains.count("default") == 0);
00049         REQUIRE(chains.count("first_chain") == 1);
00050         REQUIRE(chains.count("second_chain") == 1);
00051         REQUIRE(chains.count("third_chain") == 1);
00052         REQUIRE(chains["first_chain"] != nullptr);
00053         REQUIRE(chains["second_chain"] != nullptr);
00054         REQUIRE(chains["third_chain"] != nullptr);
00055         REQUIRE(
00056             str(*chains["first_chain"]) ==
00057             "chain first_chain {\n"
00058             "}");
00059         REQUIRE(
00060             str(*chains["second_chain"]) ==
00061             "chain second_chain {\n"
00062             "}");
00063         REQUIRE(
00064             str(*chains["third_chain"]) ==
00065             "chain third_chain {\n"
00066             "}");
00067     }
00068     SECTION("default chain listed last")
00069     {
00070         tao::pegtl::string_input<> in(
00071             "chain first_chain {}\n"
00072             "chain second_chain {}\n"
00073             "chain third_chain {}\n"
00074             "chain default {}\n", "");
00075         auto root = config::parse(in);
00076         REQUIRE(root != nullptr);
00077         auto chains = init_chains(*root);
00078         REQUIRE(chains.count("default") == 0);
00079         REQUIRE(chains.count("first_chain") == 1);
00080         REQUIRE(chains.count("second_chain") == 1);
00081         REQUIRE(chains.count("third_chain") == 1);
00082         REQUIRE(chains["first_chain"] != nullptr);
00083         REQUIRE(chains["second_chain"] != nullptr);
00084         REQUIRE(chains["third_chain"] != nullptr);
00085         REQUIRE(
00086             str(*chains["first_chain"]) ==
00087             "chain first_chain {\n"
00088             "}");
00089         REQUIRE(
00090             str(*chains["second_chain"]) ==
00091             "chain second_chain {\n"
00092             "}");
00093         REQUIRE(
00094             str(*chains["third_chain"]) ==
00095             "chain third_chain {\n"
00096             "}");
00097     }
00098 }
00099
00100
00101 TEST_CASE("'parse_action' parses an action from the given AST node.",
00102     "[ConfigParser]")
00103 {
00104     std::map<std::string, std::shared_ptr<Chain>> chains;
00105     chains["some_chain"] = std::make_shared<Chain>("some_chain");
00106     SECTION("Accept action.")
00107     {
00108         tao::pegtl::string_input<> in(
```

```

00109         "chain default {\n"
00110         "    accept;\n"
00111         "}\n", "");
00112     auto root = config::parse(in);
00113     REQUIRE(root != nullptr);
00114     REQUIRE_FALSE(root->children.empty());
00115     REQUIRE(root->children[0] != nullptr);
00116     REQUIRE_FALSE(root->children[0]->children.empty());
00117     REQUIRE(root->children[0]->children[0] != nullptr);
00118     auto action = parse_action(
00119             *root->children[0]->children[0], {}, {}, chains);
00120     REQUIRE(str(*action) == "accept");
00121 }
00122 SECTION("Accept action, with priority.")
00123 {
00124     tao::pegtl::string_input<> in(
00125         "chain default {\n"
00126         "    accept with priority -1;\n"
00127         "}\n", "");
00128     auto root = config::parse(in);
00129     REQUIRE(root != nullptr);
00130     REQUIRE_FALSE(root->children.empty());
00131     REQUIRE(root->children[0] != nullptr);
00132     REQUIRE_FALSE(root->children[0]->children.empty());
00133     REQUIRE(root->children[0]->children[0] != nullptr);
00134     auto action = parse_action(
00135             *root->children[0]->children[0], -1, {}, chains);
00136     REQUIRE(str(*action) == "accept with priority -1");
00137 }
00138 SECTION("Accept action, with condition.")
00139 {
00140     tao::pegtl::string_input<> in(
00141         "chain default {\n"
00142         "    accept if PING from 192.168 to 127.0/8;\n"
00143         "}\n", "");
00144     auto root = config::parse(in);
00145     REQUIRE(root != nullptr);
00146     REQUIRE_FALSE(root->children.empty());
00147     REQUIRE(root->children[0] != nullptr);
00148     REQUIRE_FALSE(root->children[0]->children.empty());
00149     REQUIRE(root->children[0]->children[0] != nullptr);
00150     auto action =
00151         parse_action(
00152             *root->children[0]->children[0], {},
00153             If().type("PING").from("192.168").to("127.0/8"), chains);
00154     REQUIRE(str(*action) == "accept if PING from 192.168 to 127.0/8");
00155 }
00156 SECTION("Accept action, with priority and condition.")
00157 {
00158     tao::pegtl::string_input<> in(
00159         "chain default {\n"
00160         "    accept with priority -1 if PING from 192.168 to 127.0/8;\n"
00161         "}\n", "");
00162     auto root = config::parse(in);
00163     REQUIRE(root != nullptr);
00164     REQUIRE_FALSE(root->children.empty());
00165     REQUIRE(root->children[0] != nullptr);
00166     REQUIRE_FALSE(root->children[0]->children.empty());
00167     REQUIRE(root->children[0]->children[0] != nullptr);
00168     auto action =
00169         parse_action(
00170             *root->children[0]->children[0], -1,
00171             If().type("PING").from("192.168").to("127.0/8"), chains);
00172     REQUIRE(
00173         str(*action) ==
00174         "accept with priority -1 if PING from 192.168 to 127.0/8");
00175 }
00176 SECTION("Reject action.")
00177 {
00178     tao::pegtl::string_input<> in(
00179         "chain default {\n"
00180         "    reject;\n"
00181         "}\n", "");
00182     auto root = config::parse(in);
00183     REQUIRE(root != nullptr);
00184     REQUIRE_FALSE(root->children.empty());
00185     REQUIRE(root->children[0] != nullptr);
00186     REQUIRE_FALSE(root->children[0]->children.empty());
00187     REQUIRE(root->children[0]->children[0] != nullptr);
00188     auto action = parse_action(
00189             *root->children[0]->children[0], {}, {}, chains);

```

```

00190     REQUIRE(str(*action) == "reject");
00191 }
00192 SECTION("Reject action, with priority (priority ignored for reject).")
00193 {
00194     tao::pegtl::string_input<> in(
00195         "chain default {\n"
00196         "    reject with priority -1;\n"
00197         "}\n", "");
00198     auto root = config::parse(in);
00199     REQUIRE(root != nullptr);
00200     REQUIRE_FALSE(root->children.empty());
00201     REQUIRE(root->children[0] != nullptr);
00202     REQUIRE_FALSE(root->children[0]->children.empty());
00203     REQUIRE(root->children[0]->children[0] != nullptr);
00204     auto action = parse_action(
00205         *root->children[0]->children[0], -1, {}, chains);
00206     REQUIRE(str(*action) == "reject");
00207 }
00208 SECTION("Reject action, with condition.")
00209 {
00210     tao::pegtl::string_input<> in(
00211         "chain default {\n"
00212         "    reject if PING from 192.168 to 127.0/8;\n"
00213         "}\n", "");
00214     auto root = config::parse(in);
00215     REQUIRE(root != nullptr);
00216     REQUIRE_FALSE(root->children.empty());
00217     REQUIRE(root->children[0] != nullptr);
00218     REQUIRE_FALSE(root->children[0]->children.empty());
00219     REQUIRE(root->children[0]->children[0] != nullptr);
00220     auto action =
00221         parse_action(
00222             *root->children[0]->children[0], {},
00223             If().type("PING").from("192.168").to("127.0/8"), chains);
00224     REQUIRE(str(*action) == "reject if PING from 192.168 to 127.0/8");
00225 }
00226 SECTION("Reject action, with priority and condition "
00227     "(priority ignored for reject).")
00228 {
00229     tao::pegtl::string_input<> in(
00230         "chain default {\n"
00231         "    reject if PING from 192.168 to 127.0/8;\n"
00232         "}\n", "");
00233     auto root = config::parse(in);
00234     REQUIRE(root != nullptr);
00235     REQUIRE_FALSE(root->children.empty());
00236     REQUIRE(root->children[0] != nullptr);
00237     REQUIRE_FALSE(root->children[0]->children.empty());
00238     REQUIRE(root->children[0]->children[0] != nullptr);
00239     auto action =
00240         parse_action(
00241             *root->children[0]->children[0], -1,
00242             If().type("PING").from("192.168").to("127.0/8"), chains);
00243     REQUIRE(
00244         str(*action) ==
00245         "reject if PING from 192.168 to 127.0/8");
00246 }
00247 SECTION("Call action.")
00248 {
00249     tao::pegtl::string_input<> in(
00250         "chain default {\n"
00251         "    call some_chain;\n"
00252         "}\n", "");
00253     auto root = config::parse(in);
00254     REQUIRE(root != nullptr);
00255     REQUIRE_FALSE(root->children.empty());
00256     REQUIRE(root->children[0] != nullptr);
00257     REQUIRE_FALSE(root->children[0]->children.empty());
00258     REQUIRE(root->children[0]->children[0] != nullptr);
00259     auto action = parse_action(
00260         *root->children[0]->children[0], {}, {}, chains);
00261     REQUIRE(str(*action) == "call some_chain");
00262 }
00263 SECTION("Call action, throws and error if calling the default chain.")
00264 {
00265     tao::pegtl::string_input<> in(
00266         "chain some_chain {\n"
00267         "    call default;\n"
00268         "}\n", "");
00269     auto root = config::parse(in);
00270     REQUIRE(root != nullptr);

```

```

00271     REQUIRE_FALSE(root->children.empty());
00272     REQUIRE(root->children[0] != nullptr);
00273     REQUIRE_FALSE(root->children[0]->children.empty());
00274     REQUIRE(root->children[0]->children[0] != nullptr);
00275     REQUIRE_THROWS_AS(
00276         parse_action(*root->children[0]->children[0], {}, {}, chains),
00277         std::invalid_argument);
00278     REQUIRE_THROWS_WITH(
00279         parse_action(*root->children[0]->children[0], {}, {}, chains),
00280         "cannot 'call' the default chain");
00281 }
00282 SECTION("Call action, with priority.")
00283 {
00284     tao::pegtl::string_input<> in(
00285         "chain default {\n"
00286         "    call some_chain with priority -1;\n"
00287         "}\n", "");
00288     auto root = config::parse(in);
00289     REQUIRE(root != nullptr);
00290     REQUIRE_FALSE(root->children.empty());
00291     REQUIRE(root->children[0] != nullptr);
00292     REQUIRE_FALSE(root->children[0]->children.empty());
00293     REQUIRE(root->children[0]->children[0] != nullptr);
00294     auto action = parse_action(
00295         *root->children[0]->children[0], -1, {}, chains);
00296     REQUIRE(str(*action) == "call some_chain with priority -1");
00297 }
00298 SECTION("Call action, with condition.")
00299 {
00300     tao::pegtl::string_input<> in(
00301         "chain default {\n"
00302         "    call some_chain if PING from 192.168 to 127.0/8;\n"
00303         "}\n", "");
00304     auto root = config::parse(in);
00305     REQUIRE(root != nullptr);
00306     REQUIRE_FALSE(root->children.empty());
00307     REQUIRE(root->children[0] != nullptr);
00308     REQUIRE_FALSE(root->children[0]->children.empty());
00309     REQUIRE(root->children[0]->children[0] != nullptr);
00310     auto action =
00311         parse_action(
00312             *root->children[0]->children[0], {},
00313             If().type("PING").from("192.168").to("127.0/8"), chains);
00314     REQUIRE(
00315         str(*action) == "call some_chain if PING from 192.168 to 127.0/8");
00316 }
00317 SECTION("Call action, with priority and condition.")
00318 {
00319     tao::pegtl::string_input<> in(
00320         "chain default {\n"
00321         "    call some_chain with priority -1 "
00322         "if PING from 192.168 to 127.0/8;\n"
00323         "}\n", "");
00324     auto root = config::parse(in);
00325     REQUIRE(root != nullptr);
00326     REQUIRE_FALSE(root->children.empty());
00327     REQUIRE(root->children[0] != nullptr);
00328     REQUIRE_FALSE(root->children[0]->children.empty());
00329     REQUIRE(root->children[0]->children[0] != nullptr);
00330     auto action =
00331         parse_action(
00332             *root->children[0]->children[0], -1,
00333             If().type("PING").from("192.168").to("127.0/8"), chains);
00334     REQUIRE(
00335         str(*action) ==
00336             "call some_chain with priority -1 if PING from 192.168 to 127.0/8");
00337 }
00338 SECTION("GoTo action.")
00339 {
00340     tao::pegtl::string_input<> in(
00341         "chain default {\n"
00342         "    goto some_chain;\n"
00343         "}\n", "");
00344     auto root = config::parse(in);
00345     REQUIRE(root != nullptr);
00346     REQUIRE_FALSE(root->children.empty());
00347     REQUIRE(root->children[0] != nullptr);
00348     REQUIRE_FALSE(root->children[0]->children.empty());
00349     REQUIRE(root->children[0]->children[0] != nullptr);
00350     auto action = parse_action(
00351         *root->children[0]->children[0], {}, {}, chains);

```

```

00352     REQUIRE(str(*action) == "goto some_chain");
00353 }
00354 SECTION("GoTo action, throws an error if going to the default chain.")
00355 {
00356     tao::pegtl::string_input<> in(
00357         "chain some_chain {\n"
00358         "    goto default;\n"
00359         "}\n", "");
00360     auto root = config::parse(in);
00361     REQUIRE(root != nullptr);
00362     REQUIRE_FALSE(root->children.empty());
00363     REQUIRE(root->children[0] != nullptr);
00364     REQUIRE_FALSE(root->children[0]->children.empty());
00365     REQUIRE(root->children[0]->children[0] != nullptr);
00366     REQUIRE_THROWS_AS(
00367         parse_action(*root->children[0]->children[0], {}, {}, chains),
00368         std::invalid_argument);
00369     REQUIRE_THROWS_WITH(
00370         parse_action(*root->children[0]->children[0], {}, {}, chains),
00371         "cannot 'goto' the default chain");
00372 }
00373 SECTION("GoTo action, with priority.")
00374 {
00375     tao::pegtl::string_input<> in(
00376         "chain default {\n"
00377         "    goto some_chain with priority -1;\n"
00378         "}\n", "");
00379     auto root = config::parse(in);
00380     REQUIRE(root != nullptr);
00381     REQUIRE_FALSE(root->children.empty());
00382     REQUIRE(root->children[0] != nullptr);
00383     REQUIRE_FALSE(root->children[0]->children.empty());
00384     REQUIRE(root->children[0]->children[0] != nullptr);
00385     auto action =
00386         parse_action(*root->children[0]->children[0], -1, {}, chains);
00387     REQUIRE(str(*action) == "goto some_chain with priority -1");
00388 }
00389 SECTION("GoTo action, with condition.")
00390 {
00391     tao::pegtl::string_input<> in(
00392         "chain default {\n"
00393         "    goto some_chain if PING from 192.168 to 127.0/8;\n"
00394         "}\n", "");
00395     auto root = config::parse(in);
00396     REQUIRE(root != nullptr);
00397     REQUIRE_FALSE(root->children.empty());
00398     REQUIRE(root->children[0] != nullptr);
00399     REQUIRE_FALSE(root->children[0]->children.empty());
00400     REQUIRE(root->children[0]->children[0] != nullptr);
00401     auto action =
00402         parse_action(
00403             *root->children[0]->children[0], {},
00404             If().type("PING").from("192.168").to("127.0/8"), chains);
00405     REQUIRE(
00406         str(*action) == "goto some_chain if PING from 192.168 to 127.0/8");
00407 }
00408 SECTION("GoTo action, with priority and condition.")
00409 {
00410     tao::pegtl::string_input<> in(
00411         "chain default {\n"
00412         "    goto some_chain with priority -1 "
00413         "if PING from 192.168 to 127.0/8;\n"
00414         "}\n", "");
00415     auto root = config::parse(in);
00416     REQUIRE(root != nullptr);
00417     REQUIRE_FALSE(root->children.empty());
00418     REQUIRE(root->children[0] != nullptr);
00419     REQUIRE_FALSE(root->children[0]->children.empty());
00420     REQUIRE(root->children[0]->children[0] != nullptr);
00421     auto action =
00422         parse_action(
00423             *root->children[0]->children[0], -1,
00424             If().type("PING").from("192.168").to("127.0/8"), chains);
00425     REQUIRE(
00426         str(*action) ==
00427         "goto some_chain with priority -1 if PING from 192.168 to 127.0/8");
00428 }
00429 }
00430
00431
00432 TEST_CASE("'parse_condition' parses a condition from the given AST node.",
```

```

00433     "[ConfigParser]")
00434 {
00435     SECTION("Condition with type.")
00436 {
00437     tao::pegtl::string_input<> in(
00438         "chain default {\n"
00439         "    accept if PING;\n"
00440         "}\n", "");
00441     auto root = config::parse(in);
00442     REQUIRE(root != nullptr);
00443     REQUIRE_FALSE(root->children.empty());
00444     REQUIRE(root->children[0] != nullptr);
00445     REQUIRE_FALSE(root->children[0]->children.empty());
00446     REQUIRE(root->children[0]->children[0] != nullptr);
00447     REQUIRE_FALSE(root->children[0]->children[0]->children.empty());
00448     REQUIRE(root->children[0]->children[0]->children[0] != nullptr);
00449     auto condition = parse_condition(
00450             *root->children[0]->children[0]->children[0]);
00451     REQUIRE(str(condition) == "if PING");
00452 }
00453 SECTION("Condition source.")
00454 {
00455     tao::pegtl::string_input<> in(
00456         "chain default {\n"
00457         "    accept if from 192.168;\n"
00458         "}\n", "");
00459     auto root = config::parse(in);
00460     REQUIRE(root != nullptr);
00461     REQUIRE_FALSE(root->children.empty());
00462     REQUIRE(root->children[0] != nullptr);
00463     REQUIRE_FALSE(root->children[0]->children.empty());
00464     REQUIRE(root->children[0]->children[0] != nullptr);
00465     REQUIRE_FALSE(root->children[0]->children[0]->children.empty());
00466     REQUIRE(root->children[0]->children[0]->children[0] != nullptr);
00467     auto condition = parse_condition(
00468             *root->children[0]->children[0]->children[0]);
00469     REQUIRE(str(condition) == "if from 192.168");
00470 }
00471 SECTION("Condition destination.")
00472 {
00473     tao::pegtl::string_input<> in(
00474         "chain default {\n"
00475         "    accept if to 128.0/8;\n"
00476         "}\n", "");
00477     auto root = config::parse(in);
00478     REQUIRE(root != nullptr);
00479     REQUIRE_FALSE(root->children.empty());
00480     REQUIRE(root->children[0] != nullptr);
00481     REQUIRE_FALSE(root->children[0]->children.empty());
00482     REQUIRE(root->children[0]->children[0] != nullptr);
00483     REQUIRE_FALSE(root->children[0]->children[0]->children.empty());
00484     REQUIRE(root->children[0]->children[0]->children[0] != nullptr);
00485     auto condition = parse_condition(
00486             *root->children[0]->children[0]->children[0]);
00487     REQUIRE(str(condition) == "if to 128.0/8");
00488 }
00489 SECTION("Condition source and destination.")
00490 {
00491     tao::pegtl::string_input<> in(
00492         "chain default {\n"
00493         "    accept if from 192.168 to 127.0/8;\n"
00494         "}\n", "");
00495     auto root = config::parse(in);
00496     REQUIRE(root != nullptr);
00497     REQUIRE_FALSE(root->children.empty());
00498     REQUIRE(root->children[0] != nullptr);
00499     REQUIRE_FALSE(root->children[0]->children.empty());
00500     REQUIRE(root->children[0]->children[0] != nullptr);
00501     REQUIRE_FALSE(root->children[0]->children[0]->children.empty());
00502     REQUIRE(root->children[0]->children[0]->children[0] != nullptr);
00503     auto condition = parse_condition(
00504             *root->children[0]->children[0]->children[0]);
00505     REQUIRE(str(condition) == "if from 192.168 to 127.0/8");
00506 }
00507 SECTION("Condition type and source.")
00508 {
00509     tao::pegtl::string_input<> in(
00510         "chain default {\n"
00511         "    accept if PING from 192.168;\n"
00512         "}\n", "");
00513     auto root = config::parse(in);

```

```

00514     REQUIRE(root != nullptr);
00515     REQUIRE_FALSE(root->children.empty());
00516     REQUIRE(root->children[0] != nullptr);
00517     REQUIRE_FALSE(root->children[0]->children.empty());
00518     REQUIRE(root->children[0]->children[0] != nullptr);
00519     REQUIRE_FALSE(root->children[0]->children[0]->children.empty());
00520     REQUIRE(root->children[0]->children[0]->children[0] != nullptr);
00521     auto condition = parse_condition(
00522         *root->children[0]->children[0]->children[0]);
00523     REQUIRE(str(condition) == "if PING from 192.168");
00524 }
00525 SECTION("Condition type and destination.")
00526 {
00527     tao::pegtl::string_input<> in(
00528         "chain default {\n"
00529             "    accept if PING to 127.0/8;\n"
00530         "}\n", "");
00531     auto root = config::parse(in);
00532     REQUIRE(root != nullptr);
00533     REQUIRE_FALSE(root->children.empty());
00534     REQUIRE(root->children[0] != nullptr);
00535     REQUIRE_FALSE(root->children[0]->children.empty());
00536     REQUIRE(root->children[0]->children[0] != nullptr);
00537     REQUIRE_FALSE(root->children[0]->children[0]->children.empty());
00538     REQUIRE(root->children[0]->children[0]->children[0] != nullptr);
00539     auto condition = parse_condition(
00540         *root->children[0]->children[0]->children[0]);
00541     REQUIRE(str(condition) == "if PING to 127.0/8");
00542 }
00543 SECTION("Condition type, source, and destination.")
00544 {
00545     tao::pegtl::string_input<> in(
00546         "chain default {\n"
00547             "    accept if PING from 192.168 to 127.0/8;\n"
00548         "}\n", "");
00549     auto root = config::parse(in);
00550     REQUIRE(root != nullptr);
00551     REQUIRE_FALSE(root->children.empty());
00552     REQUIRE(root->children[0] != nullptr);
00553     REQUIRE_FALSE(root->children[0]->children.empty());
00554     REQUIRE(root->children[0]->children[0] != nullptr);
00555     REQUIRE_FALSE(root->children[0]->children[0]->children.empty());
00556     REQUIRE(root->children[0]->children[0]->children[0] != nullptr);
00557     auto condition = parse_condition(
00558         *root->children[0]->children[0]->children[0]);
00559     REQUIRE(str(condition) == "if PING from 192.168 to 127.0/8");
00560 }
00561 }
00562
00563
00564 TEST_CASE("'parse_chain' parses an action from the given AST node.",
00565     "[ConfigParser]")
00566 {
00567     Chain default_chain("default");
00568     std::map<std::string, std::shared_ptr<Chain>> chains;
00569     chains["some_chain"] = std::make_shared<Chain>("some_chain");
00570 SECTION("Accept action.")
00571 {
00572     tao::pegtl::string_input<> in(
00573         "chain default {\n"
00574             "    accept;\n"
00575             "    accept with priority -1;\n"
00576             "    accept if PING from 192.168 to 127.0/8;\n"
00577             "    accept with priority -1 if PING from 192.168 to 127.0/8;\n"
00578         "}\n", "");
00579     auto root = config::parse(in);
00580     REQUIRE(root != nullptr);
00581     REQUIRE_FALSE(root->children.empty());
00582     REQUIRE(root->children[0] != nullptr);
00583     parse_chain(default_chain, *root->children[0], chains);
00584     REQUIRE(str(default_chain) ==
00585         "chain default {\n"
00586             "    accept;\n"
00587             "    accept with priority -1;\n"
00588             "    accept if PING from 192.168 to 127.0/8;\n"
00589             "    accept with priority -1 if PING from 192.168 to 127.0/8;\n"
00590         "}");
00591 }
00592 SECTION("Reject action.")
00593 {
00594     tao::pegtl::string_input<> in(

```

```

00595         "chain default {\n"
00596             "    reject;\n"
00597             "    reject if PING from 192.168 to 127.0/8;\n"
00598             "}\n", "");
00599     auto root = config::parse(in);
00600     REQUIRE(root != nullptr);
00601     REQUIRE_FALSE(root->children.empty());
00602     REQUIRE(root->children[0] != nullptr);
00603     parse_chain(default_chain, *root->children[0], chains);
00604     REQUIRE(str(default_chain) ==
00605             "chain default {\n"
00606                 "    reject;\n"
00607                 "    reject if PING from 192.168 to 127.0/8;\n"
00608                 "};");
00609 }
00610 SECTION("Call action.")
00611 {
00612     tao::pegtl::string_input<> in(
00613         "chain default {\n"
00614             "    call some_chain;\n"
00615             "    call some_chain with priority -1;\n"
00616             "    call some_chain if PING from 192.168 to 127.0/8;\n"
00617             "    call some_chain with priority -1"
00618             "if PING from 192.168 to 127.0/8;\n"
00619             "}\n", "");
00620     auto root = config::parse(in);
00621     REQUIRE(root != nullptr);
00622     REQUIRE_FALSE(root->children.empty());
00623     REQUIRE(root->children[0] != nullptr);
00624     parse_chain(default_chain, *root->children[0], chains);
00625     REQUIRE(str(default_chain) ==
00626             "chain default {\n"
00627                 "    call some_chain;\n"
00628                     "    call some_chain with priority -1;\n"
00629                     "    call some_chain if PING from 192.168 to 127.0/8;\n"
00630                     "    call some_chain with priority -1 "
00631                     "if PING from 192.168 to 127.0/8;\n"
00632                     "};");
00633 }
00634 SECTION("GoTo action.")
00635 {
00636     tao::pegtl::string_input<> in(
00637         "chain default {\n"
00638             "    goto some_chain;\n"
00639             "    goto some_chain with priority -1;\n"
00640             "    goto some_chain if PING from 192.168 to 127.0/8;\n"
00641             "    goto some_chain with priority -1"
00642             "if PING from 192.168 to 127.0/8;\n"
00643             "}\n", "");
00644     auto root = config::parse(in);
00645     REQUIRE(root != nullptr);
00646     REQUIRE_FALSE(root->children.empty());
00647     REQUIRE(root->children[0] != nullptr);
00648     parse_chain(default_chain, *root->children[0], chains);
00649     REQUIRE(str(default_chain) ==
00650             "chain default {\n"
00651                 "    goto some_chain;\n"
00652                     "    goto some_chain with priority -1;\n"
00653                     "    goto some_chain if PING from 192.168 to 127.0/8;\n"
00654                     "    goto some_chain with priority -1 "
00655                     "if PING from 192.168 to 127.0/8;\n"
00656                     "};");
00657 }
00658 }
00659
00660
00661 TEST_CASE("'parse_filter' parses the filter from the given AST root node.", [
00662     "[ConfigParser]")
00663 {
00664     auto ping = packet_v2::Packet(to_vector(PingV2()));
00665     auto heartbeat = packet_v2::Packet(to_vector(HeartbeatV2()));
00666     auto encapsulated_data =
00667         packet_v2::Packet(to_vector(EncapsulatedDataV2()));
00668     SECTION("Successful parse.")
00669 {
00670     tao::pegtl::string_input<> in(
00671         "chain default {\n"
00672             "    call first_chain if PING;\n"
00673             "    goto second_chain if HEARTBEAT;\n"
00674             "    reject;\n"
00675             "}\n"

```

```

00676      "\n"
00677      "chain first_chain {\n"
00678          "    accept with priority 1;\n"
00679      "}\n"
00680      "\n"
00681      "chain second_chain {\n"
00682          "    accept with priority 2;\n"
00683      "}\n", "");
00684      auto root = config::parse(in);
00685      REQUIRE(root != nullptr);
00686      auto filter = parse_filter(*root);
00687      REQUIRE(filter != nullptr);
00688      auto ping_result = filter->will_accept(ping, MAVAddress("127.1"));
00689      REQUIRE(ping_result.first == true);
00690      REQUIRE(ping_result.second == 1);
00691      auto heartbeat_result =
00692          filter->will_accept(heartbeat, MAVAddress("127.1"));
00693      REQUIRE(heartbeat_result.first == true);
00694      REQUIRE(heartbeat_result.second == 2);
00695      auto encapsulated_result =
00696          filter->will_accept(encapsulated_data, MAVAddress("127.1"));
00697      REQUIRE(encapsulated_result.first == false);
00698  }
00699 SECTION("Default filter action is 'accept'")
00700 {
00701     tao::pegtl::string_input<> in(
00702         "default_action accept;\n"
00703         "chain default {\n"
00704             "}\n", "");
00705     auto root = config::parse(in);
00706     REQUIRE(root != nullptr);
00707     auto filter = parse_filter(*root);
00708     REQUIRE(filter != nullptr);
00709     auto result = filter->will_accept(ping, MAVAddress("127.1"));
00710     REQUIRE(result.first == true);
00711     REQUIRE(result.second == 0);
00712 }
00713 SECTION("Default filter action is 'reject'")
00714 {
00715     tao::pegtl::string_input<> in(
00716         "default_action reject;\n"
00717         "chain default {\n"
00718             "}\n", "");
00719     auto root = config::parse(in);
00720     REQUIRE(root != nullptr);
00721     auto filter = parse_filter(*root);
00722     REQUIRE(filter != nullptr);
00723     auto result = filter->will_accept(ping, MAVAddress("127.1"));
00724     REQUIRE(result.first == false);
00725 }
00726 SECTION("The default, default filter action is 'reject'")
00727 {
00728     tao::pegtl::string_input<> in(
00729         "chain default {\n"
00730             "}\n", "");
00731     auto root = config::parse(in);
00732     REQUIRE(root != nullptr);
00733     auto filter = parse_filter(*root);
00734     REQUIRE(filter != nullptr);
00735     auto result = filter->will_accept(ping, MAVAddress("127.1"));
00736     REQUIRE(result.first == false);
00737 }
00738 }
00739
00740
00741 TEST_CASE("'parse_serial' parses a serial interface from a serial interface "
00742             "AST node.", "[ConfigParser]")
00743 {
00744     SECTION("With flow control.")
00745     {
00746         tao::pegtl::string_input<> in(
00747             "serial {\n"
00748                 "    device ./ttyS0;\n"
00749                 "    baudrate 115200;\n"
00750                 "    flow_control yes;\n"
00751             "}\n", "");
00752         auto root = config::parse(in);
00753         REQUIRE(root != nullptr);
00754         REQUIRE_FALSE(root->children.empty());
00755         REQUIRE(root->children[0] != nullptr);
00756         auto filter = std::make_shared<Filter>(Chain("default"));

```

```

00757     auto connection_pool = std::make_shared<ConnectionPool>();
00758     auto serial_port =
00759         parse_serial(*root->children[0], filter, connection_pool);
00760     REQUIRE(
00761         str(*serial_port) ==
00762             "serial {\n"
00763                 "device ./ttyS0;\n"
00764                 "baudrate 115200;\n"
00765                 "flow_control yes;\n"
00766             "}\n");
00767 }
00768 SECTION("Without flow control.")
00769 {
00770     tao::pegtl::string_input<> in(
00771         "serial {\n"
00772             "device ./ttyS0;\n"
00773                 "baudrate 115200;\n"
00774                 "flow_control no;\n"
00775             "}\n", "");
00776     auto root = config::parse(in);
00777     REQUIRE(root != nullptr);
00778     REQUIRE_FALSE(root->children.empty());
00779     REQUIRE(root->children[0] != nullptr);
00780     auto filter = std::make_shared<Filter>(Chain("default"));
00781     auto connection_pool = std::make_shared<ConnectionPool>();
00782     auto serial_port =
00783         parse_serial(*root->children[0], filter, connection_pool);
00784     REQUIRE(
00785         str(*serial_port) ==
00786             "serial {\n"
00787                 "device ./ttyS0;\n"
00788                 "baudrate 115200;\n"
00789                 "flow_control no;\n"
00790             "}\n");
00791 }
00792 SECTION("Default flow control is off.")
00793 {
00794     tao::pegtl::string_input<> in(
00795         "serial {\n"
00796             "device ./ttyS0;\n"
00797                 "baudrate 115200;\n"
00798             "}\n", "");
00799     auto root = config::parse(in);
00800     REQUIRE(root != nullptr);
00801     REQUIRE_FALSE(root->children.empty());
00802     REQUIRE(root->children[0] != nullptr);
00803     auto filter = std::make_shared<Filter>(Chain("default"));
00804     auto connection_pool = std::make_shared<ConnectionPool>();
00805     auto serial_port =
00806         parse_serial(*root->children[0], filter, connection_pool);
00807     REQUIRE(
00808         str(*serial_port) ==
00809             "serial {\n"
00810                 "device ./ttyS0;\n"
00811                 "baudrate 115200;\n"
00812                 "flow_control no;\n"
00813             "}\n");
00814 }
00815 SECTION("Default baud rate is 9600.")
00816 {
00817     tao::pegtl::string_input<> in(
00818         "serial {\n"
00819             "device ./ttyS0;\n"
00820             "}\n", "");
00821     auto root = config::parse(in);
00822     REQUIRE(root != nullptr);
00823     REQUIRE_FALSE(root->children.empty());
00824     REQUIRE(root->children[0] != nullptr);
00825     auto filter = std::make_shared<Filter>(Chain("default"));
00826     auto connection_pool = std::make_shared<ConnectionPool>();
00827     auto serial_port =
00828         parse_serial(*root->children[0], filter, connection_pool);
00829     REQUIRE(
00830         str(*serial_port) ==
00831             "serial {\n"
00832                 "device ./ttyS0;\n"
00833                 "baudrate 9600;\n"
00834                 "flow_control no;\n"
00835             "}\n");
00836 }
00837 SECTION("Throw error if device string is missing.")

```

```

00838     {
00839         tao::pegtl::string_input<> in(
00840             "serial {\n"
00841             "\n", "");
00842         auto root = config::parse(in);
00843         REQUIRE(root != nullptr);
00844         REQUIRE_FALSE(root->children.empty());
00845         REQUIRE(root->children[0] != nullptr);
00846         auto filter = std::make_shared<Filter>(Chain("default"));
00847         auto connection_pool = std::make_shared<ConnectionPool>();
00848         REQUIRE_THROWS_AS(
00849             parse_serial(*root->children[0], filter, connection_pool),
00850             std::invalid_argument);
00851         REQUIRE_THROWS_WITH(
00852             parse_serial(*root->children[0], filter, connection_pool),
00853             "missing device string");
00854     }
00855 SECTION("With flow control.")
00856 {
00857     auto ping = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00858     auto set_mode = std::make_shared<packet_v2::Packet>(
00859         to_vector(SetModeV2()));
00860     auto param_ext_request_list =
00861         std::make_shared<packet_v2::Packet>(
00862             to_vector(ParamExtRequestListV2()));
00863     fakeit::Mock<ConnectionPool> mock_pool;
00864     std::shared_ptr<Connection> connection;
00865     fakeit::When(Method(mock_pool, add)).AlwaysDo([&](auto conn)
00866     {
00867         connection = conn.lock();
00868     });
00869     tao::pegtl::string_input<> in(
00870         "serial {\n"
00871         "\n    device ./ttyS0;\n"
00872         "\n    preload 127.1;\n"
00873         "\n    preload 32.64;\n"
00874         "\n", "");
00875     auto root = config::parse(in);
00876     REQUIRE(root != nullptr);
00877     REQUIRE_FALSE(root->children.empty());
00878     REQUIRE(root->children[0] != nullptr);
00879     auto filter = std::make_shared<Filter>(Chain("default"), true);
00880     auto connection_pool = mock_shared(mock_pool);
00881     auto serial_port =
00882         parse_serial(*root->children[0], filter, connection_pool);
00883     REQUIRE(serial_port != nullptr);
00884     fakeit::Verify(Method(mock_pool, add)).Exactly(1);
00885     connection->send(ping);
00886     connection->send(set_mode);
00887     connection->send(param_ext_request_list);
00888     // ping packet
00889     auto packet = connection->next_packet();
00890     REQUIRE(packet != nullptr);
00891     REQUIRE(*packet == *ping);
00892     // set_mode packet
00893     packet = connection->next_packet();
00894     REQUIRE(packet != nullptr);
00895     REQUIRE(*packet == *param_ext_request_list);
00896     // no more packets
00897     packet = connection->next_packet();
00898     REQUIRE(packet == nullptr);
00899 }
00900 }
00901
00902
00903 TEST_CASE("'parse_udp' parses a UDP interface from a UDP interface AST node.",
00904     "[ConfigParser]")
00905 {
00906     SECTION("Without specific IP address.")
00907     {
00908         tao::pegtl::string_input<> in(
00909             "udp {\n"
00910             "\n    port 14500;\n"
00911             "\n", "");
00912         auto root = config::parse(in);
00913         REQUIRE(root != nullptr);
00914         REQUIRE_FALSE(root->children.empty());
00915         REQUIRE(root->children[0] != nullptr);
00916         auto filter = std::make_shared<Filter>(Chain("default"));
00917         auto connection_pool = std::make_shared<ConnectionPool>();
00918         auto udp_socket =

```

```

00919         parse_udp(*root->children[0], filter, connection_pool);
00920     REQUIRE(
00921         str(*udp_socket) ==
00922             "udp {\n"
00923                 "    port 14500;\n"
00924             "}\n");
00925 }
00926 SECTION("With specific IP address.")
00927 {
00928     tao::pegtl::string_input<> in(
00929         "udp {\n"
00930             "    port 14500;\n"
00931                 "    address 127.0.0.1;\n"
00932             "}\n", "");
00933     auto root = config::parse(in);
00934     REQUIRE(root != nullptr);
00935     REQUIRE_FALSE(root->children.empty());
00936     REQUIRE(root->children[0] != nullptr);
00937     auto filter = std::make_shared<Filter>(Chain("default"));
00938     auto connection_pool = std::make_shared<ConnectionPool>();
00939     auto udp_socket =
00940         parse_udp(*root->children[0], filter, connection_pool);
00941     REQUIRE(
00942         str(*udp_socket) ==
00943             "udp {\n"
00944                 "    port 14500;\n"
00945                     "    address 127.0.0.1;\n"
00946             "}\n");
00947 }
00948 SECTION("Without specific IP address (with bitrate limit).")
00949 {
00950     tao::pegtl::string_input<> in(
00951         "udp {\n"
00952             "    port 14500;\n"
00953                 "    max_bitrate 8192;\n"
00954             "}\n", "");
00955     auto root = config::parse(in);
00956     REQUIRE(root != nullptr);
00957     REQUIRE_FALSE(root->children.empty());
00958     REQUIRE(root->children[0] != nullptr);
00959     auto filter = std::make_shared<Filter>(Chain("default"));
00960     auto connection_pool = std::make_shared<ConnectionPool>();
00961     auto udp_socket =
00962         parse_udp(*root->children[0], filter, connection_pool);
00963     REQUIRE(
00964         str(*udp_socket) ==
00965             "udp {\n"
00966                 "    port 14500;\n"
00967                     "    max_bitrate 8192;\n"
00968             "}\n");
00969 }
00970 SECTION("With specific IP address (with bitrate limit).")
00971 {
00972     tao::pegtl::string_input<> in(
00973         "udp {\n"
00974             "    port 14500;\n"
00975                 "    address 127.0.0.1;\n"
00976                     "    max_bitrate 8192;\n"
00977             "}\n", "");
00978     auto root = config::parse(in);
00979     REQUIRE(root != nullptr);
00980     REQUIRE_FALSE(root->children.empty());
00981     REQUIRE(root->children[0] != nullptr);
00982     auto filter = std::make_shared<Filter>(Chain("default"));
00983     auto connection_pool = std::make_shared<ConnectionPool>();
00984     auto udp_socket =
00985         parse_udp(*root->children[0], filter, connection_pool);
00986     REQUIRE(
00987         str(*udp_socket) ==
00988             "udp {\n"
00989                 "    port 14500;\n"
00990                     "    address 127.0.0.1;\n"
00991                         "    max_bitrate 8192;\n"
00992             "}\n");
00993 }
00994 }
00995
00996
00997 TEST_CASE("'"parse_interfaces' parses serial port and UDP interfaces from "
00998         "the root node.", "[ConfigParser]")
00999 {

```

```
01000     tao::pegtl::string_input<> in(
01001         "serial {\n"
01002             "    device ./ttyS0;\n"
01003             "    baudrate 115200;\n"
01004             "    flow_control yes;\n"
01005         "}\n"
01006         "\n"
01007         "serial {\n"
01008             "    device ./ttyS1;\n"
01009             "    baudrate 300;\n"
01010             "    flow_control no;\n"
01011         "}\n"
01012         "\n"
01013         "udp {\n"
01014             "    port 14500;\n"
01015         "}\n"
01016         "\n"
01017         "udp {\n"
01018             "    port 14501;\n"
01019             "    address 127.0.0.1;\n"
01020         "}\n"
01021         "\n"
01022         "udp {\n"
01023             "    port 14502;\n"
01024             "    max_bitrate 8192;\n"
01025         "}\n"
01026         "\n"
01027         "udp {\n"
01028             "    port 14503;\n"
01029             "    address 127.0.0.1;\n"
01030             "    max_bitrate 8192;\n"
01031         "}\n", "");
01032     auto root = config::parse(in);
01033     REQUIRE(root != nullptr);
01034     auto filter = std::make_unique<Filter>(Chain("default"));
01035     auto interfaces = parse_interfaces(*root, std::move(filter));
01036     REQUIRE(interfaces.size() == 6);
01037     REQUIRE(interfaces[0] != nullptr);
01038     REQUIRE(
01039         str(*interfaces[0]) ==
01040             "serial {\n"
01041                 "    device ./ttyS0;\n"
01042                 "    baudrate 115200;\n"
01043                 "    flow_control yes;\n"
01044             "}");
01045     REQUIRE(interfaces[1] != nullptr);
01046     REQUIRE(
01047         str(*interfaces[1]) ==
01048             "serial {\n"
01049                 "    device ./ttyS1;\n"
01050                 "    baudrate 300;\n"
01051                 "    flow_control no;\n"
01052             "}");
01053     REQUIRE(interfaces[2] != nullptr);
01054     REQUIRE(
01055         str(*interfaces[2]) ==
01056             "udp {\n"
01057                 "    port 14500;\n"
01058             "}");
01059     REQUIRE(interfaces[3] != nullptr);
01060     REQUIRE(
01061         str(*interfaces[3]) ==
01062             "udp {\n"
01063                 "    port 14501;\n"
01064                 "    address 127.0.0.1;\n"
01065             "}");
01066     REQUIRE(interfaces[4] != nullptr);
01067     REQUIRE(
01068         str(*interfaces[4]) ==
01069             "udp {\n"
01070                 "    port 14502;\n"
01071                 "    max_bitrate 8192;\n"
01072             "}");
01073     REQUIRE(interfaces[5] != nullptr);
01074     REQUIRE(
01075         str(*interfaces[5]) ==
01076             "udp {\n"
01077                 "    port 14503;\n"
01078                 "    address 127.0.0.1;\n"
01079                 "    max_bitrate 8192;\n"
01080             "});
```

```

01081 }
01082
01083
01084 TEST_CASE("ConfigParser can parse a file.", "[ConfigParser]")
01085 {
01086     SECTION("When the file exists and is valid.")
01087     {
01088         REQUIRE_NO_THROW(ConfigParser("test/mavtables.conf"));
01089     }
01090     SECTION("Throws an error if the file does not exist.")
01091     {
01092         REQUIRE_THROWS(
01093             ConfigParser("file_that_does_not_exist.conf"));
01094     }
01095 }
01096
01097
01098 TEST_CASE("ConfigParser are printable.", "[ConfigParser]")
01099 {
01100     ConfigParser config("test/mavtables.conf");
01101     REQUIRE(
01102         str(config) ==
01103         ":001: default_action\n"
01104         ":001: | accept\n"
01105         ":004: udp\n"
01106         ":005: | port 14500\n"
01107         ":006: | address 127.0.0.1\n"
01108         ":007: | max_bitrate 8388608\n"
01109         ":011: serial\n"
01110         ":012: | device ./ttyp0\n"
01111         ":013: | baudrate 115200\n"
01112         ":014: | flow_control yes\n"
01113         ":015: | preload 1.1\n"
01114         ":016: | preload 62.34\n"
01115         ":020: chain default\n"
01116         ":022: | call some_chain10\n"
01117         ":022: | | condition\n"
01118         ":022: | | | source 127.1\n"
01119         ":022: | | | dest 192.0\n"
01120         ":023: | reject\n"
01121         ":027: chain some_chain10\n"
01122         ":029: | accept\n"
01123         ":029: | | priority 99\n"
01124         ":029: | | condition\n"
01125         ":029: | | | dest 192.0\n"
01126         ":030: | accept\n"
01127         ":030: | | condition\n"
01128         ":030: | | | packet_type PING\n"
01129         ":031: | accept\n";
01130 }
01131
01132
01133 TEST_CASE("ConfigParser's 'make_app' method returns an application object.",
01134     "[ConfigParser]")
01135 {
01136     ConfigParser config("test/mavtables.conf");
01137     std::unique_ptr<App> app;
01138     REQUIRE_NO_THROW(app = config.make_app());
01139     REQUIRE(app != nullptr);
01140 }

```

16.203 test_Connection.cpp File Reference

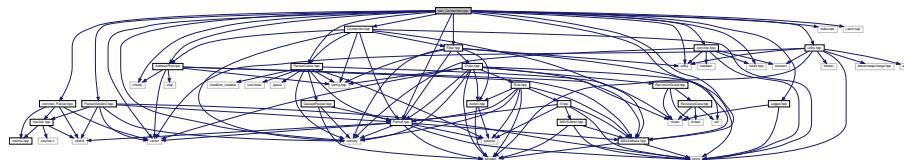
```

#include <chrono>
#include <memory>
#include <set>
#include <stdexcept>
#include <utility>
#include <vector>
#include <catch.hpp>
#include <fakeit.hpp>

```

```
#include "AddressPool.hpp"
#include "Connection.hpp"
#include "Filter.hpp"
#include "Logger.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "PacketQueue.hpp"
#include "PacketVersion2.hpp"
#include "utility.hpp"
#include "common.hpp"
#include "common_Packet.hpp"

Include dependency graph for test_Connection.cpp:
```



Functions

- [TEST_CASE \("Connection's can be constructed.", "\[Connection\]"\)](#)
- [TEST_CASE \("Connection's are printable", "\[Connection\]"\)](#)
- [TEST_CASE \("Connection's 'add_address' method adds/updates addresses.", "\[Connection\]"\)](#)
- [TEST_CASE \("Connection's 'next_packet' method.", "\[Connection\]"\)](#)
- [TEST_CASE \("Connection's 'send' method ensures the given packet is not "nullptr.", "\[Connection\]"\)](#)
- [TEST_CASE \("Connection's 'send' method \(with destination address\).", "\[Connection\]"\)](#)
- [TEST_CASE \("Connection's 'send' method \(without destination address\).", "\[Connection\]"\)](#)
- [TEST_CASE \("Connection's 'send' method \(with broadcast address 0.0\).", "\[Connection\]"\)](#)
- [TEST_CASE \("Connection's 'send' method \(with component broadcast address x.0\).", "\[Connection\]"\)](#)
- [TEST_CASE \("Connection's 'send' method \(destination address, system reachable, " "component unreachable.", "\[Connection\]"\)](#)

16.203.1 Function Documentation

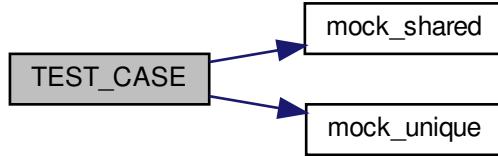
16.203.1.1 TEST_CASE() [1/10]

```
TEST_CASE (
    "Connection's can be constructed." ,
    "" [Connection] )
```

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

References [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



16.203.1.2 TEST_CASE() [2/10]

```
TEST_CASE (
    "Connection's are printable" ,
    "" [Connection] )
```

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

References [mock_shared\(\)](#).

Here is the call graph for this function:



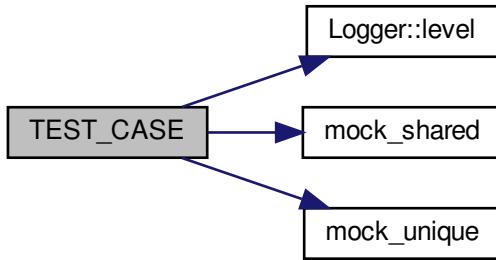
16.203.1.3 TEST_CASE() [3/10]

```
TEST_CASE (
    "Connection's 'add_address' method adds/updates addresses." ,
    "" [Connection] )
```

Definition at line 119 of file [test_Connection.cpp](#).

References [Logger::level\(\)](#), [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



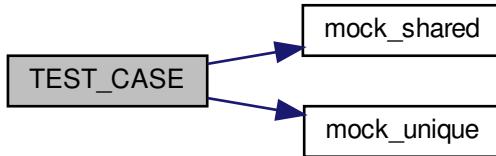
16.203.1.4 TEST_CASE() [4/10]

```
TEST_CASE (
    "Connection's 'next_packet' method." ,
    "" [Connection] )
```

Definition at line 183 of file [test_Connection.cpp](#).

References [mock_shared\(\)](#), [mock_unique\(\)](#), and [ping](#).

Here is the call graph for this function:



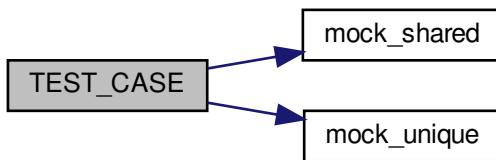
16.203.1.5 TEST_CASE() [5/10]

```
TEST_CASE (
    "Connection's 'send' method ensures the given packet is not \"nullptr.\" ,
    "" [Connection] )
```

Definition at line 234 of file [test_Connection.cpp](#).

References [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



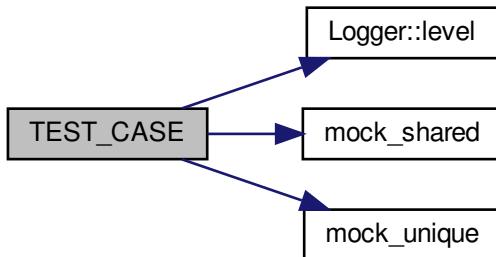
16.203.1.6 TEST_CASE() [6/10]

```
TEST_CASE (
    "Connection's 'send' method (with destination address).",
    "" [Connection] )
```

Definition at line 253 of file [test_Connection.cpp](#).

References [Logger::level\(\)](#), [mock_shared\(\)](#), [mock_unique\(\)](#), and [ping](#).

Here is the call graph for this function:



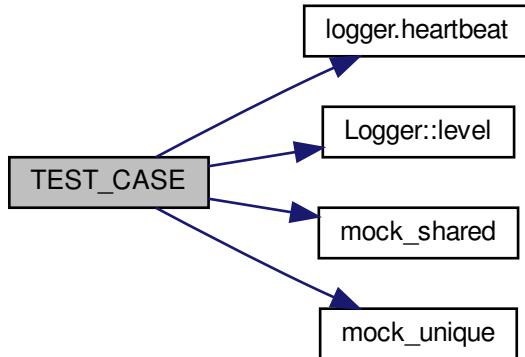
16.203.1.7 TEST_CASE() [7/10]

```
TEST_CASE (
    "Connection's 'send' method (without destination address).",
    "" [Connection] )
```

Definition at line 414 of file [test_Connection.cpp](#).

References [logger::heartbeat\(\)](#), [Logger::level\(\)](#), [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



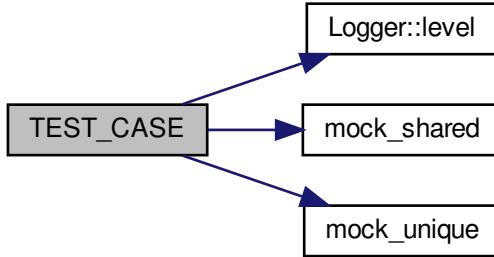
16.203.1.8 TEST_CASE() [8/10]

```
TEST_CASE (
    "Connection's 'send' method (with broadcast address 0.0).",
    "" [Connection] )
```

Definition at line 627 of file [test_Connection.cpp](#).

References [Logger::level\(\)](#), [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



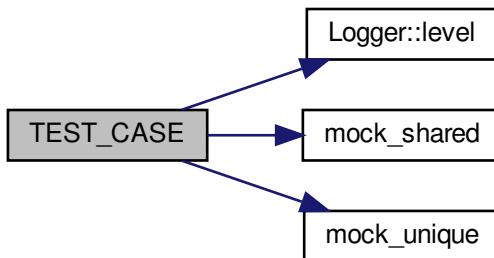
16.203.1.9 TEST_CASE() [9/10]

```
TEST_CASE (
    "Connection's 'send' method (with component broadcast address x.0).",
    "" [Connection] )
```

Definition at line 840 of file [test_Connection.cpp](#).

References [Logger::level\(\)](#), [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



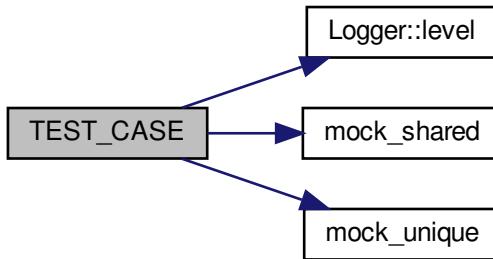
16.203.1.10 TEST_CASE() [10/10]

```
TEST_CASE ( )
```

Definition at line 1110 of file [test_Connection.cpp](#).

References [Logger::level\(\)](#), [mock_shared\(\)](#), [mock_unique\(\)](#), and [ping](#).

Here is the call graph for this function:



16.204 test_Connection.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <chrono>
00019 #include <memory>
00020 #include <set>
00021 #include <stdexcept>
00022 #include <utility>
00023 #include <vector>
00024
00025 #include <catch.hpp>
00026 #include <fakeit.hpp>
00027
00028 #include "AddressPool.hpp"
00029 #include "Connection.hpp"
00030 #include "Filter.hpp"
00031 #include "Logger.hpp"
00032 #include "MAVAddress.hpp"
00033 #include "Packet.hpp"
00034 #include "PacketQueue.hpp"
```

```

00035 #include "PacketVersion2.hpp"
00036 #include "utility.hpp"
00037
00038 #include "common.hpp"
00039 #include "common_Packet.hpp"
00040
00041
00042 using namespace std::chrono_literals;
00043
00044
00045 // NOTE: This testing file is not the best at not testing the implementation.
00046 //       If a better solution is found it should be rewritten.
00047
00048
00049 TEST_CASE("Connection's can be constructed.", "[Connection]")
00050 {
00051     fakeit::Mock<Filter> mock_filter;
00052     fakeit::Mock<AddressPool> mock_pool;
00053     fakeit::Mock<PacketQueue> mock_queue;
00054     auto filter = mock_shared(mock_filter);
00055     auto pool = mock_unique(mock_pool);
00056     auto queue = mock_unique(mock_queue);
00057     SECTION("With default arguments.")
00058     {
00059         REQUIRE_NO_THROW(Connection("name", filter));
00060     }
00061     SECTION("As a regular connection.")
00062     {
00063         REQUIRE_NO_THROW(
00064             Connection(
00065                 "name", filter, false, std::move(pool), std::move(queue)));
00066     }
00067     SECTION("As a mirror connection.")
00068     {
00069         REQUIRE_NO_THROW(
00070             Connection(
00071                 "name", filter, true, std::move(pool), std::move(queue)));
00072     }
00073     SECTION("Ensures the filter pointer is not null.")
00074     {
00075         REQUIRE_THROWS_AS(
00076             Connection(
00077                 "name", nullptr, false, std::move(pool), std::move(queue)),
00078             std::invalid_argument);
00079         pool = mock_unique(mock_pool);
00080         queue = mock_unique(mock_queue);
00081         REQUIRE_THROWS_WITH(
00082             Connection(
00083                 "name", nullptr, false, std::move(pool), std::move(queue)),
00084             "Given filter pointer is null.");
00085     }
00086     SECTION("Ensures the pool pointer is not null.")
00087     {
00088         REQUIRE_THROWS_AS(
00089             Connection("name", filter, false, nullptr, std::move(queue)),
00090             std::invalid_argument);
00091         queue = mock_unique(mock_queue);
00092         REQUIRE_THROWS_WITH(
00093             Connection("name", filter, false, nullptr, std::move(queue)),
00094             "Given pool pointer is null.");
00095     }
00096     SECTION("Ensures the queue pointer is not null.")
00097     {
00098         REQUIRE_THROWS_AS(
00099             Connection("name", filter, false, std::move(pool), nullptr),
00100             std::invalid_argument);
00101         pool = mock_unique(mock_pool);
00102         REQUIRE_THROWS_WITH(
00103             Connection("name", filter, false, std::move(pool), nullptr),
00104             "Given queue pointer is null.");
00105     }
00106 }
00107
00108
00109 TEST_CASE("Connection's are printable", "[Connection]")
00110 {
00111     fakeit::Mock<Filter> mock_filter;
00112     auto filter = mock_shared(mock_filter);
00113     REQUIRE(str(Connection("some_name", filter)) == "some_name");
00114     REQUIRE(str(Connection("/dev/ttyUSB0", filter)) == "/dev/ttyUSB0");
00115     REQUIRE(str(Connection("127.0.0.1:14555", filter)) == "127.0.0.1:14555");

```

```

00116 }
00117
00118
00119 TEST_CASE("Connection's 'add_address' method adds/updates addresses.",
00120     "[Connection]")
00121 {
00122     Logger::level(0);
00123     fakeit::Mock<Filter> mock_filter;
00124     fakeit::Mock<AddressPool>> mock_pool;
00125     fakeit::Mock<PacketQueue> mock_queue;
00126     fakeit::Fake(Method(mock_pool, add));
00127     auto filter = mock_shared(mock_filter);
00128     auto pool = mock_unique(mock_pool);
00129     auto queue = mock_unique(mock_queue);
00130     Connection conn("DEVICE", filter, false, std::move(pool), std::move(queue));
00131     SECTION("does not contain address (with logging)")
00132     {
00133         Logger::level(1);
00134         MockCOut mock_cout;
00135         fakeit::When(Method(mock_pool, contains)).AlwaysReturn(false);
00136         conn.add_address(MAVAddress("192.168"));
00137         fakeit::Verify(Method(mock_pool, add).Matching([](auto a)
00138         {
00139             return a == MAVAddress("192.168");
00140         })).Once();
00141         REQUIRE(
00142             mock_cout.buffer().substr(21) ==
00143             "new component 192.168 on DEVICE\n");
00144     }
00145     SECTION("does not contain address (without logging)")
00146     {
00147         MockCOut mock_cout;
00148         fakeit::When(Method(mock_pool, contains)).AlwaysReturn(false);
00149         conn.add_address(MAVAddress("192.168"));
00150         fakeit::Verify(Method(mock_pool, add).Matching([](auto a)
00151         {
00152             return a == MAVAddress("192.168");
00153         })).Once();
00154         REQUIRE(mock_cout.buffer().empty());
00155     }
00156     SECTION("already contains address (with logging)")
00157     {
00158         Logger::level(1);
00159         MockCOut mock_cout;
00160         fakeit::When(Method(mock_pool, contains)).AlwaysReturn(true);
00161         conn.add_address(MAVAddress("192.168"));
00162         fakeit::Verify(Method(mock_pool, add).Matching([](auto a)
00163         {
00164             return a == MAVAddress("192.168");
00165         })).Once();
00166         REQUIRE(mock_cout.buffer().empty());
00167     }
00168     SECTION("already contains address (without logging)")
00169     {
00170         MockCOut mock_cout;
00171         fakeit::When(Method(mock_pool, contains)).AlwaysReturn(true);
00172         conn.add_address(MAVAddress("192.168"));
00173         fakeit::Verify(Method(mock_pool, add).Matching([](auto a)
00174         {
00175             return a == MAVAddress("192.168");
00176         })).Once();
00177         REQUIRE(mock_cout.buffer().empty());
00178     }
00179     Logger::level(0);
00180 }
00181
00182
00183 TEST_CASE("Connection's 'next_packet' method.", "[Connection]")
00184 {
00185     auto ping = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00186     fakeit::Mock<Filter> mock_filter;
00187     fakeit::Mock<AddressPool>> mock_pool;
00188     fakeit::Mock<PacketQueue> mock_queue;
00189     auto filter = mock_shared(mock_filter);
00190     auto pool = mock_unique(mock_pool);
00191     auto queue = mock_unique(mock_queue);
00192     Connection conn("name", filter, false, std::move(pool), std::move(queue));
00193     SECTION("Returns the next packet.")
00194     {
00195         fakeit::When(
00196             OverloadedMethod(

```

```

00197             mock_queue, pop,
00198             std::shared_ptr<const Packet>(
00199                 const std::chrono::nanoseconds &)).Return(ping);
00200             std::chrono::nanoseconds timeout = 1ms;
00201             auto packet = conn.next_packet(timeout);
00202             REQUIRE(packet != nullptr);
00203             REQUIRE(*packet == *ping);
00204             fakeit::Verify(
00205                 OverloadedMethod(
00206                     mock_queue, pop,
00207                     std::shared_ptr<const Packet>(
00208                         const std::chrono::nanoseconds &)).Matching([](auto a)
00209 {
00210     return a == 1ms;
00211 }).Once();
00212 }
00213 SECTION("Or times out and returns nullptr.")
00214 {
00215     fakeit::When(
00216         OverloadedMethod(
00217             mock_queue, pop,
00218             std::shared_ptr<const Packet>(
00219                 const std::chrono::nanoseconds &)).Return(nullptr);
00220             std::chrono::nanoseconds timeout = 0ms;
00221             REQUIRE(conn.next_packet(timeout) == nullptr);
00222             fakeit::Verify(
00223                 OverloadedMethod(
00224                     mock_queue, pop,
00225                     std::shared_ptr<const Packet>(
00226                         const std::chrono::nanoseconds &)).Matching([](auto a)
00227 {
00228     return a == 0ms;
00229 }).Once();
00230 }
00231 }
00232
00233
00234 TEST_CASE("Connection's 'send' method ensures the given packet is not "
00235     "nullptr.", "[Connection]")
00236 {
00237     fakeit::Mock<Filter> mock_filter;
00238     fakeit::Mock<AddressPool<>> mock_pool;
00239     fakeit::Mock<PacketQueue> mock_queue;
00240     auto filter = mock_shared(mock_filter);
00241     auto pool = mock_unique(mock_pool);
00242     auto queue = mock_unique(mock_queue);
00243     Connection conn("name", filter, false, std::move(pool), std::move(queue));
00244 SECTION("Ensures the given packet is not nullptr.")
00245 {
00246     REQUIRE_THROWS_AS(conn.send(nullptr), std::invalid_argument);
00247     REQUIRE_THROWS_WITH(
00248         conn.send(nullptr), "Given packet pointer is null.");
00249 }
00250 }
00251
00252
00253 TEST_CASE("Connection's 'send' method (with destination address).",
00254     "[Connection]")
00255 {
00256     Logger::level(2);
00257     // Mocked objects.
00258     fakeit::Mock<Filter> mock_filter;
00259     fakeit::Mock<AddressPool<>> mock_pool;
00260     fakeit::Mock<PacketQueue> mock_queue;
00261     fakeit::Fake(Method(mock_queue, push));
00262     auto filter = mock_shared(mock_filter);
00263     auto pool = mock_unique(mock_pool);
00264     auto queue = mock_unique(mock_queue);
00265     // Packets for testing.
00266     auto source_connection = std::make_shared<Connection>("SOURCE", filter);
00267     auto ping = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00268     ping->connection(source_connection);
00269     // Connection for testing.
00270     Connection conn("DEST", filter, false, std::move(pool), std::move(queue));
00271 SECTION("Adds the packet to the PacketQueue if the destination can be "
00272     "reached on this connection and the filter allows it "
00273     "(with logging.)")
00274 {
00275     Logger::level(3);
00276     MockCOut mock_cout;
00277     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(

```

```

00278      [&] (auto & a, auto & b)
00279      {
00280          (void)a;
00281          (void)b;
00282          return std::pair<bool, int>(true, 2);
00283      });
00284      fakeit::When(Method(mock_pool, contains)).AlwaysDo([&] (auto & a)
00285      {
00286          return a != MAVAddress("192.168");
00287      });
00288      conn.send(ping);
00289      fakeit::Verify(Method(mock_queue, push)).Once();
00290      fakeit::Verify(Method(mock_queue, push).Matching([&] (auto a, auto b)
00291      {
00292          return a != nullptr && *a == *ping && b == 2;
00293      })).Once();
00294      REQUIRE(
00295          mock_cout.buffer().substr(21) ==
00296          "accepted PING (#4) from 192.168 to 127.1 (v2.0) "
00297          "source SOURCE dest DEST\n");
00298  }
00299 SECTION("Adds the packet to the PacketQueue if the destination can be "
00300         "reached on this connection and the filter allows it "
00301         "(without logging).")
00302 {
00303     MockCOut mock_cout;
00304     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00305         [&] (auto & a, auto & b)
00306         {
00307             (void)a;
00308             (void)b;
00309             return std::pair<bool, int>(true, 2);
00310         });
00311     fakeit::When(Method(mock_pool, contains)).AlwaysDo([&] (auto & a)
00312     {
00313         return a != MAVAddress("192.168");
00314     });
00315     conn.send(ping);
00316     fakeit::Verify(Method(mock_queue, push)).Once();
00317     fakeit::Verify(Method(mock_queue, push).Matching([&] (auto a, auto b)
00318     {
00319         return a != nullptr && *a == *ping && b == 2;
00320     })).Once();
00321     REQUIRE(mock_cout.buffer().empty());
00322 }
00323 SECTION("Silently drops the packet if the filter rejects it "
00324         "(with logging).")
00325 {
00326     Logger::level(3);
00327     MockCOut mock_cout;
00328     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00329         [&] (auto & a, auto & b)
00330         {
00331             (void)a;
00332             (void)b;
00333             return std::pair<bool, int>(false, 0);
00334         });
00335     fakeit::When(Method(mock_pool, contains)).AlwaysDo([&] (auto & a)
00336     {
00337         return a != MAVAddress("192.168");
00338     });
00339     conn.send(ping);
00340     fakeit::Verify(Method(mock_queue, push)).Exactly(0);
00341     REQUIRE(
00342         mock_cout.buffer().substr(21) ==
00343         "rejected PING (#4) from 192.168 to 127.1 (v2.0) "
00344         "source SOURCE dest DEST\n");
00345 }
00346 SECTION("Silently drops the packet if the filter rejects it "
00347         "(without logging).")
00348 {
00349     MockCOut mock_cout;
00350     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00351         [&] (auto & a, auto & b)
00352         {
00353             (void)a;
00354             (void)b;
00355             return std::pair<bool, int>(false, 0);
00356         });
00357     fakeit::When(Method(mock_pool, contains)).AlwaysDo([&] (auto & a)
00358     {

```

```

00359         return a != MAVAddress("192.168");
00360     });
00361     conn.send(ping);
00362     fakeit::Verify(Method(mock_queue, push)).Exactly(0);
00363     REQUIRE(mock_cout.buffer().empty());
00364 }
00365 SECTION("Silently drops the packet if the destination cannot be "
00366           "reached on this connection (with logging).")
00367 {
00368     Logger::level(3);
00369     MockCOut mock_cout;
00370     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00371         [&](auto & a, auto & b)
00372     {
00373         (void)a;
00374         (void)b;
00375         return std::pair<bool, int>(true, 0);
00376     });
00377     fakeit::When(Method(mock_pool, addresses)).AlwaysReturn(
00378         std::vector<MAVAddress>());
00379     fakeit::When(Method(mock_pool, contains)).AlwaysDo([&](auto & a)
00380     {
00381         (void)a;
00382         return false;
00383     });
00384     conn.send(ping);
00385     fakeit::Verify(Method(mock_queue, push)).Exactly(0);
00386     REQUIRE(mock_cout.buffer().empty());
00387 }
00388 SECTION("Silently drops the packet if the destination cannot be "
00389           "reached on this connection (without logging).")
00390 {
00391     MockCOut mock_cout;
00392     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00393         [&](auto & a, auto & b)
00394     {
00395         (void)a;
00396         (void)b;
00397         return std::pair<bool, int>(true, 0);
00398     });
00399     fakeit::When(Method(mock_pool, addresses)).AlwaysReturn(
00400         std::vector<MAVAddress>());
00401     fakeit::When(Method(mock_pool, contains)).AlwaysDo([&](auto & a)
00402     {
00403         (void)a;
00404         return false;
00405     });
00406     conn.send(ping);
00407     fakeit::Verify(Method(mock_queue, push)).Exactly(0);
00408     REQUIRE(mock_cout.buffer().empty());
00409 }
00410 Logger::level(0);
00411 }
00412
00413
00414 TEST_CASE("Connection's 'send' method (without destination address).",
00415             "[Connection]")
00416 {
00417     Logger::level(2);
00418     // Mocked objects.
00419     fakeit::Mock<Filter> mock_filter;
00420     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00421         [&](auto & a, auto & b)
00422     {
00423         (void)a;
00424
00425         if (b == MAVAddress("192.168"))
00426         {
00427             return std::pair<bool, int>(true, 2);
00428         }
00429
00430         if (b == MAVAddress("172.16"))
00431         {
00432             return std::pair<bool, int>(true, -3);
00433         }
00434
00435         return std::pair<bool, int>(false, 0);
00436     });
00437     fakeit::Mock<AddressPool<>> mock_pool;
00438     fakeit::When(Method(mock_pool, contains)).AlwaysDo([](MAVAddress addr)
00439     {

```

```
00440     if (addr == MAVAddress("10.10"))
00441     {
00442         return true;
00443     }
00444
00445     if (addr == MAVAddress("172.16"))
00446     {
00447         return true;
00448     }
00449
00450     if (addr == MAVAddress("192.168"))
00451     {
00452         return true;
00453     }
00454
00455     return false;
00456 };
00457 fakeit::Mock<PacketQueue> mock_queue;
00458 fakeit::Fake(Method(mock_queue, push));
00459 auto filter = mock_shared(mock_filter);
00460 auto pool = mock_unique(mock_pool);
00461 auto queue = mock_unique(mock_queue);
00462 // Packets for testing.
00463 auto source_connection = std::make_shared<Connection>("SOURCE", filter);
00464 auto heartbeat =
00465     std::make_shared<packet_v2::Packet>(to_vector(HeartbeatV2()));
00466 heartbeat->connection(source_connection);
00467 // Connection for testing.
00468 Connection conn("DEST", filter, false, std::move(pool), std::move(queue));
00469 SECTION("Adds the packet to the PacketQueue if the filter allows it for "
00470         "any of the reachable addresses and favors the higher priority "
00471         "(increasing priority) (with logging).")
00472 {
00473     Logger::level(3);
00474     MockCOut mock_cout;
00475     fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00476     {
00477         std::vector<MAVAddress> addr =
00478         {
00479             MAVAddress("10.10"),
00480             MAVAddress("172.16"),
00481             MAVAddress("192.168")
00482         };
00483         return addr;
00484     });
00485     conn.send(heartbeat);
00486     fakeit::Verify(Method(mock_queue, push)).Once();
00487     fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
00488     {
00489         return a != nullptr && *a == *heartbeat && b == 2;
00490     })).Once();
00491     REQUIRE(
00492         mock_cout.buffer().substr(21) ==
00493         "accepted HEARTBEAT (#0) from 127.1 (v2.0) "
00494         "source SOURCE dest DEST\n");
00495 }
00496 SECTION("Adds the packet to the PacketQueue if the filter allows it for "
00497         "any of the reachable addresses and favors the higher priority "
00498         "(increasing priority) (without logging).")
00499 {
00500     MockCOut mock_cout;
00501     fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00502     {
00503         std::vector<MAVAddress> addr =
00504         {
00505             MAVAddress("10.10"),
00506             MAVAddress("172.16"),
00507             MAVAddress("192.168")
00508         };
00509         return addr;
00510     });
00511     conn.send(heartbeat);
00512     fakeit::Verify(Method(mock_queue, push)).Once();
00513     fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
00514     {
00515         return a != nullptr && *a == *heartbeat && b == 2;
00516     })).Once();
00517     REQUIRE(mock_cout.buffer().empty());
00518 }
00519 SECTION("Adds the packet to the PacketQueue if the filter allows it for "
00520         "any of the reachable addresses and favors the higher priority "
```

```

00521         "(decreasing priority) (with logging)."
00522     {
00523         Logger::level(3);
00524         MockCOut mock_cout;
00525         fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00526         {
00527             std::vector<MAVAddress> addr =
00528             {
00529                 MAVAddress("192.168"),
00530                 MAVAddress("172.16"),
00531                 MAVAddress("10.10")
00532             };
00533             return addr;
00534         });
00535         conn.send(heartbeat);
00536         fakeit::Verify(Method(mock_queue, push)).Once();
00537         fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
00538         {
00539             return a != nullptr && *a == *heartbeat && b == 2;
00540         })).Once();
00541         REQUIRE(
00542             mock_cout.buffer().substr(21) ==
00543             "accepted HEARTBEAT (#0) from 127.1 (v2.0) "
00544             "source SOURCE dest DEST\n");
00545     }
00546     SECTION("Adds the packet to the PacketQueue if the filter allows it for "
00547         "any of the reachable addresses and favors the higher priority "
00548         "(decreasing priority) (without logging).")
00549     {
00550         MockCOut mock_cout;
00551         fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00552         {
00553             std::vector<MAVAddress> addr =
00554             {
00555                 MAVAddress("192.168"),
00556                 MAVAddress("172.16"),
00557                 MAVAddress("10.10")
00558             };
00559             return addr;
00560         });
00561         conn.send(heartbeat);
00562         fakeit::Verify(Method(mock_queue, push)).Once();
00563         fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
00564         {
00565             return a != nullptr && *a == *heartbeat && b == 2;
00566         })).Once();
00567         REQUIRE(mock_cout.buffer().empty());
00568     }
00569     SECTION("Silently drops the packet if the filter rejects it for all "
00570         "reachable addresses (with logging).")
00571     {
00572         Logger::level(3);
00573         MockCOut mock_cout;
00574         fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00575             [&](auto & a, auto & b)
00576             {
00577                 (void)a;
00578                 (void)b;
00579                 return std::pair<bool, int>(false, 0);
00580             });
00581         fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00582         {
00583             std::vector<MAVAddress> addr =
00584             {
00585                 MAVAddress("10.10"),
00586                 MAVAddress("172.16"),
00587                 MAVAddress("192.168")
00588             };
00589             return addr;
00590         });
00591         conn.send(heartbeat);
00592         fakeit::Verify(Method(mock_queue, push)).Exactly(0);
00593         REQUIRE(
00594             mock_cout.buffer().substr(21) ==
00595             "rejected HEARTBEAT (#0) from 127.1 (v2.0) "
00596             "source SOURCE dest DEST\n");
00597     }
00598     SECTION("Silently drops the packet if the filter rejects it for all "
00599         "reachable addresses (without logging).")
00600     {
00601         MockCOut mock_cout;

```

```

00602     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00603         [&](auto & a, auto & b)
00604     {
00605         (void)a;
00606         (void)b;
00607         return std::pair<bool, int>(false, 0);
00608     });
00609     fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00610     {
00611         std::vector<MAVAddress> addr =
00612         {
00613             MAVAddress("10.10"),
00614             MAVAddress("172.16"),
00615             MAVAddress("192.168")
00616         };
00617         return addr;
00618     });
00619     conn.send(heartbeat);
00620     fakeit::Verify(Method(mock_queue, push)).Exactly(0);
00621     REQUIRE(mock_cout.buffer().empty());
00622 }
00623 Logger::level(0);
00624 }
00625
00626
00627 TEST_CASE("Connection's 'send' method (with broadcast address 0.0).",
00628     "[Connection]")
00629 {
00630     Logger::level(2);
00631     // Mocked objects.
00632     fakeit::Mock<Filter> mock_filter;
00633     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00634         [&](auto & a, auto & b)
00635     {
00636         (void)a;
00637
00638         if (b == MAVAddress("192.168"))
00639         {
00640             return std::pair<bool, int>(true, 2);
00641         }
00642
00643         if (b == MAVAddress("172.16"))
00644         {
00645             return std::pair<bool, int>(true, -3);
00646         }
00647
00648         return std::pair<bool, int>(false, 0);
00649     });
00650     fakeit::Mock<AddressPool<>> mock_pool;
00651     fakeit::When(Method(mock_pool, contains)).AlwaysDo([](MAVAddress addr)
00652     {
00653         if (addr == MAVAddress("10.10"))
00654         {
00655             return true;
00656         }
00657
00658         if (addr == MAVAddress("172.16"))
00659         {
00660             return true;
00661         }
00662
00663         if (addr == MAVAddress("192.168"))
00664         {
00665             return true;
00666         }
00667
00668         return false;
00669     });
00670     fakeit::Mock<PacketQueue> mock_queue;
00671     fakeit::Fake(Method(mock_queue, push));
00672     auto filter = mock_shared(mock_filter);
00673     auto pool = mock_unique(mock_pool);
00674     auto queue = mock_unique(mock_queue);
00675     // Packets for testing.
00676     auto source_connection = std::make_shared<Connection>("SOURCE", filter);
00677     auto mission_set_current =
00678         std::make_shared<packet_v2::Packet>(to_vector(MissionSetCurrentV2()));
00679     mission_set_current->connection(source_connection);
00680     // Connection for testing.
00681     Connection conn("DEST", filter, false, std::move(pool), std::move(queue));
00682     SECTION("Adds the packet to the PacketQueue if the filter allows it for "

```

```

00683             "any of the reachable addresses and favors the higher priority "
00684             "(increasing priority) (with logging)."
00685     {
00686         Logger::level(3);
00687         MockCOut mock_cout;
00688         fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00689         {
00690             std::vector<MAVAddress> addr =
00691             {
00692                 MAVAddress("10.10"),
00693                 MAVAddress("172.16"),
00694                 MAVAddress("192.168")
00695             };
00696             return addr;
00697         });
00698         conn.send(mission_set_current);
00699         fakeit::Verify(Method(mock_queue, push)).Once();
00700         fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
00701         {
00702             return a != nullptr && *a == *mission_set_current && b == 2;
00703         })).Once();
00704         REQUIRE(
00705             mock_cout.buffer().substr(21) ==
00706             "accepted MISSION_SET_CURRENT (#41) from 255.0 to 0.0 (v2.0) "
00707             "source SOURCE dest DEST\n");
00708     }
00709     SECTION("Adds the packet to the PacketQueue if the filter allows it for "
00710             "any of the reachable addresses and favors the higher priority "
00711             "(increasing priority) (without logging).")
00712     {
00713         MockCOut mock_cout;
00714         fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00715         {
00716             std::vector<MAVAddress> addr =
00717             {
00718                 MAVAddress("10.10"),
00719                 MAVAddress("172.16"),
00720                 MAVAddress("192.168")
00721             };
00722             return addr;
00723         });
00724         conn.send(mission_set_current);
00725         fakeit::Verify(Method(mock_queue, push)).Once();
00726         fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
00727         {
00728             return a != nullptr && *a == *mission_set_current && b == 2;
00729         })).Once();
00730         REQUIRE(mock_cout.buffer().empty());
00731     }
00732     SECTION("Adds the packet to the PacketQueue if the filter allows it for "
00733             "any of the reachable addresses and favors the higher priority "
00734             "(decreasing priority) (with logging).")
00735     {
00736         Logger::level(3);
00737         MockCOut mock_cout;
00738         fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00739         {
00740             std::vector<MAVAddress> addr =
00741             {
00742                 MAVAddress("192.168"),
00743                 MAVAddress("172.16"),
00744                 MAVAddress("10.10")
00745             };
00746             return addr;
00747         });
00748         conn.send(mission_set_current);
00749         fakeit::Verify(Method(mock_queue, push)).Once();
00750         fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
00751         {
00752             return a != nullptr && *a == *mission_set_current && b == 2;
00753         })).Once();
00754         REQUIRE(
00755             mock_cout.buffer().substr(21) ==
00756             "accepted MISSION_SET_CURRENT (#41) from 255.0 to 0.0 (v2.0) "
00757             "source SOURCE dest DEST\n");
00758     }
00759     SECTION("Adds the packet to the PacketQueue if the filter allows it for "
00760             "any of the reachable addresses and favors the higher priority "
00761             "(decreasing priority) (without logging).")
00762     {
00763         MockCOut mock_cout;

```

```

00764     fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00765     {
00766         std::vector<MAVAddress> addr =
00767         {
00768             MAVAddress("192.168"),
00769             MAVAddress("172.16"),
00770             MAVAddress("10.10")
00771         };
00772         return addr;
00773     });
00774     conn.send(mission_set_current);
00775     fakeit::Verify(Method(mock_queue, push)).Once();
00776     fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
00777     {
00778         return a != nullptr && *a == *mission_set_current && b == 2;
00779     })).Once();
00780     REQUIRE(mock_cout.buffer().empty());
00781 }
00782 SECTION("Silently drops the packet if the filter rejects it for all "
00783         "reachable addresses (with logging).")
00784 {
00785     Logger::level(3);
00786     MockCOut mock_cout;
00787     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00788         [&](auto & a, auto & b)
00789     {
00790         (void)a;
00791         (void)b;
00792         return std::pair<bool, int>(false, 0);
00793     });
00794     fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00795     {
00796         std::vector<MAVAddress> addr =
00797         {
00798             MAVAddress("10.10"),
00799             MAVAddress("192.168"),
00800             MAVAddress("172.16")
00801         };
00802         return addr;
00803     });
00804     conn.send(mission_set_current);
00805     fakeit::Verify(Method(mock_queue, push)).Exactly(0);
00806     REQUIRE(
00807         mock_cout.buffer().substr(21) ==
00808         "rejected MISSION_SET_CURRENT (#41) from 255.0 to 0.0 (v2.0) "
00809         "source SOURCE dest DEST\n");
00810 }
00811 SECTION("Silently drops the packet if the filter rejects it for all "
00812         "reachable addresses (without logging).")
00813 {
00814     MockCOut mock_cout;
00815     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00816         [&](auto & a, auto & b)
00817     {
00818         (void)a;
00819         (void)b;
00820         return std::pair<bool, int>(false, 0);
00821     });
00822     fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00823     {
00824         std::vector<MAVAddress> addr =
00825         {
00826             MAVAddress("10.10"),
00827             MAVAddress("192.168"),
00828             MAVAddress("172.16")
00829         };
00830         return addr;
00831     });
00832     conn.send(mission_set_current);
00833     fakeit::Verify(Method(mock_queue, push)).Exactly(0);
00834     REQUIRE(mock_cout.buffer().empty());
00835 }
00836 Logger::level(0);
00837 }
00838
00839
00840 TEST_CASE("Connection's 'send' method (with component broadcast address x.0).",
00841             "[Connection]")
00842 {
00843     Logger::level(2);
00844     // Mocked objects.

```

```

00845     fakeit::Mock<Filter> mock_filter;
00846     fakeit::Mock<AddressPool<>> mock_pool;
00847     fakeit::When(Method(mock_pool, contains)).AlwaysDo([](MAVAddress addr)
00848     {
00849         if (addr == MAVAddress("10.10"))
00850         {
00851             return true;
00852         }
00853
00854         if (addr == MAVAddress("123.16"))
00855         {
00856             return true;
00857         }
00858
00859         if (addr == MAVAddress("123.17"))
00860         {
00861             return true;
00862         }
00863
00864         if (addr == MAVAddress("123.168"))
00865         {
00866             return true;
00867         }
00868
00869         return false;
00870     });
00871     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00872         [&](auto & a, auto & b)
00873     {
00874         (void)a;
00875
00876         if (b == MAVAddress("123.168"))
00877         {
00878             return std::pair<bool, int>(true, 2);
00879         }
00880
00881         if (b == MAVAddress("123.16"))
00882         {
00883             return std::pair<bool, int>(true, -3);
00884         }
00885
00886         return std::pair<bool, int>(false, 0);
00887     });
00888     fakeit::Mock<PacketQueue> mock_queue;
00889     fakeit::Fake(Method(mock_queue, push));
00890     auto filter = mock_shared(mock_filter);
00891     auto pool = mock_unique(mock_pool);
00892     auto queue = mock_unique(mock_queue);
00893     // Packets for testing.
00894     auto set_mode = std::make_shared<packet_v2::Packet>(to_vector(SetModeV2()));
00895     auto source_connection = std::make_shared<Connection>("SOURCE", filter);
00896     set_mode->connection(source_connection);
00897     // Connection for testing.
00898     Connection conn("DEST", filter, false, std::move(pool), std::move(queue));
00899     SECTION("Adds the packet to the PacketQueue if the filter allows it for "
00900             "any reachable component address of the given system address and "
00901             "favors the higher priority (increasing priority) (with logging).")
00902     {
00903         Logger::level(3);
00904         MockCout mock_cout;
00905         fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00906         {
00907             std::vector<MAVAddress> addr =
00908             {
00909                 MAVAddress("10.10"),
00910                 MAVAddress("123.16"),
00911                 MAVAddress("123.17"),
00912                 MAVAddress("123.168")
00913             };
00914             return addr;
00915         });
00916         conn.send(set_mode);
00917         fakeit::Verify(Method(mock_queue, push)).Once();
00918         fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
00919         {
00920             return a != nullptr && *a == *set_mode && b == 2;
00921         })).Once();
00922         REQUIRE(
00923             mock_cout.buffer().substr(21) ==
00924             "accepted SET_MODE (#11) from 172.0 to 123.0 (v2.0) "
00925             "source SOURCE dest DEST\n");

```

```
00926     }
00927     SECTION("Adds the packet to the PacketQueue if the filter allows it for "
00928             "any reachable component address of the given system address and "
00929             "favors the higher priority (increasing priority) "
00930             "(without logging).")
00931 {
00932     MockCout mock_cout;
00933     fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00934     {
00935         std::vector<MAVAddress> addr =
00936         {
00937             MAVAddress("10.10"),
00938             MAVAddress("123.16"),
00939             MAVAddress("123.17"),
00940             MAVAddress("123.168")
00941         };
00942         return addr;
00943     });
00944     conn.send(set_mode);
00945     fakeit::Verify(Method(mock_queue, push)).Once();
00946     fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
00947     {
00948         return a != nullptr && *a == *set_mode && b == 2;
00949     })).Once();
00950     REQUIRE(mock_cout.buffer().empty());
00951 }
00952 SECTION("Adds the packet to the PacketQueue if the filter allows it for "
00953         "any reachable component address of the given system address and "
00954         "favors the higher priority (decreasing priority) (with logging).")
00955 {
00956     Logger::level(3);
00957     MockCout mock_cout;
00958     fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00959     {
00960         std::vector<MAVAddress> addr =
00961         {
00962             MAVAddress("123.168"),
00963             MAVAddress("123.17"),
00964             MAVAddress("123.16"),
00965             MAVAddress("10.10")
00966         };
00967         return addr;
00968     });
00969     conn.send(set_mode);
00970     fakeit::Verify(Method(mock_queue, push)).Once();
00971     fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
00972     {
00973         return a != nullptr && *a == *set_mode && b == 2;
00974     })).Once();
00975     REQUIRE(
00976         mock_cout.buffer().substr(21) ==
00977         "accepted SET_MODE (#11) from 172.0 to 123.0 (v2.0) "
00978         "source SOURCE dest DEST\n");
00979 }
00980 SECTION("Adds the packet to the PacketQueue if the filter allows it for "
00981         "any reachable component address of the given system address and "
00982         "favors the higher priority (decreasing priority) "
00983         "(without logging).")
00984 {
00985     MockCout mock_cout;
00986     fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
00987     {
00988         std::vector<MAVAddress> addr =
00989         {
00990             MAVAddress("123.168"),
00991             MAVAddress("123.17"),
00992             MAVAddress("123.16"),
00993             MAVAddress("10.10")
00994         };
00995         return addr;
00996     });
00997     conn.send(set_mode);
00998     fakeit::Verify(Method(mock_queue, push)).Once();
00999     fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
01000     {
01001         return a != nullptr && *a == *set_mode && b == 2;
01002     })).Once();
01003     REQUIRE(mock_cout.buffer().empty());
01004 }
01005 SECTION("Silently drops the packet if the filter rejects it for all "
01006         "matching addresses (with logging).")
```

```

01007    {
01008        Logger::level(3);
01009        MockCOut mock_cout;
01010        fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
01011            [&](auto & a, auto & b)
01012            {
01013                (void)a;
01014                (void)b;
01015                return std::pair<bool, int>(false, 0);
01016            });
01017        fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
01018        {
01019            std::vector<MAVAddress> addr =
01020            {
01021                MAVAddress("10.10"),
01022                MAVAddress("123.16"),
01023                MAVAddress("123.17"),
01024                MAVAddress("123.168")
01025            };
01026            return addr;
01027        });
01028        conn.send(set_mode);
01029        fakeit::Verify(Method(mock_queue, push)).Exactly(0);
01030        REQUIRE(
01031            mock_cout.buffer().substr(21) ==
01032            "rejected SET_MODE (#11) from 172.0 to 123.0 (v2.0) "
01033            "source SOURCE dest DEST\n");
01034    }
01035    SECTION("Silently drops the packet if the filter rejects it for all "
01036            "matching addresses (without logging).")
01037    {
01038        MockCOut mock_cout;
01039        fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
01040            [&](auto & a, auto & b)
01041            {
01042                (void)a;
01043                (void)b;
01044                return std::pair<bool, int>(false, 0);
01045            });
01046        fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
01047        {
01048            std::vector<MAVAddress> addr =
01049            {
01050                MAVAddress("10.10"),
01051                MAVAddress("123.16"),
01052                MAVAddress("123.17"),
01053                MAVAddress("123.168")
01054            };
01055            return addr;
01056        });
01057        conn.send(set_mode);
01058        fakeit::Verify(Method(mock_queue, push)).Exactly(0);
01059        REQUIRE(mock_cout.buffer().empty());
01060    }
01061    SECTION("Silently drops the packet if the destination system cannot be "
01062            "reached on this connection (with logging).")
01063    {
01064        Logger::level(3);
01065        MockCOut mock_cout;
01066        fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
01067            [&](auto & a, auto & b)
01068            {
01069                (void)a;
01070                (void)b;
01071                return std::pair<bool, int>(true, 0);
01072            });
01073        fakeit::When(Method(mock_pool, addresses)).AlwaysReturn(
01074            std::vector<MAVAddress>());
01075        fakeit::When(Method(mock_pool, contains)).AlwaysDo([&](auto & a)
01076        {
01077            (void)a;
01078            return false;
01079        });
01080        conn.send(set_mode);
01081        fakeit::Verify(Method(mock_queue, push)).Exactly(0);
01082        REQUIRE(mock_cout.buffer().empty());
01083    }
01084    SECTION("Silently drops the packet if the destination system cannot be "
01085            "reached on this connection (without logging).")
01086    {
01087        MockCOut mock_cout;

```

```

01088     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
01089         [&](auto & a, auto & b)
01090     {
01091         (void)a;
01092         (void)b;
01093         return std::pair<bool, int>(true, 0);
01094     });
01095     fakeit::When(Method(mock_pool, addresses)).AlwaysReturn(
01096         std::vector<MAVAddress>());
01097     fakeit::When(Method(mock_pool, contains)).AlwaysDo([&](auto & a)
01098     {
01099         (void)a;
01100         return false;
01101     });
01102     conn.send(set_mode);
01103     fakeit::Verify(Method(mock_queue, push)).Exactly(0);
01104     REQUIRE(mock_cout.buffer().empty());
01105 }
01106 Logger::level(0);
01107 }
01108
01109
01110 TEST_CASE("Connection's 'send' method (destination address, system reachable, "
01111     "component unreachable.", "[Connection]")
01112 {
01113     Logger::level(2);
01114     // Mocked objects.
01115     fakeit::Mock<Filter> mock_filter;
01116     fakeit::Mock<AddressPool<>> mock_pool;
01117     fakeit::When(Method(mock_pool, addresses)).AlwaysDo([]()
01118     {
01119         std::vector<MAVAddress> addr =
01120         {
01121             MAVAddress("10.10"),
01122             MAVAddress("127.16"),
01123             MAVAddress("127.17"),
01124             MAVAddress("127.168")
01125         };
01126         return addr;
01127     });
01128     fakeit::When(Method(mock_pool, contains)).AlwaysDo([](MAVAddress addr)
01129     {
01130         if (addr == MAVAddress("10.10"))
01131         {
01132             return true;
01133         }
01134
01135         if (addr == MAVAddress("127.16"))
01136         {
01137             return true;
01138         }
01139
01140         if (addr == MAVAddress("127.17"))
01141         {
01142             return true;
01143         }
01144
01145         if (addr == MAVAddress("127.168"))
01146         {
01147             return true;
01148         }
01149
01150         return false;
01151     });
01152     fakeit::Mock<PacketQueue> mock_queue;
01153     auto filter = mock_shared(mock_filter);
01154     auto pool = mock_unique(mock_pool);
01155     auto queue = mock_unique(mock_queue);
01156     // Packets for testing.
01157     auto ping = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
01158     auto source_connection = std::make_shared<Connection>("SOURCE", filter);
01159     ping->connection(source_connection);
01160     // Connection for testing.
01161     Connection conn("DEST", filter, false, std::move(pool), std::move(queue));
01162     SECTION("Adds the packet to the PacketQueue if any component of the "
01163         "system is reachable on the connection and the filter allows it "
01164         "to the original component (with logging).")
01165     {
01166         Logger::level(3);
01167         MockCOut mock_cout;
01168         fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(

```

```

01169      [&](auto & a, auto & b)
01170      {
01171          (void)a;
01172
01173          if (b == MAVAddress("127.1"))
01174          {
01175              return std::pair<bool, int>(true, 2);
01176          }
01177
01178          return std::pair<bool, int>(false, 0);
01179      });
01180      fakeit::Fake(Method(mock_queue, push));
01181      conn.send(ping);
01182      fakeit::Verify(Method(mock_queue, push)).Once();
01183      fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
01184      {
01185          return a != nullptr && *a == *ping && b == 2;
01186      })).Once();
01187      REQUIRE(
01188          mock_cout.buffer().substr(21) ==
01189          "accepted PING (#4) from 192.168 to 127.1 (v2.0) "
01190          "source SOURCE dest DEST\n");
01191  }
01192 SECTION("Adds the packet to the PacketQueue if any component of the "
01193         "system is reachable on the connection and the filter allows it "
01194         "to the original component (without logging).")
01195 {
01196     MockCOut mock_cout;
01197     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
01198         [&](auto & a, auto & b)
01199     {
01200         (void)a;
01201
01202         if (b == MAVAddress("127.1"))
01203         {
01204             return std::pair<bool, int>(true, 2);
01205         }
01206
01207         return std::pair<bool, int>(false, 0);
01208     });
01209     fakeit::Fake(Method(mock_queue, push));
01210     conn.send(ping);
01211     fakeit::Verify(Method(mock_queue, push)).Once();
01212     fakeit::Verify(Method(mock_queue, push).Matching([&](auto a, auto b)
01213     {
01214         return a != nullptr && *a == *ping && b == 2;
01215     })).Once();
01216     REQUIRE(mock_cout.buffer().empty());
01217 }
01218 SECTION("Silently drops the packet if the filter rejects it (using "
01219         "it's original address) (with logging).")
01220 {
01221     Logger::level(3);
01222     MockCOut mock_cout;
01223     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
01224         [&](auto & a, auto & b)
01225     {
01226         (void)a;
01227
01228         if (b == MAVAddress("127.1"))
01229         {
01230             return std::pair<bool, int>(false, 0);
01231         }
01232
01233         return std::pair<bool, int>(true, 2);
01234     });
01235     fakeit::Fake(Method(mock_queue, push));
01236     conn.send(ping);
01237     fakeit::Verify(Method(mock_queue, push)).Exactly(0);
01238     REQUIRE(
01239         mock_cout.buffer().substr(21) ==
01240         "rejected PING (#4) from 192.168 to 127.1 (v2.0) "
01241         "source SOURCE dest DEST\n");
01242 }
01243 SECTION("Silently drops the packet if the filter rejects it (using "
01244         "it's original address) (without logging).")
01245 {
01246     MockCOut mock_cout;
01247     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
01248         [&](auto & a, auto & b)
01249     {

```

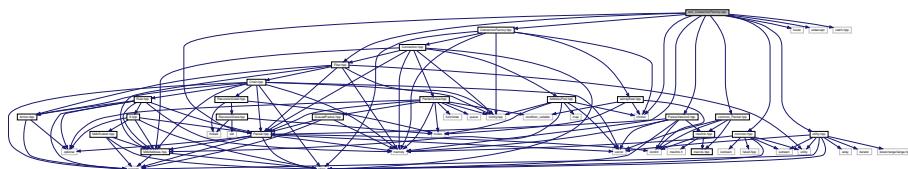
```

01250         (void)a;
01251
01252     if (b == MAVAddress("127.1"))
01253     {
01254         return std::pair<bool, int>(false, 0);
01255     }
01256
01257     return std::pair<bool, int>(true, 2);
01258 });
01259 fakeit::Fake(Method(mock_queue, push));
01260 conn.send(ping);
01261 fakeit::Verify(Method(mock_queue, push)).Exactly(0);
01262 REQUIRE(mock_cout.buffer().empty());
01263 }
01264 Logger::level(0);
01265 }
```

16.205 test_ConnectionFactory.cpp File Reference

```

#include <chrono>
#include <future>
#include <memory>
#include <stdexcept>
#include <catch.hpp>
#include "ConnectionFactory.hpp"
#include "Filter.hpp"
#include "Packet.hpp"
#include "PacketVersion2.hpp"
#include "utility.hpp"
#include "common.hpp"
#include "common_Packet.hpp"
Include dependency graph for test_ConnectionFactory.cpp:
```



Functions

- **TEST_CASE** ("ConnectionFactory's can be constructed.", "[ConnectionFactory]")
- **TEST_CASE** ("ConnectionFactory's 'get' method returns a new connection.", "[ConnectionFactory]")
- **TEST_CASE** ("ConnectionFactory's 'wait_for_packet' method waits for a packet " "on any of the connections created by the factory.", "[ConnectionFactory]")

16.205.1 Function Documentation

16.205.1.1 TEST_CASE() [1/3]

```
TEST_CASE (
    "ConnectionFactory's can be constructed." ,
    "" [ConnectionFactory] )
```

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

References [mock_shared\(\)](#).

Here is the call graph for this function:



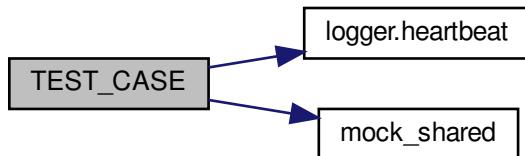
16.205.1.2 TEST_CASE() [2/3]

```
TEST_CASE (
    "ConnectionFactory's 'get' method returns a new connection." ,
    "" [ConnectionFactory] )
```

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

References [logger::heartbeat\(\)](#), and [mock_shared\(\)](#).

Here is the call graph for this function:



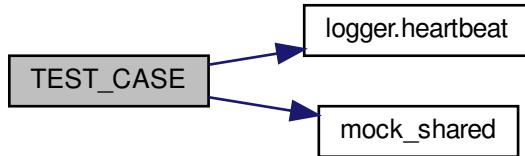
16.205.1.3 TEST_CASE() [3/3]

```
TEST_CASE (
    "ConnectionFactory's 'wait_for_packet' method waits for a packet " "on any of the
connections created by the factory." ,
    "" [ConnectionFactory] )
```

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

References [logger::heartbeat\(\)](#), and [mock_shared\(\)](#).

Here is the call graph for this function:



16.206 test_ConnectionFactory.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <chrono>
00019 #include <future>
00020 #include <memory>
00021 #include <stdexcept>
00022
00023 #include <catch.hpp>
00024
00025 #include "ConnectionFactory.hpp"
00026 #include "Filter.hpp"
00027 #include "Packet.hpp"
00028 #include "PacketVersion2.hpp"
00029 #include "utility.hpp"
00030
00031 #include "common.hpp"
00032 #include "common_Packet.hpp"
00033
00034
```

```

00035 using namespace std::chrono_literals;
00036
00037
00038 TEST_CASE("ConnectionFactory's can be constructed.", "[ConnectionFactory]")
00039 {
00040     fakeit::Mock<Filter> mock_filter;
00041     auto filter = mock_shared(mock_filter);
00042     REQUIRE_NO_THROW(ConnectionFactory<>(filter));
00043     REQUIRE_NO_THROW(ConnectionFactory<>(filter, true));
00044     REQUIRE_NO_THROW(ConnectionFactory<>(filter, false));
00045     SECTION("And ensures the given filter is not null.")
00046     {
00047         REQUIRE_THROWS_AS(ConnectionFactory<>(nullptr), std::invalid_argument);
00048         REQUIRE_THROWS_WITH(
00049             ConnectionFactory<>(nullptr), "Given filter pointer is null.");
00050     }
00051 }
00052
00053
00054 TEST_CASE("ConnectionFactory's 'get' method returns a new connection.",
00055     "[ConnectionFactory]")
00056 {
00057     auto heartbeat =
00058         std::make_shared<packet_v2::Packet>(to_vector(HeartbeatV2()));
00059     fakeit::Mock<Filter> mock_filter;
00060     fakeit::When(Method(mock_filter, will_accept)
00061                 .AlwaysDo([](auto &a, auto &b)
00062                 {
00063                     (void)a;
00064                     (void)b;
00065                     return std::pair<bool, int>(true, 0);
00066                 }));
00067     auto filter = mock_shared(mock_filter);
00068     REQUIRE(filter != nullptr);
00069     ConnectionFactory<> connection_factory(filter);
00070     SECTION("with connection name")
00071     {
00072         std::unique_ptr<Connection> conn = connection_factory.get("CONNECTION");
00073         REQUIRE(conn != nullptr);
00074         REQUIRE(str(*conn) == "CONNECTION");
00075         conn->add_address(MAVAddress("192.168"));
00076         conn->send(heartbeat);
00077         auto packet = conn->next_packet(0s);
00078         REQUIRE(packet != nullptr);
00079         REQUIRE(*packet == *heartbeat);
00080     }
00081     SECTION("without connection name")
00082     {
00083         std::unique_ptr<Connection> conn = connection_factory.get();
00084         REQUIRE(conn != nullptr);
00085         REQUIRE(str(*conn) == "unknown");
00086         conn->add_address(MAVAddress("192.168"));
00087         conn->send(heartbeat);
00088         auto packet = conn->next_packet(0s);
00089         REQUIRE(packet != nullptr);
00090         REQUIRE(*packet == *heartbeat);
00091     }
00092 }
00093
00094
00095 TEST_CASE("ConnectionFactory's 'wait_for_packet' method waits for a packet "
00096     "on any of the connections created by the factory.",
00097     "[ConnectionFactory]")
00098 {
00099     auto heartbeat =
00100         std::make_shared<packet_v2::Packet>(to_vector(HeartbeatV2()));
00101     fakeit::Mock<Filter> mock_filter;
00102     fakeit::When(Method(mock_filter, will_accept)
00103                 .AlwaysDo([](auto &a, auto &b)
00104                 {
00105                     (void)a;
00106                     (void)b;
00107                     return std::pair<bool, int>(true, 0);
00108                 }));
00109     auto filter = mock_shared(mock_filter);
00110     REQUIRE(filter != nullptr);
00111     ConnectionFactory<> connection_factory(filter);
00112     std::unique_ptr<Connection> conn1 = connection_factory.get();
00113     std::unique_ptr<Connection> conn2 = connection_factory.get();
00114     conn1->add_address(MAVAddress("192.168"));
00115     conn2->add_address(MAVAddress("192.168"));

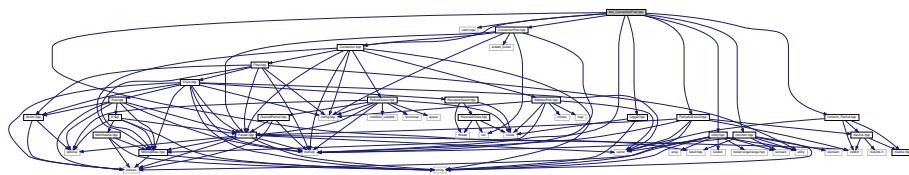
```

```

00116     SECTION("First connection.")
00117     {
00118         auto future = std::async(std::launch::async, [&]()
00119         {
00120             return connection_factory.wait_for_packet(10s);
00121         });
00122         REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00123         conn1->send(heartbeat);
00124         REQUIRE(future.wait_for(5ms) == std::future_status::ready);
00125         REQUIRE(future.get());
00126     }
00127     SECTION("Second connection.")
00128     {
00129         auto future = std::async(std::launch::async, [&]()
00130         {
00131             return connection_factory.wait_for_packet(10s);
00132         });
00133         REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00134         conn2->send(heartbeat);
00135         REQUIRE(future.wait_for(5ms) == std::future_status::ready);
00136         REQUIRE(future.get());
00137     }
00138     SECTION("Returns false on timeout.")
00139     {
00140         auto future = std::async(std::launch::async, [&]()
00141         {
00142             return connection_factory.wait_for_packet(1ms);
00143         });
00144         REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00145         REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00146         REQUIRE_FALSE(future.get());
00147     }
00148 }
```

16.207 test_ConnectionPool.cpp File Reference

```
#include <memory>
#include <catch.hpp>
#include <fakeit.hpp>
#include "Connection.hpp"
#include "ConnectionPool.hpp"
#include "Logger.hpp"
#include "PacketVersion2.hpp"
#include "utility.hpp"
#include "common.hpp"
#include "common_Packet.hpp"
Include dependency graph for test_ConnectionPool.cpp:
```



Functions

- [TEST_CASE \("ConnectionPool's can be constructed.", "\[ConnectionPool\]"\)](#)
- [TEST_CASE \("ConnectionPool's can store at least one connection and send a packet over it.", "\[ConnectionPool\]"\)](#)

- [TEST_CASE](#) ("ConnectionPool's can store more than one connection and send a " "packet over them.", "[[ConnectionPool](#)]")
- [TEST_CASE](#) ("ConnectionPool's 'remove' method removes a connection.", "[[ConnectionPool](#)]")
- [TEST_CASE](#) ("ConnectionPool's 'send' method removes connections that have " "expired.", "[[ConnectionPool](#)]")

16.207.1 Function Documentation

16.207.1.1 TEST_CASE() [1/5]

```
TEST_CASE (
    "ConnectionPool's can be constructed." ,
    "" [ConnectionPool] )
```

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

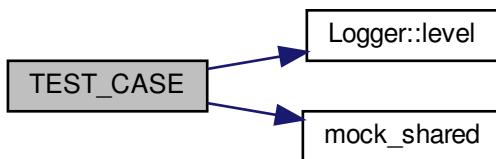
16.207.1.2 TEST_CASE() [2/5]

```
TEST_CASE (
    "ConnectionPool's can store at least one connection and send a " "packet over it." ,
    "" [ConnectionPool] )
```

Definition at line 39 of file [test_ConnectionPool.cpp](#).

References [Logger::level\(\)](#), [mock_shared\(\)](#), and [ping](#).

Here is the call graph for this function:



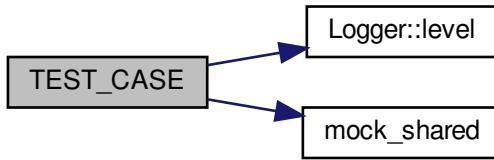
16.207.1.3 TEST_CASE() [3/5]

```
TEST_CASE (
    "ConnectionPool's can store more than one connection and send a \"packet over them."
,
    "" [ConnectionPool] )
```

Definition at line 83 of file [test_ConnectionPool.cpp](#).

References [Logger::level\(\)](#), [mock_shared\(\)](#), and [ping](#).

Here is the call graph for this function:



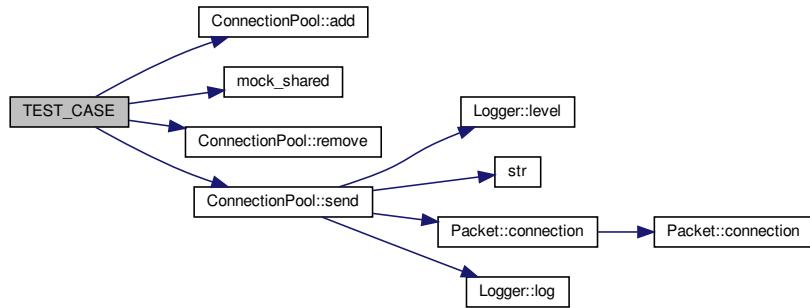
16.207.1.4 TEST_CASE() [4/5]

```
TEST_CASE (
    "ConnectionPool's 'remove' method removes a connection." ,
    "" [ConnectionPool] )
```

Definition at line 140 of file [test_ConnectionPool.cpp](#).

References [ConnectionPool::add\(\)](#), [mock_shared\(\)](#), [ConnectionPool::remove\(\)](#), and [ConnectionPool::send\(\)](#).

Here is the call graph for this function:



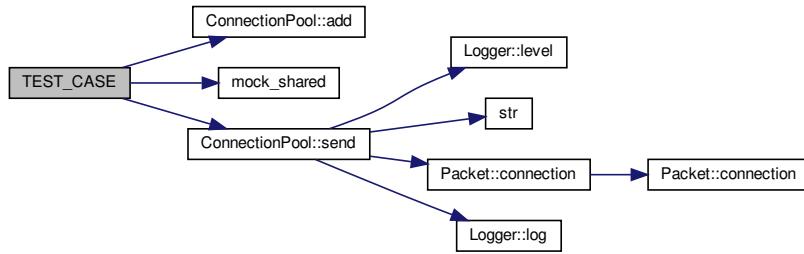
16.207.1.5 TEST_CASE() [5/5]

```
TEST_CASE (
    "ConnectionPool's 'send' method removes connections that have \"expired.",
    "" [ConnectionPool] )
```

Definition at line 167 of file [test_ConnectionPool.cpp](#).

References [ConnectionPool::add\(\)](#), [mock_shared\(\)](#), and [ConnectionPool::send\(\)](#).

Here is the call graph for this function:



16.208 test_ConnectionPool.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <memory>
00019
00020 #include <catch.hpp>
00021 #include <fakeit.hpp>
00022
00023 #include "Connection.hpp"
00024 #include "ConnectionPool.hpp"
00025 #include "Logger.hpp"
00026 #include "PacketVersion2.hpp"
00027 #include "utility.hpp"
00028
00029 #include "common.hpp"
00030 #include "common_Packet.hpp"
00031
00032
00033 TEST_CASE("ConnectionPool's can be constructed.", "[ConnectionPool]")
00034 {
00035     REQUIRE_NO_THROW(ConnectionPool());
}
```

```
00036 }
00037
00038
00039 TEST_CASE("ConnectionPool's can store at least one connection and send a "
00040         "packet over it.", "[ConnectionPool]")
00041 {
00042     Logger::level(1);
00043     auto packet = std::make_unique<packet_v2::Packet>(to_vector(PingV2()));
00044     fakeit::Mock<Connection> mock;
00045     fakeit::Fake(Method(mock, send));
00046     std::shared_ptr<Connection> connection = mock_shared(mock);
00047     fakeit::Mock<Filter> mock_filter;
00048     auto filter = mock_shared(mock_filter);
00049     auto source_connection = std::make_shared<Connection>("SOURCE", filter);
00050     auto ping = std::make_unique<packet_v2::Packet>(to_vector(PingV2()));
00051     ping->connection(source_connection);
00052     ConnectionPool pool;
00053     SECTION("with logging")
00054     {
00055         Logger::level(2);
00056         MockCOut mock_cout;
00057         pool.add(connection);
00058         pool.send(std::move(ping));
00059         fakeit::Verify(Method(mock, send).Matching([&](auto a)
00060         {
00061             return *a == *packet;
00062         })).Once();
00063         REQUIRE(
00064             mock_cout.buffer().substr(21) ==
00065             "received PING (#4) from 192.168 to 127.1 (v2.0) "
00066             "source SOURCE\n");
00067     }
00068     SECTION("without logging")
00069     {
00070         MockCOut mock_cout;
00071         pool.add(connection);
00072         pool.send(std::move(ping));
00073         fakeit::Verify(Method(mock, send).Matching([&](auto a)
00074         {
00075             return *a == *packet;
00076         })).Once();
00077         REQUIRE(mock_cout.buffer().empty());
00078     }
00079     Logger::level(0);
00080 }
00081
00082
00083 TEST_CASE("ConnectionPool's can store more than one connection and send a "
00084         "packet over them.", "[ConnectionPool]")
00085 {
00086     Logger::level(1);
00087     auto packet = std::make_unique<packet_v2::Packet>(to_vector(PingV2()));
00088     fakeit::Mock<Connection> mock1;
00089     fakeit::Mock<Connection> mock2;
00090     fakeit::Fake(Method(mock1, send));
00091     fakeit::Fake(Method(mock2, send));
00092     std::shared_ptr<Connection> connection1 = mock_shared(mock1);
00093     std::shared_ptr<Connection> connection2 = mock_shared(mock2);
00094     fakeit::Mock<Filter> mock_filter;
00095     auto filter = mock_shared(mock_filter);
00096     auto source_connection = std::make_shared<Connection>("SOURCE", filter);
00097     auto ping = std::make_unique<packet_v2::Packet>(to_vector(PingV2()));
00098     ping->connection(source_connection);
00099     ConnectionPool pool;
00100     SECTION("with logging")
00101     {
00102         Logger::level(2);
00103         MockCOut mock_cout;
00104         pool.add(connection1);
00105         pool.add(connection2);
00106         pool.send(std::move(ping));
00107         fakeit::Verify(Method(mock1, send).Matching([&](auto a)
00108         {
00109             return *a == *packet;
00110         })).Once();
00111         fakeit::Verify(Method(mock2, send).Matching([&](auto a)
00112         {
00113             return *a == *packet;
00114         })).Once();
00115         REQUIRE(
00116             mock_cout.buffer().substr(21) ==
```

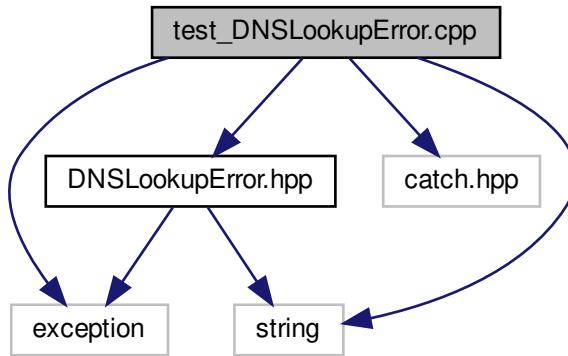
```

00175         "received PING (#4) from 192.168 to 127.1 (v2.0) "
00176         "source SOURCE\n");
00177     }
00178     SECTION("without logging")
00179     {
00180         MockCOut mock_cout;
00181         pool.add(connection1);
00182         pool.add(connection2);
00183         pool.send(std::move(ping));
00184         fakeit::Verify(Method(mock1, send).Matching([&](auto a)
00185         {
00186             return *a == *packet;
00187         }).Once());
00188         fakeit::Verify(Method(mock2, send).Matching([&](auto a)
00189         {
00190             return *a == *packet;
00191         }).Once());
00192         REQUIRE(mock_cout.buffer().empty());
00193     }
00194     Logger::level(0);
00195 }
00196
00197
00198
00199
00200 TEST_CASE("ConnectionPool's 'remove' method removes a connection.",
00201     "[ConnectionPool]")
00202 {
00203     auto packet = std::make_unique<packet_v2::Packet>(to_vector(PingV2()));
00204     fakeit::Mock<Connection> mock1;
00205     fakeit::Mock<Connection> mock2;
00206     fakeit::Fake(Method(mock1, send));
00207     fakeit::Fake(Method(mock2, send));
00208     std::shared_ptr<Connection> connection1 = mock_shared(mock1);
00209     std::shared_ptr<Connection> connection2 = mock_shared(mock2);
00210     ConnectionPool pool;
00211     pool.add(connection1);
00212     pool.add(connection2);
00213     pool.send(std::make_unique<packet_v2::Packet>(to_vector(PingV2())));
00214     pool.remove(connection1);
00215     pool.send(std::make_unique<packet_v2::Packet>(to_vector(PingV2())));
00216     fakeit::Verify(Method(mock1, send).Matching([&](auto a)
00217     {
00218         return *a == *packet;
00219     }).Once());
00220     fakeit::Verify(Method(mock2, send).Matching([&](auto a)
00221     {
00222         return *a == *packet;
00223     }).Exactly(2));
00224 }
00225
00226
00227
00228
00229 TEST_CASE("ConnectionPool's 'send' method removes connections that have "
00230     "expired.", "[ConnectionPool]")
00231 {
00232     auto packet = std::make_unique<packet_v2::Packet>(to_vector(PingV2()));
00233     fakeit::Mock<Connection> mock1;
00234     fakeit::Mock<Connection> mock2;
00235     fakeit::Fake(Method(mock1, send));
00236     fakeit::Fake(Method(mock2, send));
00237     std::shared_ptr<Connection> connection1 = mock_shared(mock1);
00238     std::shared_ptr<Connection> connection2 = mock_shared(mock2);
00239     ConnectionPool pool;
00240     pool.add(connection1);
00241     pool.add(connection2);
00242     pool.send(std::make_unique<packet_v2::Packet>(to_vector(PingV2())));
00243     connection1.reset();
00244     pool.send(std::make_unique<packet_v2::Packet>(to_vector(PingV2())));
00245     fakeit::Verify(Method(mock1, send).Matching([&](auto a)
00246     {
00247         return *a == *packet;
00248     }).Once());
00249     fakeit::Verify(Method(mock2, send).Matching([&](auto a)
00250     {
00251         return *a == *packet;
00252     }).Exactly(2));
00253 }

```

16.209 test_DNSLookupError.cpp File Reference

```
#include <exception>
#include <string>
#include <catch.hpp>
#include "DNSLookupError.hpp"
Include dependency graph for test_DNSLookupError.cpp:
```



Functions

- [TEST_CASE \("DNSLookupError's can be thrown.", "\[DNSLookupError\]"\)](#)
- [TEST_CASE \("The 'what' method gives the unresolved hostname.", "\[DNSLookupError\]"\)](#)

16.209.1 Function Documentation

16.209.1.1 TEST_CASE() [1/2]

```
TEST_CASE (
    "DNSLookupError's can be thrown." ,
    "" [DNSLookupError] )
```

Definition at line 32 of file [test_DNSLookupError.cpp](#).

16.209.1.2 TEST_CASE() [2/2]

```
TEST_CASE (
    "The 'what' method gives the unresolved hostname." ,
    "" [DNSLookupError] )
```

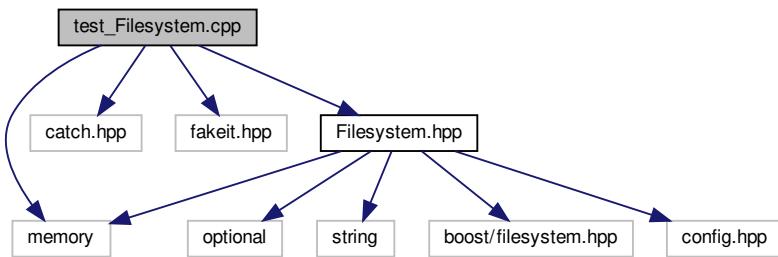
Definition at line 49 of file [test_DNSLookupError.cpp](#).

16.210 test_DNSLookupError.cpp

```
00001 // MAVLINK router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <exception>
00019 #include <string>
00020
00021 #include <catch.hpp>
00022
00023 #include "DNSLookupError.hpp"
00024
00025
00026 [[noreturn]] static void throw_lookup_error(std::string hostname)
00027 {
00028     throw DNSLookupError(hostname);
00029 }
00030
00031
00032 TEST_CASE("DNSLookupError's can be thrown.", "[DNSLookupError]")
00033 {
00034     SECTION("And can be caught.")
00035     {
00036         REQUIRE_THROWS(throw_lookup_error("example.com"));
00037     }
00038     SECTION("And can be caught as DNSLookupError.")
00039     {
00040         REQUIRE_THROWS_AS(throw_lookup_error("example.com"), DNSLookupError);
00041     }
00042     SECTION("And can be caught as std::exception.")
00043     {
00044         REQUIRE_THROWS_AS(throw_lookup_error("example.com"), std::exception);
00045     }
00046 }
00047
00048
00049 TEST_CASE("The 'what' method gives the unresolved hostname.",
00050             "[DNSLookupError]")
00051 {
00052     REQUIRE_THROWS_WITH(
00053         throw_lookup_error("example.com"),
00054         "DNSLookupError: Could not find an IP address for \\"example.com\\"");
00055 }
00056
00057
00058 // Required for complete function coverage.
00059 TEST_CASE("Run dynamic destructors (DNSLookupError).", "[DNSLookupError]")
00060 {
00061     DNSLookupError *dns = nullptr;
00062     REQUIRE_NO_THROW(dns = new DNSLookupError("error"));
00063     REQUIRE_NO_THROW(delete dns);
00064 }
```

16.211 test_Filesystem.cpp File Reference

```
#include <memory>
#include <catch.hpp>
#include <fakeit.hpp>
#include "Filesystem.hpp"
Include dependency graph for test_Filesystem.cpp:
```



Functions

- [TEST_CASE](#) ("Filesystem's 'exist' method determines the existence of a file.", "[Filesystem]")

16.211.1 Function Documentation

16.211.1.1 TEST_CASE()

```
TEST_CASE (
    "Filesystem's 'exist' method determines the existence of a file." ,
    "" [Filesystem] )
```

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

16.212 test_Filesystem.cpp

```

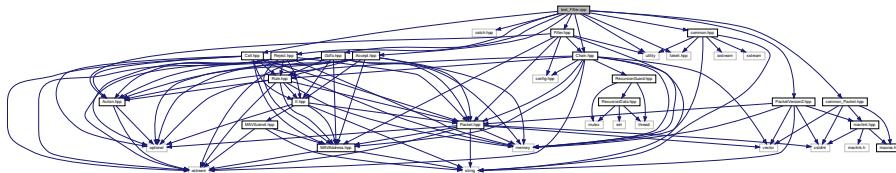
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <memory>
00019
00020 #include <catch.hpp>
00021 #include <fakeit.hpp>
00022
00023 #include "Filesystem.hpp"
00024
00025
00026 TEST_CASE("Filesystem's 'exist' method determines the existence of a file.",
00027             "[Filesystem]")
00028 {
00029     // Note: both compile time and dynamic construction used for complete
00030     // coverage.
00031     SECTION("Returns 'true' when the file exists.")
00032     {
00033         Filesystem filesystem;
00034         REQUIRE(filesystem.exists("test/mavtables.conf"));
00035     }
00036     SECTION("Returns 'false' when the file does not exist.")
00037     {
00038         auto filesystem = std::make_unique<Filesystem>();
00039         REQUIRE_FALSE(
00040             filesystem->exists("file_that_does_not_exist.conf"));
00041     }
00042 }
```

16.213 test_Filter.cpp File Reference

```

#include <utility>
#include <catch.hpp>
#include <fakeit.hpp>
#include "Accept.hpp"
#include "Call.hpp"
#include "Chain.hpp"
#include "Filter.hpp"
#include "GoTo.hpp"
#include "PacketVersion2.hpp"
#include "Reject.hpp"
#include "common.hpp"
#include "common_Packet.hpp"
```

Include dependency graph for test_Filter.cpp:



Functions

- `TEST_CASE ("Filter's are constructable.", "[Filter]")`
- `TEST_CASE ("Filter's are comparable.", "[Filter]")`
- `TEST_CASE ("Filter's are copyable.", "[Filter]")`
- `TEST_CASE ("Filter's are movable.", "[Filter]")`
- `TEST_CASE ("Filter's are assignable.", "[Filter]")`
- `TEST_CASE ("Filter's 'will_accept' method determines whether to accept or "reject a packet/address combination.", "[Filter]")`

Variables

- `auto filter_b = Filter(Chain("test_chain_b"))`
- `filter_a = std::move(filter_b)`

16.213.1 Function Documentation

16.213.1.1 TEST_CASE() [1/6]

```
TEST_CASE (
    "Filter's are constructable." ,
    "" [Filter] )
```

Definition at line 35 of file [test_Filter.cpp](#).

16.213.1.2 TEST_CASE() [2/6]

```
TEST_CASE (
    "Filter's are comparable." ,
    "" [Filter] )
```

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

16.213.1.3 TEST_CASE() [3/6]

```
TEST_CASE (
    "Filter's are copyable." ,
    "" [Filter] )
```

Definition at line [77](#) of file [test_Filter.cpp](#).

16.213.1.4 TEST_CASE() [4/6]

```
TEST_CASE (
    "Filter's are movable." ,
    "" [Filter] )
```

Definition at line [85](#) of file [test_Filter.cpp](#).

16.213.1.5 TEST_CASE() [5/6]

```
TEST_CASE (
    "Filter's are assignable." ,
    "" [Filter] )
```

Definition at line [93](#) of file [test_Filter.cpp](#).

References [filter_a](#), and [filter_b](#).

16.213.1.6 TEST_CASE() [6/6]

```
TEST_CASE (
    "Filter's 'will_accept' method determines whether to accept or " "reject a packet/address
combination." ,
    "" [Filter] )
```

Definition at line [113](#) of file [test_Filter.cpp](#).

References [ping](#).

16.213.2 Variable Documentation

16.213.2.1 filter_a

```
filter_a = std::move(filter_b)
```

Definition at line 108 of file [test_Filter.cpp](#).

16.213.2.2 filter_b

```
auto filter_b = Filter(Chain("test_chain_b"))
```

Definition at line 106 of file [test_Filter.cpp](#).

16.214 test_Filter.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <utility>
00019
00020 #include <catch.hpp>
00021 #include <fakeit.hpp>
00022
00023 #include "Accept.hpp"
00024 #include "Call.hpp"
00025 #include "Chain.hpp"
00026 #include "Filter.hpp"
00027 #include "GoTo.hpp"
00028 #include "PacketVersion2.hpp"
00029 #include "Reject.hpp"
00030
00031 #include "common.hpp"
00032 #include "common_Packet.hpp"
00033
00034
00035 TEST_CASE("Filter's are constructable.", "[Filter]")
00036 {
00037     Chain chain("test_chain");
00038     SECTION("Without default accept.")
00039     {
00040         REQUIRE_NO_THROW(Filter(chain));
00041     }
00042     SECTION("With default accept.")
00043     {
00044         REQUIRE_NO_THROW(Filter(chain, true));
00045     }
00046 }
00047
00048 TEST_CASE("Filter's are comparable.", "[Filter]")
00049
00050 {
```

```

00051     Chain chain1("test_chain_1");
00052     Chain chain2("test_chain_2");
00053     SECTION("with ==")
00054     {
00055         REQUIRE(Filter(chain1) == Filter(chain1));
00056         REQUIRE(Filter(chain2) == Filter(chain2));
00057         // Reject is default.
00058         REQUIRE(Filter(chain1, false) == Filter(chain1));
00059         REQUIRE(Filter(chain1, true) == Filter(chain1, true));
00060         REQUIRE(Filter(chain1, false) == Filter(chain1, false));
00061         REQUIRE_FALSE(Filter(chain1) == Filter(chain2));
00062         REQUIRE_FALSE(Filter(chain1, true) == Filter(chain1));
00063     }
00064     SECTION("with !=")
00065     {
00066         REQUIRE(Filter(chain1) != Filter(chain2));
00067         REQUIRE(Filter(chain1, true) != Filter(chain1));
00068         REQUIRE_FALSE(Filter(chain1) != Filter(chain1));
00069         REQUIRE_FALSE(Filter(chain2) != Filter(chain2));
00070         REQUIRE_FALSE(Filter(chain1, false) != Filter(chain1));
00071         REQUIRE_FALSE(Filter(chain1, true) != Filter(chain1, true));
00072         REQUIRE_FALSE(Filter(chain1, false) != Filter(chain1, false));
00073     }
00074 }
00075
00076
00077 TEST_CASE("Filter's are copyable.", "[Filter]")
00078 {
00079     auto original = Filter(Chain("test_chain"));
00080     auto copy(original);
00081     REQUIRE(copy == Filter(Chain("test_chain")));
00082 }
00083
00084
00085 TEST_CASE("Filter's are movable.", "[Filter]")
00086 {
00087     auto original = Filter(Chain("test_chain"));
00088     auto moved(std::move(original));
00089     REQUIRE(moved == Filter(Chain("test_chain")));
00090 }
00091
00092
00093 TEST_CASE("Filter's are assignable.", "[Filter]")
00094 {
00095     auto filter_a = Filter(Chain("test_chain_a"));
00096     auto filter_b = Filter(Chain("test_chain_b"));
00097     REQUIRE(filter_a == Filter(Chain("test_chain_a")));
00098     filter_a = filter_b;
00099     REQUIRE(filter_a == Filter(Chain("test_chain_b")));
00100 }
00101
00102
00103 TEST_CASE("Filter's are assignable (by move semantics).", "[Filter]")
00104 {
00105     auto filter_a = Filter(Chain("test_chain_a"));
00106     auto filter_b = Filter(Chain("test_chain_b"));
00107     REQUIRE(filter_a == Filter(Chain("test_chain_a")));
00108     filter_a = std::move(filter_b);
00109     REQUIRE(filter_a == Filter(Chain("test_chain_b")));
00110 }
00111
00112
00113 TEST_CASE("Filter's 'will_accept' method determines whether to accept or "
00114             "reject a packet/address combination.", "[Filter]")
00115 {
00116     auto ping = packet_v2::Packet(to_vector(PingV2()));
00117     SECTION("Accept packet, default priority.")
00118     {
00119         Chain chain("test_chain");
00120         chain.append(std::make_unique<Accept>());
00121         REQUIRE(
00122             Filter(chain).will_accept(ping, MAVAddress("192.168")) ==
00123             std::make_pair(true, 0));
00124     }
00125     SECTION("Accept packet, with priority.")
00126     {
00127         Chain chain("test_chain");
00128         chain.append(std::make_unique<Accept>(3));
00129         REQUIRE(
00130             Filter(chain).will_accept(ping, MAVAddress("192.168")) ==
00131             std::make_pair(true, 3));

```

```

00132     }
00133     SECTION("Reject packet.")
00134     {
00135         Chain chain("test_chain");
00136         chain.append(std::make_unique<Reject>());
00137         REQUIRE_FALSE(
00138             Filter(chain).will_accept(ping, MAVAddress("192.168")).first);
00139     }
00140     SECTION("Default action.")
00141     {
00142         auto subchain = std::make_shared<Chain>("test_subchain");
00143         Chain chain("test_chain");
00144         chain.append(std::make_unique<GoTo>(subchain));
00145         REQUIRE_FALSE(
00146             Filter(chain).will_accept(ping, MAVAddress("192.168")).first);
00147         REQUIRE(
00148             Filter(chain, true).will_accept(ping, MAVAddress("192.168")) ==
00149             std::make_pair(true, 0));
00150     }
00151     SECTION("Undecided action.")
00152     {
00153         auto subchain = std::make_shared<Chain>("test_subchain");
00154         Chain chain("test_chain");
00155         chain.append(std::make_unique<Call>(subchain));
00156         REQUIRE_FALSE(
00157             Filter(chain).will_accept(ping, MAVAddress("192.168")).first);
00158         REQUIRE(
00159             Filter(chain, true).will_accept(ping, MAVAddress("192.168")) ==
00160             std::make_pair(true, 0));
00161     }
00162 }

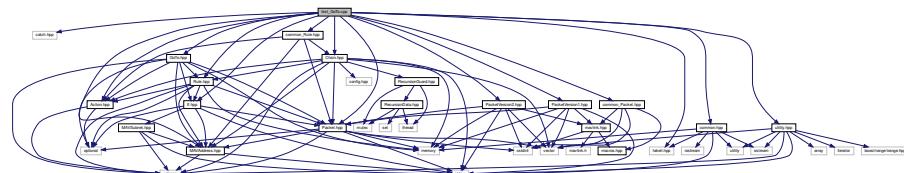
```

16.215 test_GoTo.cpp File Reference

```

#include <catch.hpp>
#include <fakeit.hpp>
#include "Action.hpp"
#include "Chain.hpp"
#include "GoTo.hpp"
#include "If.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "PacketVersion1.hpp"
#include "PacketVersion2.hpp"
#include "Rule.hpp"
#include "utility.hpp"
#include "common.hpp"
#include "common_Packet.hpp"
#include "common_Rule.hpp"
Include dependency graph for test_GoTo.cpp:

```



Functions

- TEST_CASE ("GoTo's are constructable.", "[GoTo]")

- [TEST_CASE \("GoTo's are comparable.", "\[GoTo\]"\)](#)
- [TEST_CASE \("GoTo's 'action' method determines what to do with a " "packet/address combination.", "\[GoTo\]"\)](#)
- [TEST_CASE \("GoTo's are printable \(without a condition but with a priority\).", "\[GoTo\]"\)](#)
- [TEST_CASE \("GoTo's are printable \(with a condition but without a priority\).", "\[GoTo\]"\)](#)
- [GoTo goto_ \(chain, -3, If\(\).type\("PING"\).from\("192.168/8"\).to\("172.16/4"\)\)](#)
- [TEST_CASE \("GoTo's 'clone' method returns a polymorphic copy.", "\[GoTo\]"\)](#)

Variables

- [auto ping = packet_v2::Packet\(to_vector\(PingV2\(\)\)\)](#)
- [GoTo goto_ \(chain\)](#)
- [Rule & rule = goto_](#)

16.215.1 Function Documentation

16.215.1.1 goto_()

```
GoTo goto_ (
    chain ,
    - 3,
    If().type ("PING") .from ("192.168/8") .to ("172.16/4") )
```

16.215.1.2 TEST_CASE() [1/6]

```
TEST_CASE (
    "GoTo's are constructable." ,
    "" [GoTo] )
```

Definition at line 37 of file [test_GoTo.cpp](#).

References [mock_shared\(\)](#).

Here is the call graph for this function:



16.215.1.3 TEST_CASE() [2/6]

```
TEST_CASE (
    "GoTo's are comparable." ,
    "" [GoTo] )
```

Definition at line 80 of file [test_GoTo.cpp](#).

References [mock_shared\(\)](#).

Here is the call graph for this function:



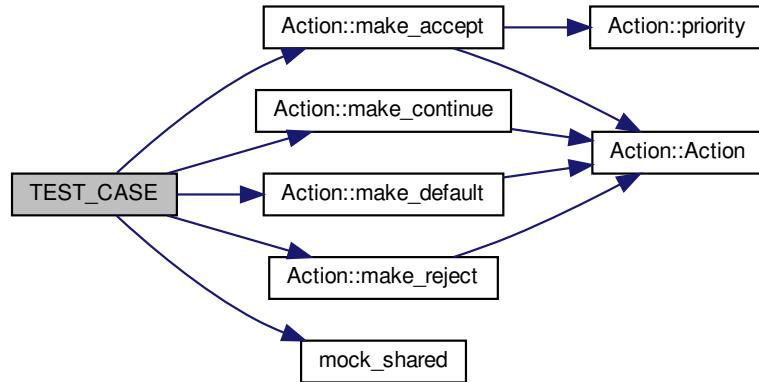
16.215.1.4 TEST_CASE() [3/6]

```
TEST_CASE (
    "GoTo's 'action' method determines what to do with a \"packet/address combination."
    ,
    "" [GoTo] )
```

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

References [Action::make_accept\(\)](#), [Action::make_continue\(\)](#), [Action::make_default\(\)](#), [Action::make_reject\(\)](#), [mock_shared\(\)](#), and [ping](#).

Here is the call graph for this function:



16.215.1.5 TEST_CASE() [4/6]

```
TEST_CASE (
    "GoTo's are printable (without a condition but with a priority).",
    "" [GoTo] )
```

Definition at line 243 of file [test_GoTo.cpp](#).

References [goto_](#), [ping](#), and [rule](#).

16.215.1.6 TEST_CASE() [5/6]

```
TEST_CASE (
    "GoTo's are printable (with a condition but without a priority).",
    "" [GoTo] )
```

Definition at line 261 of file [test_GoTo.cpp](#).

References [goto_](#), [ping](#), and [rule](#).

16.215.1.7 TEST_CASE() [6/6]

```
TEST_CASE (
    "GoTo's 'clone' method returns a polymorphic copy.",
    "" [GoTo] )
```

Definition at line 304 of file [test_GoTo.cpp](#).

16.215.2 Variable Documentation

16.215.2.1 goto_

```
GoTo goto_(chain, -3, If().type("PING").from("192.168/8").to("172.16/4")) (
    chain )
```

16.215.2.2 ping

```
auto ping = packet_v2::Packet(to_vector(PingV2()))
```

Definition at line 229 of file [test_GoTo.cpp](#).

16.215.2.3 rule

```
Rule& rule = goto_
```

Definition at line 231 of file [test_GoTo.cpp](#).

16.216 test_GoTo.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <catch.hpp>
00019 #include <fakeit.hpp>
00020
00021 #include "Action.hpp"
00022 #include "Chain.hpp"
00023 #include "GoTo.hpp"
00024 #include "If.hpp"
00025 #include "MAVAddress.hpp"
00026 #include "Packet.hpp"
00027 #include "PacketVersion1.hpp"
00028 #include "PacketVersion2.hpp"
00029 #include "Rule.hpp"
00030 #include "utility.hpp"
00031
00032 #include "common.hpp"
00033 #include "common_Packet.hpp"
00034 #include "common_Rule.hpp"
00035
00036
00037 TEST_CASE("GoTo's are constructable.", "[GoTo]")
00038 {
00039     fakeit::Mock<Chain> mock;
00040     std::shared_ptr<Chain> chain = mock_shared(mock);
00041     SECTION("Without a condition (match all packet/address combinations) or a "
00042             "priority.")
00043     {
00044         REQUIRE_NO_THROW(GoTo(chain));
00045     }
00046     SECTION("Without a condition (match all packet/address combinations) but "
00047             "with a priority.")
00048     {
00049         REQUIRE_NO_THROW(GoTo(chain, 3));
00050     }
```

```

00051     SECTION("With a condition and without a priority.")
00052     {
00053         REQUIRE_NO_THROW(GoTo(chain, If()));
00054         REQUIRE_NO_THROW(GoTo(chain, If().type("PING")));
00055         REQUIRE_NO_THROW(GoTo(chain, If().from("192.168")));
00056         REQUIRE_NO_THROW(GoTo(chain, If().to("172.16")));
00057     }
00058     SECTION("With both a condition and a priority.")
00059     {
00060         REQUIRE_NO_THROW(GoTo(chain, 3, If()));
00061         REQUIRE_NO_THROW(GoTo(chain, 3, If().type("PING")));
00062         REQUIRE_NO_THROW(GoTo(chain, 3, If().from("192.168")));
00063         REQUIRE_NO_THROW(GoTo(chain, 3, If().to("172.16")));
00064     }
00065     SECTION("Ensures the chain's shared pointer is not null.")
00066     {
00067         REQUIRE_THROWS_AS(GoTo(nullptr), std::invalid_argument);
00068         REQUIRE_THROWS_AS(GoTo(nullptr, 3), std::invalid_argument);
00069         REQUIRE_THROWS_AS(
00070             GoTo(nullptr, 3, If().type("PING")), std::invalid_argument);
00071         REQUIRE_THROWS_WITH(GoTo(nullptr), "Given chain pointer is null.");
00072         REQUIRE_THROWS_WITH(GoTo(nullptr, 3), "Given chain pointer is null.");
00073         REQUIRE_THROWS_WITH(
00074             GoTo(nullptr, 3, If().type("PING")),
00075             "Given chain pointer is null.");
00076     }
00077 }
00078
00079
00080 TEST_CASE("GoTo's are comparable.", "[GoTo]")
00081 {
00082     fakeit::Mock<Chain> mock1;
00083     fakeit::Mock<Chain> mock2;
00084     std::shared_ptr<Chain> chain1 = mock_shared(mock1);
00085     std::shared_ptr<Chain> chain2 = mock_shared(mock2);
00086     SECTION("with ==")
00087     {
00088         REQUIRE(GoTo(chain1) == GoTo(chain1));
00089         REQUIRE(
00090             GoTo(chain1, If().type("PING")) == GoTo(chain1, If().type("PING")));
00091         REQUIRE(GoTo(chain1, 3) == GoTo(chain1, 3));
00092         REQUIRE(
00093             GoTo(chain1, 3, If().type("PING")) ==
00094             GoTo(chain1, 3, If().type("PING")));
00095         REQUIRE_FALSE(GoTo(chain1) == GoTo(chain2));
00096         REQUIRE_FALSE(
00097             GoTo(chain1, If().type("PING")) ==
00098             GoTo(chain1, If().type("SET_MODE")));
00099         REQUIRE_FALSE(GoTo(chain1, If().type("PING")) == GoTo(chain1,
If()));
00100         REQUIRE_FALSE(GoTo(chain1, If().type("PING")) == GoTo(chain1));
00101         REQUIRE_FALSE(GoTo(chain1, 3) == GoTo(chain1, -3));
00102         REQUIRE_FALSE(GoTo(chain1, 3) == GoTo(chain1));
00103     }
00104     SECTION("with !=")
00105     {
00106         REQUIRE(GoTo(chain1, If().type("PING")) != GoTo(chain1));
00107         REQUIRE(GoTo(chain1, If().type("PING")) != GoTo(chain1, If()));
00108         REQUIRE(
00109             GoTo(chain1, If().type("PING")) !=
00110             GoTo(chain1, If().type("SET_MODE")));
00111         REQUIRE(GoTo(chain1, 3) != GoTo(chain1, -3));
00112         REQUIRE(GoTo(chain1, 3) != GoTo(chain1));
00113         REQUIRE(GoTo(chain1) != GoTo(chain2));
00114         REQUIRE_FALSE(GoTo(chain1) != GoTo(chain1));
00115         REQUIRE_FALSE(
00116             GoTo(chain1, If().type("PING")) != GoTo(chain1, If().type("PING")));
00117         REQUIRE_FALSE(GoTo(chain1, 3) != GoTo(chain1, 3));
00118         REQUIRE_FALSE(
00119             GoTo(chain1, 3, If().type("PING")) !=
00120             GoTo(chain1, 3, If().type("PING")));
00121     }
00122 }
00123
00124
00125 TEST_CASE("GoTo's 'action' method determines what to do with a "
00126             "packet/address combination.", "[GoTo]")
00127 {
00128     fakeit::Mock<Chain> accept_mock;
00129     fakeit::When(Method(accept_mock, action)).AlwaysReturn(
00130         Action::make_accept());

```

```

00131     std::shared_ptr<Chain> accept_chain = mock_shared(accept_mock);
00132     fakeit::Mock<Chain> reject_mock;
00133     fakeit::When(Method(reject_mock, action)).AlwaysReturn(
00134         Action::make_reject());
00135     std::shared_ptr<Chain> reject_chain = mock_shared(reject_mock);
00136     fakeit::Mock<Chain> continue_mock;
00137     fakeit::When(Method(continue_mock, action)).AlwaysReturn(
00138         Action::make_continue());
00139     std::shared_ptr<Chain> continue_chain = mock_shared(continue_mock);
00140     fakeit::Mock<Chain> default_mock;
00141     fakeit::When(Method(default_mock, action)).AlwaysReturn(
00142         Action::make_default());
00143     std::shared_ptr<Chain> default_chain = mock_shared(default_mock);
00144     fakeit::Mock<Chain> accept10_mock;
00145     fakeit::When(Method(accept10_mock, action)).AlwaysReturn(
00146         Action::make_accept(10));
00147     std::shared_ptr<Chain> accept10_chain = mock_shared(accept10_mock);
00148     auto ping = packet_v2::Packet(to_vector(PingV2()));
00149     SECTION("Check call to chain's action method.")
00150     {
00151         REQUIRE(GoTo(std::make_shared<TestChain>()).action(
00152             ping, MAVAddress("192.168")) ==
00153             Action::make_accept());
00154         fakeit::Mock<Chain> mock;
00155         fakeit::When(Method(mock, action)).AlwaysReturn(Action::make_accept());
00156         std::shared_ptr<Chain> chain = mock_shared(mock);
00157         MAVAddress address("192.168");
00158         GoTo(chain).action(ping, address);
00159         fakeit::Verify(
00160             Method(mock, action).Matching([&](auto & a, auto & b)
00161             {
00162                 return a == ping && b == MAVAddress("192.168");
00163             })).Once();
00164     }
00165     SECTION("Delegates to the contained chain if there is no conditional.")
00166     {
00167         // Without priority.
00168         REQUIRE(
00169             GoTo(accept_chain).action(ping, MAVAddress("192.168")) ==
00170             Action::make_accept());
00171         REQUIRE(
00172             GoTo(reject_chain).action(ping, MAVAddress("192.168")) ==
00173             Action::make_reject());
00174         REQUIRE(
00175             GoTo(continue_chain).action(ping, MAVAddress("192.168")) ==
00176             Action::make_default());
00177         REQUIRE(
00178             GoTo(default_chain).action(ping, MAVAddress("192.168")) ==
00179             Action::make_default());
00180         // With priority (adds priority).
00181         REQUIRE(
00182             GoTo(accept_chain, 3).action(ping, MAVAddress("192.168")) ==
00183             Action::make_accept(3));
00184         // Priority already set (no override).
00185         REQUIRE(
00186             GoTo(accept10_chain, 3).action(ping, MAVAddress("192.168")) ==
00187             Action::make_accept(10));
00188     }
00189     SECTION("Delegates to the contained chain if the conditional is a match.")
00190     {
00191         // Without priority.
00192         REQUIRE(
00193             GoTo(accept_chain, If().to("192.168")).action(
00194                 ping, MAVAddress("192.168")) ==
00195                 Action::make_accept());
00196         REQUIRE(
00197             GoTo(reject_chain, If().to("192.168")).action(
00198                 ping, MAVAddress("192.168")) ==
00199                 Action::make_reject());
00200         REQUIRE(
00201             GoTo(continue_chain, If().to("192.168")).action(
00202                 ping, MAVAddress("192.168")) ==
00203                 Action::make_default());
00204         // With priority (adds priority).
00205         REQUIRE(
00206             GoTo(accept_chain, 3, If().to("192.168")).action(
00207                 ping, MAVAddress("192.168")) ==

```

```

00207     Action::make_accept(3));
00208     // Priority already set (no override).
00209     REQUIRE(
00210         GoTo(accept10_chain, 3, If().to("192.168")).action(
00211             ping, MAVAddress("192.168")) ==
00212             Action::make_accept(10));
00213     }
00214     SECTION("Returns the continue action if the conditional does not match.")
00215     {
00216         // Without priority.
00217         REQUIRE(
00218             GoTo(accept_chain, If().to("172.16")).action(
00219                 ping, MAVAddress("192.168")) ==
00220             Action::make_continue());
00221         // With priority.
00222         REQUIRE(
00223             GoTo(accept_chain, 3, If().to("172.16")).action(
00224                 ping, MAVAddress("192.168")) ==
00225             Action::make_continue());
00226 TEST_CASE("GoTo's are printable (without a condition or a priority).", "[GoTo]")
00227 {
00228     auto chain = std::make_shared<TestChain>();
00229     auto ping = packet_v2::Packet(to_vector(PingV2()));
00230     GoTo goto_(chain);
00231     Rule &rule = goto_;
00232     SECTION("By direct type.")
00233     {
00234         REQUIRE(str(goto_) == "goto test_chain");
00235     }
00236     SECTION("By polymorphic type.")
00237     {
00238         REQUIRE(str(rule) == "goto test_chain");
00239     }
00240 }
00241
00242
00243 TEST_CASE("GoTo's are printable (without a condition but with a priority).",
00244             "[GoTo]")
00245 {
00246     auto chain = std::make_shared<TestChain>();
00247     auto ping = packet_v2::Packet(to_vector(PingV2()));
00248     GoTo goto_(chain, -3);
00249     Rule &rule = goto_;
00250     SECTION("By direct type.")
00251     {
00252         REQUIRE(str(goto_) == "goto test_chain with priority -3");
00253     }
00254     SECTION("By polymorphic type.")
00255     {
00256         REQUIRE(str(rule) == "goto test_chain with priority -3");
00257     }
00258 }
00259
00260
00261 TEST_CASE("GoTo's are printable (with a condition but without a priority).",
00262             "[GoTo]")
00263 {
00264     auto chain = std::make_shared<TestChain>();
00265     auto ping = packet_v2::Packet(to_vector(PingV2()));
00266     GoTo goto_(chain, If().type("PING").from("192.168/8").to("172.16/4"));
00267     Rule &rule = goto_;
00268     SECTION("By direct type.")
00269     {
00270         REQUIRE(
00271             str(goto_) == "goto test_chain if PING from 192.168/8 to 172.16/4");
00272     }
00273     SECTION("By polymorphic type.")
00274     {
00275         REQUIRE(
00276             str(rule) == "goto test_chain if PING from 192.168/8 to 172.16/4");
00277     }
00278 }
00279
00280
00281 TEST_CASE("GoTo's are printable (with a condition and a priority).", "[GoTo]")
00282 {
00283     auto chain = std::make_shared<TestChain>();

```

```

00284     auto ping = packet_v2::Packet(to_vector(PingV2()));
00285     GoTo goto_(chain, -3, If().type("PING").from("192.168/8").to("172.16/4"));
00286     Rule &rule = goto_;
00287     SECTION("By direct type.")
00288     {
00289         REQUIRE(
00290             str(goto_) ==
00291             "goto test_chain with priority -3 "
00292             "if PING from 192.168/8 to 172.16/4");
00293     }
00294     SECTION("By polymorphic type.")
00295     {
00296         REQUIRE(
00297             str(rule) ==
00298             "goto test_chain with priority -3 "
00299             "if PING from 192.168/8 to 172.16/4");
00300     }
00301 }
00302
00303
00304 TEST_CASE("GoTo's 'clone' method returns a polymorphic copy.", "[GoTo]")
00305 {
00306     auto chain = std::make_shared<TestChain>();
00307     SECTION("Without a priority.")
00308     {
00309         GoTo goto_(chain, If().type("PING"));
00310         Rule &rule = goto_;
00311         std::unique_ptr<Rule> polymorphic_copy = rule.clone();
00312         REQUIRE(goto_ == *polymorphic_copy);
00313     }
00314     SECTION("With a priority.")
00315     {
00316         GoTo goto_(chain, 3, If().type("PING"));
00317         Rule &rule = goto_;
00318         std::unique_ptr<Rule> polymorphic_copy = rule.clone();
00319         REQUIRE(goto_ == *polymorphic_copy);
00320     }
00321 }

```

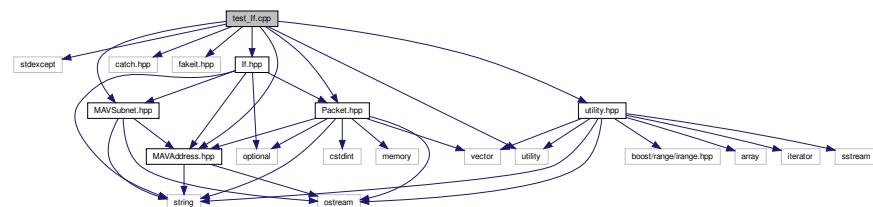
16.217 test_If.cpp File Reference

```

#include <stdexcept>
#include <utility>
#include <catch.hpp>
#include <fakeit.hpp>
#include "If.hpp"
#include "MAVAddress.hpp"
#include "MAVSubnet.hpp"
#include "Packet.hpp"
#include "utility.hpp"

```

Include dependency graph for test_If.cpp:



Functions

- [TEST_CASE \("If's are constructable.", "\[If\]"\)](#)
- [TEST_CASE \("If's are comparable.", "\[If\]"\)](#)
- [TEST_CASE \("If's are copyable.", "\[If\]"\)](#)
- [TEST_CASE \("If's are movable.", "\[If\]"\)](#)
- [TEST_CASE \("If's are assignable.", "\[If\]"\)](#)
- [TEST_CASE \("If's 'check' method determines if a packet and destination" "address matches the conditional.", "\[If\]"\)](#)
- [TEST_CASE \("If's 'type' method sets the packet ID for matching.", "\[If\]"\)](#)
- [TEST_CASE \("If's 'from' method sets the source subnet to match.", "\[If\]"\)](#)
- [TEST_CASE \("If's 'to' method sets the source subnet to match.", "\[If\]"\)](#)
- [TEST_CASE \("If's are printable.", "\[If\]"\)](#)

Variables

- [If if_b \(11, MAVSubnet\("255.255"\), {}\)](#)
- [if_a = std::move\(if_b\)](#)

16.217.1 Function Documentation

16.217.1.1 TEST_CASE() [1/10]

```
TEST_CASE (
    "If's are constructable." ,
    "" [If] )
```

Definition at line 34 of file [test_If.cpp](#).

16.217.1.2 TEST_CASE() [2/10]

```
TEST_CASE (
    "If's are comparable." ,
    "" [If] )
```

Definition at line 68 of file [test_If.cpp](#).

16.217.1.3 TEST_CASE() [3/10]

```
TEST_CASE (
    "If's are copyable." ,
    "" [If] )
```

Definition at line 139 of file [test_If.cpp](#).

16.217.1.4 TEST_CASE() [4/10]

```
TEST_CASE (
    "If's are movable." ,
    "" [If] )
```

Definition at line 147 of file [test_If.cpp](#).

16.217.1.5 TEST_CASE() [5/10]

```
TEST_CASE (
    "If's are assignable." ,
    "" [If] )
```

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

References [if_a](#), and [if_b](#).

16.217.1.6 TEST_CASE() [6/10]

```
TEST_CASE (
    "If's 'check' method determines if a packet and destination" "address matches the
conditional." ,
    "" [If] )
```

Definition at line 175 of file [test_If.cpp](#).

16.217.1.7 TEST_CASE() [7/10]

```
TEST_CASE (
    "If's 'type' method sets the packet ID for matching." ,
    "" [If] )
```

Definition at line 259 of file [test_If.cpp](#).

16.217.1.8 TEST_CASE() [8/10]

```
TEST_CASE (
    "If's 'from' method sets the source subnet to match." ,
    "" [If] )
```

Definition at line 300 of file [test_If.cpp](#).

16.217.1.9 TEST_CASE() [9/10]

```
TEST_CASE (
    "If's 'to' method sets the source subnet to match." ,
    "" [If] )
```

Definition at line 325 of file [test_If.cpp](#).

16.217.1.10 TEST_CASE() [10/10]

```
TEST_CASE (
    "If's are printable." ,
    "" [If] )
```

Definition at line 350 of file [test_If.cpp](#).

16.217.2 Variable Documentation

16.217.2.1 if_a

```
if_a = std::move(if_b)
```

Definition at line 170 of file [test_If.cpp](#).

16.217.2.2 if_b

```
If if_b(11, MAVSubnet("255.255"), {})
```

Initial value:

```
{
    If if_a(4, {}, MAVSubnet("255.255"))
```

16.218 test_If.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <stdexcept>
00019 #include <utility>
00020
00021 #include <catch.hpp>
00022 #include <fakeit.hpp>
00023
00024 #include "If.hpp"
00025 #include "MAVAddress.hpp"
00026 #include "MAVSubnet.hpp"
00027 #include "Packet.hpp"
00028 #include "utility.hpp"
00029
00030
00031 using namespace fakeit;
00032
00033
00034 TEST_CASE("If's are constructable.", "[If]")
00035 {
00036     SECTION("With the default constructor.")
00037     {
00038         REQUIRE_NO_THROW(If());
00039     }
00040     SECTION("With ID, source subnet, and destination subnet arguments.")
00041     {
00042         REQUIRE_NO_THROW(If({}, {}, {}));
00043         // ID 4 is PING.
00044         REQUIRE_NO_THROW(If(4, {}, {}));
00045         REQUIRE_NO_THROW(If({}, MAVSubnet("192.0/8"), {}));
00046         REQUIRE_NO_THROW(If({}, {}, MAVSubnet("192.0/8")));
00047     }
```

```

00047     REQUIRE_NOTHROW(If({}, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")));
00048     // ID 11 is SET_MODE.
00049     REQUIRE_NOTHROW(If(11, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")));
00050 }
00051 SECTION("ID as well as source and destination address are optional.")
00052 {
00053     REQUIRE_NOTHROW(If());
00054     REQUIRE_NOTHROW(If(4));
00055     REQUIRE_NOTHROW(If(11, MAVSubnet("192.0/8")));
00056 }
00057 SECTION("Ensures the packet ID is valid.")
00058 {
00059     // Currently, ID's 255 and 5000 are invalid.
00060     REQUIRE_THROWS_AS(If(255), std::invalid_argument);
00061     REQUIRE_THROWS_WITH(If(255), "Invalid packet ID (#255).");
00062     REQUIRE_THROWS_AS(If(5000), std::invalid_argument);
00063     REQUIRE_THROWS_WITH(If(5000), "Invalid packet ID (#5000).");
00064 }
00065 }
00066
00067
00068 TEST_CASE("If's are comparable.", "[If]")
00069 {
00070     SECTION("with ==")
00071 {
00072     // This also tests the default arguments.
00073     REQUIRE(If() == If());
00074     REQUIRE(If({}, {}, {}) == If());
00075     REQUIRE(If(4, {}, {}) == If(4));
00076     REQUIRE(
00077         If({}, MAVSubnet("192.0/8"), {}) ==
00078         If({}, MAVSubnet("192.0/8")));
00079     REQUIRE(
00080         If({}, {}, MAVSubnet("192.0/8")) ==
00081         If({}, {}, MAVSubnet("192.0/8")));
00082     REQUIRE(
00083         If({}, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")) ==
00084         If({}, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")));
00085     REQUIRE(
00086         If(11, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")) ==
00087         If(11, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")));
00088     REQUIRE_FALSE(If(0, {}, {}) == If());
00089     REQUIRE_FALSE(If(4, {}, {}) == If(0));
00090     REQUIRE_FALSE(
00091         If({}, MAVSubnet("192.0/8"), {}) ==
00092         If({}, MAVSubnet("192.0/7")));
00093     REQUIRE_FALSE(
00094         If({}, {}, MAVSubnet("191.0/8")) ==
00095         If({}, {}, MAVSubnet("192.0/8")));
00096     REQUIRE_FALSE(
00097         If({}, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")) ==
00098         If({}, MAVSubnet("192.0/8"), MAVSubnet("168.1/8")));
00099     REQUIRE_FALSE(
00100         If(1, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")) ==
00101         If(11, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")));
00102 }
00103 SECTION("with !=")
00104 {
00105     // This also tests the default arguments.
00106     REQUIRE(If(0, {}, {}) != If());
00107     REQUIRE(If(4, {}, {}) != If(0));
00108     REQUIRE(
00109         If({}, MAVSubnet("192.0/8"), {}) !=
00110         If({}, MAVSubnet("192.0/7")));
00111     REQUIRE(
00112         If({}, {}, MAVSubnet("191.0/8")) !=
00113         If({}, {}, MAVSubnet("192.0/8")));
00114     REQUIRE(
00115         If({}, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")) !=
00116         If({}, MAVSubnet("192.0/8"), MAVSubnet("168.1/8")));
00117     REQUIRE(
00118         If(1, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")) !=
00119         If(11, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")));
00120     REQUIRE_FALSE(If() != If());
00121     REQUIRE_FALSE(If({}, {}, {}) != If());
00122     REQUIRE_FALSE(If(4, {}, {}) != If(4));
00123     REQUIRE_FALSE(
00124         If({}, MAVSubnet("192.0/8"), {}) !=
00125         If({}, MAVSubnet("192.0/8")));
00126     REQUIRE_FALSE(
00127         If({}, {}, MAVSubnet("192.0/8")) !=

```

```

00128         If({}, {}, MAVSubnet("192.0/8")));
00129     REQUIRE_FALSE(
00130         If({}, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")) != 
00131         If({}, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")));
00132     REQUIRE_FALSE(
00133         If(11, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")) != 
00134         If(11, MAVSubnet("192.0/8"), MAVSubnet("168.0/8")));
00135     }
00136 }
00137
00138
00139 TEST_CASE("If's are copyable.", "[If]")
00140 {
00141     If original(4, MAVSubnet("192.168"), MAVSubnet("172.16"));
00142     If copy(original);
00143     REQUIRE(copy == If(4, MAVSubnet("192.168"), MAVSubnet("172.16")));
00144 }
00145
00146
00147 TEST_CASE("If's are movable.", "[If]")
00148 {
00149     If original(4, MAVSubnet("192.168"), MAVSubnet("172.16"));
00150     If moved(std::move(original));
00151     REQUIRE(moved == If(4, MAVSubnet("192.168"), MAVSubnet("172.16")));
00152 }
00153
00154
00155 TEST_CASE("If's are assignable.", "[If]")
00156 {
00157     If if_a(4, {}, MAVSubnet("255.255"));
00158     If if_b(11, MAVSubnet("255.255"), {});
00159     REQUIRE(if_a == If(4, {}, MAVSubnet("255.255")));
00160     if_a = if_b;
00161     REQUIRE(if_a == If(11, MAVSubnet("255.255"), {}));
00162 }
00163
00164
00165 TEST_CASE("If's are assignable (by move semantics.)", "[If]")
00166 {
00167     If if_a(4, {}, MAVSubnet("255.255"));
00168     If if_b(11, MAVSubnet("255.255"), {});
00169     REQUIRE(if_a == If(4, {}, MAVSubnet("255.255")));
00170     if_a = std::move(if_b);
00171     REQUIRE(if_a == If(11, MAVSubnet("255.255"), {}));
00172 }
00173
00174
00175 TEST_CASE("If's 'check' method determines if a packet and destination"
00176             "address matches the conditional.", "[If]")
00177 {
00178     Mock<Packet> mock;
00179     When(Method(mock, id)).AlwaysReturn(4);
00180     When(Method(mock, source)).AlwaysReturn(MAVAddress("192.168"));
00181     Packet &packet = mock.get();
00182     MAVAddress address("172.16");
00183     SECTION("Default all matching case.")
00184     {
00185         REQUIRE(If().check(packet, address));
00186     }
00187     SECTION("Based on packet ID only.")
00188     {
00189         REQUIRE_FALSE(If(1).check(packet, address));
00190         REQUIRE(If(4).check(packet, address));
00191         REQUIRE_FALSE(If(321).check(packet, address));
00192     }
00193     SECTION("Based on source subnet only.")
00194     {
00195         REQUIRE(If({}, MAVSubnet("192.168")).check(packet, address));
00196         REQUIRE_FALSE(If({}, MAVSubnet("193.168")).check(packet, address));
00197         REQUIRE(If({}, MAVSubnet("192.0/8")).check(packet, address));
00198         REQUIRE_FALSE(If({}, MAVSubnet("193.0/8")).check(packet, address));
00199     }
00200     SECTION("Based on destination subnet only.")
00201     {
00202         REQUIRE(If({}, {}, MAVSubnet("172.16")).check(packet, address));
00203         REQUIRE_FALSE(If({}, {}, MAVSubnet("171.16")).check(packet, address));
00204         REQUIRE(If({}, {}, MAVSubnet("172.16/8")).check(packet, address));
00205         REQUIRE_FALSE(If({}, {}, MAVSubnet("171.16/8")).check(packet, address));
00206     }
00207     SECTION("Based on packet ID and source subnet only.")
00208     {

```

```

00209     REQUIRE(IF(4, MAVSubnet("192.168")).check(packet, address));
00210     REQUIRE_FALSE(IF(11, MAVSubnet("192.168")).check(packet, address));
00211     REQUIRE_FALSE(IF(4, MAVSubnet("193.168")).check(packet, address));
00212     REQUIRE_FALSE(IF(11, MAVSubnet("193.168")).check(packet, address));
00213     REQUIRE(IF(4, MAVSubnet("192.0/8")).check(packet, address));
00214     REQUIRE_FALSE(IF(11, MAVSubnet("192.0/8")).check(packet, address));
00215     REQUIRE_FALSE(IF(4, MAVSubnet("193.0/8")).check(packet, address));
00216     REQUIRE_FALSE(IF(11, MAVSubnet("193.0/8")).check(packet, address));
00217 }
00218 SECTION("Based on packet ID and destination subnet only.")
00219 {
00220     REQUIRE(IF(4, {}, MAVSubnet("172.16")).check(packet, address));
00221     REQUIRE_FALSE(IF(11, {}, MAVSubnet("172.16")).check(packet, address));
00222     REQUIRE_FALSE(IF(4, {}, MAVSubnet("171.16")).check(packet, address));
00223     REQUIRE_FALSE(IF(11, {}, MAVSubnet("171.16")).check(packet, address));
00224     REQUIRE(IF(4, {}, MAVSubnet("172.16/8")).check(packet, address));
00225     REQUIRE_FALSE(IF(11, {}, MAVSubnet("172.16/8")).check(packet, address));
00226     REQUIRE_FALSE(IF(4, {}, MAVSubnet("171.16/8")).check(packet, address));
00227     REQUIRE_FALSE(IF(11, {}, MAVSubnet("171.16/8")).check(packet, address));
00228 }
00229 SECTION("Based on packet ID, source subnet, and destination subnet.")
00230 {
00231     REQUIRE(
00232         IF(4, MAVSubnet("192.168"), MAVSubnet("172.16")).check(
00233             packet, address));
00234     REQUIRE_FALSE(
00235         IF(11, MAVSubnet("192.168"), MAVSubnet("172.16")).check(
00236             packet, address));
00237     REQUIRE_FALSE(
00238         IF(4, MAVSubnet("193.168"), MAVSubnet("172.16")).check(
00239             packet, address));
00240     REQUIRE_FALSE(
00241         IF(11, MAVSubnet("193.168"), MAVSubnet("172.16")).check(
00242             packet, address));
00243     REQUIRE_FALSE(
00244         IF(4, MAVSubnet("192.168"), MAVSubnet("171.16")).check(
00245             packet, address));
00246     REQUIRE_FALSE(
00247         IF(11, MAVSubnet("192.168"), MAVSubnet("171.16")).check(
00248             packet, address));
00249     REQUIRE_FALSE(
00250         IF(4, MAVSubnet("193.168"), MAVSubnet("171.16")).check(
00251             packet, address));
00252     REQUIRE_FALSE(
00253         IF(11, MAVSubnet("193.168"), MAVSubnet("171.16")).check(
00254             packet, address));
00255 }
00256 }
00257
00258
00259 TEST_CASE("If's 'type' method sets the packet ID for matching.",
00260             "[If]")
00261 {
00262     Mock<Packet> mock;
00263     When(Method(mock, id)).AlwaysReturn(4);
00264     When(Method(mock, source)).AlwaysReturn(MAVAddress("192.168"));
00265     Packet &packet = mock.get();
00266     MAVAddress address("172.16");
00267     SECTION("When given a numeric ID.")
00268 {
00269     REQUIRE_FALSE(IF().type(0).check(packet, address));
00270     REQUIRE(IF().type(4).check(packet, address));
00271     REQUIRE_FALSE(IF().type(11).check(packet, address));
00272 }
00273 SECTION("And ensures the numeric ID is valid.")
00274 {
00275     // Note: ID's 255 and 5000 are not currently valid.
00276     REQUIRE_THROWS_AS(IF().type(255), std::invalid_argument);
00277     REQUIRE_THROWS_WITH(
00278         IF().type(255), "Invalid packet ID (#255).");
00279     REQUIRE_THROWS_AS(IF().type(5000), std::invalid_argument);
00280     REQUIRE_THROWS_WITH(
00281         IF().type(5000), "Invalid packet ID (#5000).");
00282 }
00283 SECTION("When given a packet name.")
00284 {
00285     REQUIRE_FALSE(IF().type("HEARTBEAT").check(packet, address));
00286     REQUIRE(IF().type("PING").check(packet, address));
00287     REQUIRE_FALSE(IF().type("SET_MODE").check(packet, address));
00288 }
00289 SECTION("And ensures the packet name is valid.")

```

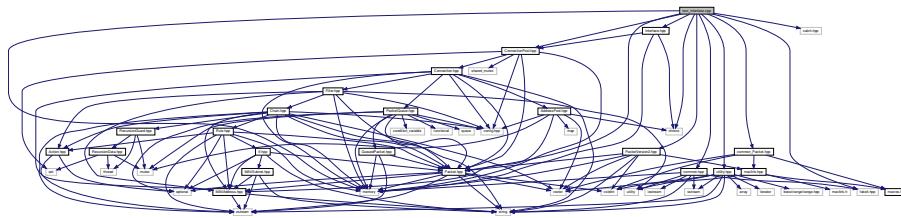
```

00290  {
00291      REQUIRE_THROWS_AS(
00292          If().type("CRAZY_PACKET_NAME"), std::invalid_argument);
00293      REQUIRE_THROWS_WITH(
00294          If().type("CRAZY_PACKET_NAME"),
00295          "Invalid packet name (\\"CRAZY_PACKET_NAME\\").");
00296  }
00297 }
00298
00299
00300 TEST_CASE("If's 'from' method sets the source subnet to match.",
00301     "[If]")
00302 {
00303     Mock<Packet> mock;
00304     When(Method(mock, id)).AlwaysReturn(4);
00305     When(Method(mock, source)).AlwaysReturn(MAVAddress("192.168"));
00306     Packet &packet = mock.get();
00307     MAVAddress address("172.16");
00308     SECTION("When a MAVSubnet object is given.")
00309     {
00310         REQUIRE(If().from(MAVSubnet("192.168")).check(packet, address));
00311         REQUIRE_FALSE(If().from(MAVSubnet("193.168")).check(packet, address));
00312         REQUIRE(If().from(MAVSubnet("192.0/8")).check(packet, address));
00313         REQUIRE_FALSE(If().from(MAVSubnet("193.0/8")).check(packet, address));
00314     }
00315     SECTION("When a string is given.")
00316     {
00317         REQUIRE(If().from("192.168").check(packet, address));
00318         REQUIRE_FALSE(If().from("193.168").check(packet, address));
00319         REQUIRE(If().from("192.0/8").check(packet, address));
00320         REQUIRE_FALSE(If().from("193.0/8").check(packet, address));
00321     }
00322 }
00323
00324
00325 TEST_CASE("If's 'to' method sets the source subnet to match.",
00326     "[If]")
00327 {
00328     Mock<Packet> mock;
00329     When(Method(mock, id)).AlwaysReturn(4);
00330     When(Method(mock, source)).AlwaysReturn(MAVAddress("192.168"));
00331     Packet &packet = mock.get();
00332     MAVAddress address("172.16");
00333     SECTION("When a MAVSubnet object is given.")
00334     {
00335         REQUIRE(If().to(MAVSubnet("172.16")).check(packet, address));
00336         REQUIRE_FALSE(If().to(MAVSubnet("171.16")).check(packet, address));
00337         REQUIRE(If().to(MAVSubnet("172.16/8")).check(packet, address));
00338         REQUIRE_FALSE(If().to(MAVSubnet("171.16/8")).check(packet, address));
00339     }
00340     SECTION("When a string is given.")
00341     {
00342         REQUIRE(If().to("172.16").check(packet, address));
00343         REQUIRE_FALSE(If().to("171.16").check(packet, address));
00344         REQUIRE(If().to("172.16/8").check(packet, address));
00345         REQUIRE_FALSE(If().to("171.16/8").check(packet, address));
00346     }
00347 }
00348
00349
00350 TEST_CASE("If's are printable.", "[If]")
00351 {
00352     REQUIRE(str(If()) == "if any");
00353     REQUIRE(str(If().type("HEARTBEAT")) == "if HEARTBEAT");
00354     REQUIRE(str(If().type("PING")) == "if PING");
00355     REQUIRE(str(If().type("SET_MODE")) == "if SET_MODE");
00356     REQUIRE(str(If().from("192.168")) == "if from 192.168");
00357     REQUIRE(str(If().from("192.0/8")) == "if from 192.0/8");
00358     REQUIRE(str(If().to("172.16")) == "if to 172.16");
00359     REQUIRE(str(If().to("172.0/8")) == "if to 172.0/8");
00360     REQUIRE(
00361         str(If().type("PING").from("192.168")) == "if PING from 192.168");
00362     REQUIRE(
00363         str(If().type("PING").to("172.16")) == "if PING to 172.16");
00364     REQUIRE(
00365         str(If().from("192.168").to("172.16")) == "if from 192.168 to 172.16");
00366     REQUIRE(
00367         str(If().type("PING").from("192.168").to("172.16")) ==
00368         "if PING from 192.168 to 172.16");
00369 }

```

16.219 test_Interface.cpp File Reference

```
#include <chrono>
#include <memory>
#include <catch.hpp>
#include <fakeit.hpp>
#include "ConnectionPool.hpp"
#include "Interface.hpp"
#include "Packet.hpp"
#include "PacketVersion2.hpp"
#include "utility.hpp"
#include "common.hpp"
#include "common_Packet.hpp"
Include dependency graph for test_Interface.cpp:
```



Functions

- [TEST_CASE \("Interface's can be constructed.", "\[Interface\]"\)](#)
- [TEST_CASE \("Interface's 'send_packet' method \(included just for coverage\).", "\[Interface\]"\)](#)
- [TEST_CASE \("Interface's 'receive_packet' method sends the packet using the " "contained ConnectionPool.", "\[Interface\]"\)](#)
- [TEST_CASE \("Interface's are printable.", "\[Interface\]"\)](#)

16.219.1 Function Documentation

16.219.1.1 TEST_CASE() [1/4]

```
TEST_CASE (
    "Interface's can be constructed." ,
    "" [Interface] )
```

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

References [mock_shared\(\)](#).

Here is the call graph for this function:



16.219.1.2 TEST_CASE() [2/4]

```
TEST_CASE (
    "Interface's 'send_packet' method (included just for coverage)." "" [Interface] )
```

Definition at line 94 of file [test_Interface.cpp](#).

References [mock_shared\(\)](#).

Here is the call graph for this function:



16.219.1.3 TEST_CASE() [3/4]

```
TEST_CASE (
    "Interface's 'receive_packet' method sends the packet using the " "contained Connection<-->
    Pool." ,
    "" [Interface] )
```

Definition at line 104 of file [test_Interface.cpp](#).

References [mock_shared\(\)](#).

Here is the call graph for this function:



16.219.1.4 TEST_CASE() [4/4]

```
TEST_CASE (
    "Interface's are printable." ,
    "" [Interface] )
```

Definition at line 119 of file [test_Interface.cpp](#).

References [mock_shared\(\)](#).

Here is the call graph for this function:



16.220 test_Interface.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
```

```
00016
00017
00018 #include <chrono>
00019 #include <memory>
00020
00021 #include <catch.hpp>
00022 #include <fakeit.hpp>
00023
00024 #include "ConnectionPool.hpp"
00025 #include "Interface.hpp"
00026 #include "Packet.hpp"
00027 #include "PacketVersion2.hpp"
00028 #include "utility.hpp"
00029
00030 #include "common.hpp"
00031 #include "common_Packet.hpp"
00032
00033
00034 namespace
00035 {
00036
00037 #ifdef __clang__
00038     #pragma clang diagnostic push
00039     #pragma clang diagnostic ignored "-Wweak-vtables"
00040 #endif
00041
00042     // Subclass of Interface used for testing the abstract class Interface.
00043 class InterfaceTestClass : public Interface
00044 {
00045     public:
00046         InterfaceTestClass(std::shared_ptr<ConnectionPool> connection_pool)
00047             : connection_pool_(std::move(connection_pool))
00048         {
00049         }
00050         // LCOV_EXCL_START
00051     ~InterfaceTestClass() = default;
00052         // LCOV_EXCL_STOP
00053     void send_packet(
00054         const std::chrono::nanoseconds &timeout =
00055             std::chrono::nanoseconds(100000)) final
00056     {
00057         (void)timeout;
00058     }
00059     void receive_packet(
00060         const std::chrono::nanoseconds &timeout =
00061             std::chrono::nanoseconds(100000)) final
00062     {
00063         (void)timeout;
00064         connection_pool_->send(
00065             std::make_unique<packet_v2::Packet>(to_vector(PingV2())));
00066     }
00067
00068     protected:
00069         std::ostream &print_(std::ostream &os) const final
00070     {
00071         os << "interface test class";
00072         return os;
00073     }
00074
00075     private:
00076         std::shared_ptr<ConnectionPool> connection_pool_;
00077     };
00078
00079 #ifdef __clang__
00080     #pragma clang diagnostic pop
00081 #endif
00082
00083 }
00084
00085
00086 TEST_CASE("Interface's can be constructed.", "[Interface]")
00087 {
00088     fakeit::Mock<ConnectionPool> mock_pool;
00089     std::shared_ptr<ConnectionPool> pool = mock_shared(mock_pool);
00090     REQUIRE_NO_THROW(InterfaceTestClass(pool));
00091 }
00092
00093
00094 TEST_CASE("Interface's 'send_packet' method (included just for coverage)."
00095             "[Interface]")
00096 }
```

```

00097     fakeit::Mock<ConnectionPool> mock_pool;
00098     std::shared_ptr<ConnectionPool> pool = mock_shared(mock_pool);
00099     InterfaceTestClass interface(pool);
00100     REQUIRE_NO_THROW(interface.send_packet());
00101 }
00102
00103
00104 TEST_CASE("Interface's 'receive_packet' method sends the packet using the "
00105             "contained ConnectionPool.", "[Interface]")
00106 {
00107     fakeit::Mock<ConnectionPool> mock_pool;
00108     fakeit::Fake(Method(mock_pool, send));
00109     std::shared_ptr<ConnectionPool> pool = mock_shared(mock_pool);
00110     InterfaceTestClass interface(pool);
00111     interface.receive_packet();
00112     fakeit::Verify(Method(mock_pool, send).Matching([](auto &a)
00113     {
00114         return a != nullptr && *a == packet_v2::Packet(to_vector(PingV2()));
00115     })).Once();
00116 }
00117
00118
00119 TEST_CASE("Interface's are printable.", "[Interface]")
00120 {
00121     fakeit::Mock<ConnectionPool> mock_pool;
00122     std::shared_ptr<ConnectionPool> pool = mock_shared(mock_pool);
00123     REQUIRE(str(InterfaceTestClass(pool)) == "interface test class");
00124 }

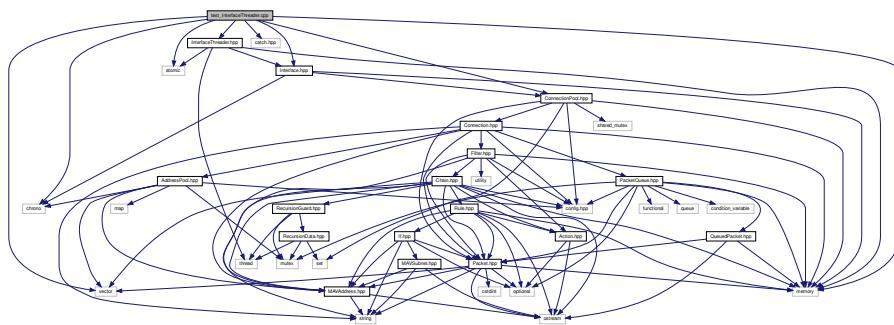
```

16.221 test_InterfaceThreader.cpp File Reference

```

#include <atomic>
#include <chrono>
#include <memory>
#include <vector>
#include <catch.hpp>
#include "ConnectionPool.hpp"
#include "Interface.hpp"
#include "InterfaceThreader.hpp"
Include dependency graph for test_InterfaceThreader.cpp:

```



Functions

- **TEST_CASE** ("InterfaceThreader's can be constructed.", "[InterfaceThreader]")
- **TEST_CASE** ("InterfaceThreader's run Interface::send_packet and " "Interface::receive_packet methods of the contained " "Interface repeatedly.", "[InterfaceThreader]")

16.221.1 Function Documentation

16.221.1.1 TEST_CASE() [1/2]

```
TEST_CASE (
    "InterfaceThreader's can be constructed." ,
    "" [InterfaceThreader] )
```

Definition at line 89 of file [test_InterfaceThreader.cpp](#).

16.221.1.2 TEST_CASE() [2/2]

```
TEST_CASE (
    "InterfaceThreader's run Interface::send_packet and " "Interface::receive_packet
methods of the contained " "Interface repeatedly." ,
    "" [InterfaceThreader] )
```

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

16.222 test_InterfaceThreader.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <atomic>
00019 #include <chrono>
00020 #include <memory>
00021 #include <vector>
00022
00023 #include <catch.hpp>
00024
00025 #include "ConnectionPool.hpp"
00026 #include "Interface.hpp"
00027 #include "InterfaceThreader.hpp"
00028
00029
00030 using namespace std::chrono_literals;
00031
00032
00033 namespace
00034 {
```

```

00035
00036 #ifdef __clang__
00037     #pragma clang diagnostic push
00038     #pragma clang diagnostic ignored "-Wweak-vtables"
00039 #endif
00040
00041     // Subclass of Interface used for testing the abstract class Interface.
00042     class InterfaceTestClass : public Interface
00043     {
00044         public:
00045             InterfaceTestClass(
00046                 std::atomic<unsigned int> &tx_counter_,
00047                 std::atomic<unsigned int> &rx_counter_)
00048                 : tx_counter(tx_counter_), rx_counter(rx_counter_)
00049             {
00050             }
00051             void send_packet(
00052                 const std::chrono::nanoseconds &timeout =
00053                     std::chrono::nanoseconds(100000) final
00054             {
00055                 std::this_thread::sleep_for(timeout);
00056                 ++tx_counter;
00057             }
00058             void receive_packet(
00059                 const std::chrono::nanoseconds &timeout =
00060                     std::chrono::nanoseconds(100000) final
00061             {
00062                 std::this_thread::sleep_for(timeout);
00063                 ++rx_counter;
00064             }
00065
00066         protected:
00067             // No point in testing this.
00068             // LCOV_EXCL_START
00069             std::ostream &print_(std::ostream &os) const final
00070             {
00071                 os << "interface test class";
00072                 return os;
00073             }
00074             // LCOV_EXCL_STOP
00075
00076         private:
00077             std::atomic<unsigned int> &tx_counter;
00078             std::atomic<unsigned int> &rx_counter;
00079
00080     };
00081
00082 #endif __clang__
00083     #pragma clang diagnostic pop
00084 #endif
00085
00086 }
00087
00088
00089 TEST_CASE("InterfaceThreader's can be constructed.", "[InterfaceThreader]")
00090 {
00091     std::atomic<unsigned int> tx_count(0);
00092     std::atomic<unsigned int> rx_count(0);
00093     SECTION("With delayed start.")
00094     {
00095         REQUIRE_NO_THROW(
00096             InterfaceThreader(
00097                 std::make_unique<InterfaceTestClass>(tx_count, rx_count),
00098                 1ms, InterfaceThreader::DELAY_START));
00099     }
00100     SECTION("With immediate start.")
00101     {
00102         REQUIRE_NO_THROW(
00103             InterfaceThreader(
00104                 std::make_unique<InterfaceTestClass>(tx_count, rx_count)));
00105     }
00106 }
00107
00108
00109 TEST_CASE("InterfaceThreader's run Interface::send_packet and "
00110             "Interface::receive_packet methods of the contained "
00111             "Interface repeatedly.", "[InterfaceThreader]")
00112 {
00113     std::atomic<unsigned int> tx_count(0);
00114     std::atomic<unsigned int> rx_count(0);
00115     SECTION("With delayed start.")

```

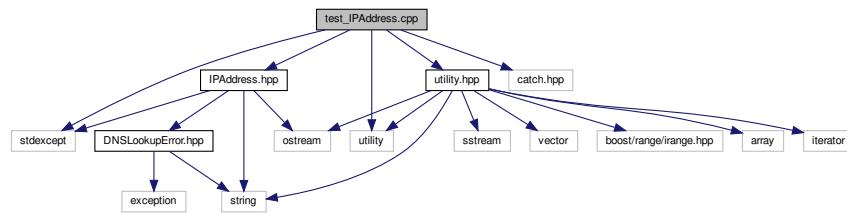
```

00116    {
00117        REQUIRE(tx_count == 0);
00118        REQUIRE(rx_count == 0);
00119        InterfaceThreader threader(
00120            std::make_unique<InterfaceTestClass>(tx_count, rx_count),
00121            lus, InterfaceThreader::DELAY_START);
00122        REQUIRE(tx_count == 0);
00123        REQUIRE(rx_count == 0);
00124        threader.start();
00125        std::this_thread::sleep_for(10ms);
00126        threader.shutdown();
00127        auto tx_count_ = tx_count.load();
00128        auto rx_count_ = rx_count.load();
00129        REQUIRE(tx_count > 0);
00130        REQUIRE(rx_count > 0);
00131        std::this_thread::sleep_for(10ms);
00132        REQUIRE(tx_count == tx_count_);
00133        REQUIRE(rx_count == rx_count_);
00134    }
00135 SECTION("With immediate start (manual shutdown).")
00136 {
00137     REQUIRE(tx_count == 0);
00138     REQUIRE(rx_count == 0);
00139     InterfaceThreader threader(
00140         std::make_unique<InterfaceTestClass>(tx_count, rx_count), lus);
00141     std::this_thread::sleep_for(10ms);
00142     threader.shutdown();
00143     auto tx_count_ = tx_count.load();
00144     auto rx_count_ = rx_count.load();
00145     REQUIRE(tx_count > 0);
00146     REQUIRE(rx_count > 0);
00147     std::this_thread::sleep_for(10ms);
00148     REQUIRE(tx_count == tx_count_);
00149     REQUIRE(rx_count == rx_count_);
00150 }
00151 SECTION("With immediate start (RAII shutdown).")
00152 {
00153     REQUIRE(tx_count == 0);
00154     REQUIRE(rx_count == 0);
00155     {
00156         InterfaceThreader threader(
00157             std::make_unique<InterfaceTestClass>(tx_count, rx_count), lus);
00158         std::this_thread::sleep_for(10ms);
00159     }
00160     auto tx_count_ = tx_count.load();
00161     auto rx_count_ = rx_count.load();
00162     REQUIRE(tx_count > 0);
00163     REQUIRE(rx_count > 0);
00164     std::this_thread::sleep_for(10ms);
00165     REQUIRE(tx_count == tx_count_);
00166     REQUIRE(rx_count == rx_count_);
00167 }
00168 }
```

16.223 test_IPAddress.cpp File Reference

```
#include <stdexcept>
#include <utility>
#include <catch.hpp>
#include "IPAddress.hpp"
#include "utility.hpp"
```

Include dependency graph for test_IPAddress.cpp:



Functions

- [TEST_CASE \("IPAddress's store an address and a port number.", "\[**IPAddress**\]""\)](#)
- [TEST_CASE \("IPAddress's are comparable.", "\[**IPAddress**\]""\)](#)
- [TEST_CASE \("IPAddress's can be constructed from an address and port.", "\[**IPAddress**\]""\)](#)
- [TEST_CASE \("IPAddress's can be constructed from strings.", "\[**IPAddress**\]""\)](#)
- [TEST_CASE \("IPAddress's are copyable.", "\[**IPAddress**\]""\)](#)
- [TEST_CASE \("The port of an IP address can be changed during a copy.", "\[**IPAddress**\]""\)](#)
- [TEST_CASE \("IPAddress's are movable.", "\[**IPAddress**\]""\)](#)
- [TEST_CASE \("IPAddress's are assignable.", "\[**IPAddress**\]""\)](#)
- [TEST_CASE \("IPAddress's are printable", "\[**IPAddress**\]""\)](#)
- [TEST_CASE \("dnslookup finds **IPAddress**'s by domain name.", "\[**IPAddress**\]""\)](#)

Variables

- [IPAddress address_b \(0xFFFFFFFF, 65535\)](#)
- [address_a = std::move\(address_b\)](#)

16.223.1 Function Documentation

16.223.1.1 TEST_CASE() [1/10]

```
TEST_CASE (
    "IPAddress's store an address and a port number." ,
    "" [IPAddress] )
```

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

16.223.1.2 TEST_CASE() [2/10]

```
TEST_CASE (
    "IPAddress's are comparable." ,
    "" [IPAddress] )
```

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

16.223.1.3 TEST_CASE() [3/10]

```
TEST_CASE (
    "IPAddress's can be constructed from an address and port." ,
    "" [IPAddress] )
```

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

16.223.1.4 TEST_CASE() [4/10]

```
TEST_CASE (
    "IPAddress's can be constructed from strings." ,
    "" [IPAddress] )
```

Definition at line [168](#) of file [test_IPAddress.cpp](#).

16.223.1.5 TEST_CASE() [5/10]

```
TEST_CASE (
    "IPAddress's are copyable." ,
    "" [IPAddress] )
```

Definition at line [286](#) of file [test_IPAddress.cpp](#).

References [address_a](#), and [address_b](#).

16.223.1.6 TEST_CASE() [6/10]

```
TEST_CASE (
    "The port of an IP address can be changed during a copy." ,
    "" [IPAddress] )
```

Definition at line 299 of file [test_IPAddress.cpp](#).

References [address_a](#), and [address_b](#).

16.223.1.7 TEST_CASE() [7/10]

```
TEST_CASE (
    "IPAddress's are movable." ,
    "" [IPAddress] )
```

Definition at line 328 of file [test_IPAddress.cpp](#).

References [address_a](#), and [address_b](#).

16.223.1.8 TEST_CASE() [8/10]

```
TEST_CASE (
    "IPAddress's are assignable." ,
    "" [IPAddress] )
```

Definition at line 339 of file [test_IPAddress.cpp](#).

References [address_a](#), and [address_b](#).

16.223.1.9 TEST_CASE() [9/10]

```
TEST_CASE (
    "IPAddress's are printable" ,
    "" [IPAddress] )
```

Definition at line 359 of file [test_IPAddress.cpp](#).

16.223.1.10 TEST_CASE() [10/10]

```
TEST_CASE (
    "dnslookup finds IPAddress's by domain name." ,
    "" [IPAddress] )
```

Definition at line 372 of file [test_IPAddress.cpp](#).

16.223.2 Variable Documentation

16.223.2.1 address_a

```
address_a = std::move(address_b)
```

Definition at line 354 of file [test_IPAddress.cpp](#).

16.223.2.2 address_b

```
IPAddress address_b(0xFFFFFFFF, 65535)
```

Initial value:

```
{  
    IPAddress address_a(0x00000000, 0)
```

16.224 test_IPAddress.cpp

```
00001 // MAVLink router and firewall.  
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>  
00003 //  
00004 // This program is free software; you can redistribute it and/or modify  
00005 // it under the terms of the GNU General Public License as published by  
00006 // the Free Software Foundation; either version 2 of the License, or  
00007 // (at your option) any later version.  
00008 //  
00009 // This program is distributed in the hope that it will be useful,  
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of  
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
00012 // GNU General Public License for more details.  
00013 //  
00014 // You should have received a copy of the GNU General Public License  
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.  
00016  
00017  
00018 #include <stdexcept>  
00019 #include <utility>  
00020  
00021 #include <catch.hpp>  
00022
```

```

00023 #include "IPAddress.hpp"
00024 #include "utility.hpp"
00025
00026
00027 TEST_CASE("IPAddress's store an address and a port number.", "[IPAddress]")
00028 {
00029     IPAddress a(0x00000000, 0);
00030     IPAddress b(0xFFFFFFFF, 65535);
00031     REQUIRE(a.address() == 0x00000000);
00032     REQUIRE(a.port() == 0);
00033     REQUIRE(b.address() == 0xFFFFFFFF);
00034     REQUIRE(b.port() == 65535);
00035 }
00036
00037
00038 TEST_CASE("IPAddress's are comparable.", "[IPAddress]")
00039 {
00040     SECTION("with ==")
00041     {
00042         REQUIRE(IPAddress(0x00000000, 0) == IPAddress(0x00000000, 0));
00043         REQUIRE(IPAddress(0x88888888, 8) == IPAddress(0x88888888, 8));
00044         REQUIRE(IPAddress(0xFFFFFFFF, 65535) == IPAddress(0xFFFFFFFF, 65535));
00045         REQUIRE_FALSE(IPAddress(0x00000000, 0) == IPAddress(0x00000000, 1));
00046         REQUIRE_FALSE(IPAddress(0x00000000, 0) == IPAddress(0x00000001, 0));
00047         REQUIRE_FALSE(
00048             IPAddress(0xFFFFFFFF, 65535) == IPAddress(0xFFFFFFFF, 65534));
00049         REQUIRE_FALSE(
00050             IPAddress(0xFFFFFFFF, 65535) == IPAddress(0xFFFFFFF, 65535));
00051     }
00052     SECTION("with !=")
00053     {
00054         REQUIRE(IPAddress(0x00000000, 0) != IPAddress(0x00000000, 1));
00055         REQUIRE(IPAddress(0x00000000, 0) != IPAddress(0x00000001, 0));
00056         REQUIRE(IPAddress(0xFFFFFFFF, 65535) != IPAddress(0xFFFFFFFF, 65534));
00057         REQUIRE(IPAddress(0xFFFFFFFF, 65535) != IPAddress(0xFFFFFFF, 65535));
00058         REQUIRE_FALSE(IPAddress(0x00000000, 0) != IPAddress(0x00000000, 0));
00059         REQUIRE_FALSE(IPAddress(0x88888888, 8) != IPAddress(0x88888888, 8));
00060         REQUIRE_FALSE(
00061             IPAddress(0xFFFFFFFF, 65535) != IPAddress(0xFFFFFFFF, 65535));
00062     }
00063     SECTION("with <")
00064     {
00065         REQUIRE(IPAddress(0x00000000, 0) < IPAddress(0x00000000, 1));
00066         REQUIRE(IPAddress(0x00000000, 0) < IPAddress(0x00000001, 0));
00067         REQUIRE(IPAddress(0x00000000, 1) < IPAddress(0x00000001, 0));
00068         REQUIRE(IPAddress(0xFFFFFFFF, 65534) < IPAddress(0xFFFFFFFF, 65535));
00069         REQUIRE(IPAddress(0xFFFFFFFF, 65535) < IPAddress(0xFFFFFFFF, 65535));
00070         REQUIRE(IPAddress(0x00000000, 0) < IPAddress(0xFFFFFFFF, 65535));
00071         REQUIRE_FALSE(IPAddress(0x00000000, 1) < IPAddress(0x00000000, 0));
00072         REQUIRE_FALSE(IPAddress(0x00000001, 0) < IPAddress(0x00000000, 0));
00073         REQUIRE_FALSE(IPAddress(0x00000001, 0) < IPAddress(0x00000000, 1));
00074         REQUIRE_FALSE(
00075             IPAddress(0xFFFFFFFF, 65535) < IPAddress(0xFFFFFFFF, 65534));
00076         REQUIRE_FALSE(
00077             IPAddress(0xFFFFFFFF, 65535) < IPAddress(0xFFFFFFF, 65535));
00078         REQUIRE_FALSE(IPAddress(0x00000000, 0) < IPAddress(0x00000000, 0));
00079         REQUIRE_FALSE(IPAddress(0x88888888, 8) < IPAddress(0x88888888, 8));
00080         REQUIRE_FALSE(
00081             IPAddress(0xFFFFFFFF, 65535) < IPAddress(0xFFFFFFFF, 65535));
00082         REQUIRE_FALSE(IPAddress(0xFFFFFFFF, 65535) < IPAddress(0x00000000, 0));
00083     }
00084     SECTION("with >")
00085     {
00086         REQUIRE(IPAddress(0x00000000, 1) > IPAddress(0x00000000, 0));
00087         REQUIRE(IPAddress(0x00000001, 0) > IPAddress(0x00000000, 0));
00088         REQUIRE(IPAddress(0x00000001, 0) > IPAddress(0x00000000, 1));
00089         REQUIRE(IPAddress(0xFFFFFFFF, 65535) > IPAddress(0xFFFFFFFF, 65534));
00090         REQUIRE(IPAddress(0xFFFFFFFF, 65535) > IPAddress(0xFFFFFFF, 65535));
00091         REQUIRE(IPAddress(0xFFFFFFFF, 65535) > IPAddress(0x00000000, 0));
00092         REQUIRE_FALSE(IPAddress(0x00000000, 0) > IPAddress(0x00000000, 1));
00093         REQUIRE_FALSE(IPAddress(0x00000000, 0) > IPAddress(0x00000001, 0));
00094         REQUIRE_FALSE(IPAddress(0x00000000, 1) > IPAddress(0x00000001, 0));
00095         REQUIRE_FALSE(
00096             IPAddress(0xFFFFFFFF, 65534) > IPAddress(0xFFFFFFFF, 65535));
00097         REQUIRE_FALSE(
00098             IPAddress(0xFFFFFFF, 65535) > IPAddress(0xFFFFFFFF, 65535));
00099         REQUIRE_FALSE(IPAddress(0x00000000, 0) > IPAddress(0x00000000, 0));
00100         REQUIRE_FALSE(IPAddress(0x88888888, 8) > IPAddress(0x88888888, 8));
00101         REQUIRE_FALSE(
00102             IPAddress(0xFFFFFFFF, 65535) > IPAddress(0xFFFFFFFF, 65535));
00103         REQUIRE_FALSE(IPAddress(0x00000000, 0) > IPAddress(0xFFFFFFF, 65535));

```

```

00104     }
00105     SECTION("with <=")
00106     {
00107         REQUIRE(IPAddress(0x00000000, 0) <= IPAddress(0x00000000, 0));
00108         REQUIRE(IPAddress(0x88888888, 8) <= IPAddress(0x88888888, 8));
00109         REQUIRE(IPAddress(0xFFFFFFFF, 65535) <= IPAddress(0xFFFFFFFF, 65535));
00110         REQUIRE(IPAddress(0x00000000, 0) <= IPAddress(0x00000000, 1));
00111         REQUIRE(IPAddress(0x00000000, 0) <= IPAddress(0x00000001, 0));
00112         REQUIRE(IPAddress(0x00000000, 1) <= IPAddress(0x00000001, 0));
00113         REQUIRE(IPAddress(0xFFFFFFF, 65534) <= IPAddress(0xFFFFFFF, 65534));
00114         REQUIRE(IPAddress(0xFFFFFFF, 65535) <= IPAddress(0xFFFFFFF, 65535));
00115         REQUIRE(IPAddress(0x00000000, 0) <= IPAddress(0xFFFFFFF, 65535));
00116         REQUIRE_FALSE(IPAddress(0x00000000, 1) <= IPAddress(0x00000000, 0));
00117         REQUIRE_FALSE(IPAddress(0x00000001, 0) <= IPAddress(0x00000000, 0));
00118         REQUIRE_FALSE(IPAddress(0x00000001, 0) <= IPAddress(0x00000000, 1));
00119         REQUIRE_FALSE(
00120             IPAddress(0xFFFFFFF, 65535) <= IPAddress(0xFFFFFFF, 65534));
00121         REQUIRE_FALSE(
00122             IPAddress(0xFFFFFFF, 65535) <= IPAddress(0xFFFFFFF, 65535));
00123         REQUIRE_FALSE(IPAddress(0xFFFFFFF, 65535) <= IPAddress(0x00000000, 0));
00124     }
00125     SECTION("with >=")
00126     {
00127         REQUIRE(IPAddress(0x00000000, 0) >= IPAddress(0x00000000, 0));
00128         REQUIRE(IPAddress(0x88888888, 8) >= IPAddress(0x88888888, 8));
00129         REQUIRE(IPAddress(0xFFFFFFF, 65535) >= IPAddress(0xFFFFFFF, 65535));
00130         REQUIRE(IPAddress(0x00000000, 1) >= IPAddress(0x00000000, 0));
00131         REQUIRE(IPAddress(0x00000001, 0) >= IPAddress(0x00000000, 0));
00132         REQUIRE(IPAddress(0x00000001, 0) >= IPAddress(0x00000000, 1));
00133         REQUIRE(IPAddress(0xFFFFFFF, 65535) >= IPAddress(0xFFFFFFF, 65534));
00134         REQUIRE(IPAddress(0xFFFFFFF, 65535) >= IPAddress(0xFFFFFFF, 65535));
00135         REQUIRE(IPAddress(0xFFFFFFF, 65535) >= IPAddress(0x00000000, 0));
00136         REQUIRE_FALSE(IPAddress(0x00000000, 0) >= IPAddress(0x00000000, 1));
00137         REQUIRE_FALSE(IPAddress(0x00000000, 0) >= IPAddress(0x00000001, 0));
00138         REQUIRE_FALSE(IPAddress(0x00000000, 1) >= IPAddress(0x00000001, 0));
00139         REQUIRE_FALSE(
00140             IPAddress(0xFFFFFFF, 65534) >= IPAddress(0xFFFFFFF, 65535));
00141         REQUIRE_FALSE(
00142             IPAddress(0xFFFFFFF, 65535) >= IPAddress(0xFFFFFFF, 65535));
00143         REQUIRE_FALSE(IPAddress(0x00000000, 0) >= IPAddress(0xFFFFFFF, 65535));
00144     }
00145 }
00146
00147
00148 TEST_CASE("IPAddress's can be constructed from an address and port.", "[IPAddress]")
00149 {
00150     REQUIRE(IPAddress(0x00000000, 0) == IPAddress(0x00000000, 0));
00151     REQUIRE(IPAddress(0x12345678, 123) == IPAddress(0x12345678, 123));
00152     REQUIRE(IPAddress(0xFFFFFFF, 65535) == IPAddress(0xFFFFFFF, 65535));
00153     SECTION("And ensures address and port are within range.")
00154     {
00155         // Fails on ARMv7.
00156         // REQUIRE_THROWS_AS(
00157         //     IPAddress(static_cast<unsigned long>(-1), 0), std::out_of_range);
00158         REQUIRE_THROWS_AS(
00159             IPAddress(0, static_cast<unsigned int>(-1)), std::out_of_range);
00160         // This causes a compilation error on ARMv7.
00161         // REQUIRE_THROWS_AS(IPAddress(0xFFFFFFF, 65535), std::out_of_range);
00162         REQUIRE_THROWS_AS(IPAddress(0xFFFFFFF, 65536), std::out_of_range);
00163     }
00164 }
00165
00166
00167
00168 TEST_CASE("IPAddress's can be constructed from strings.", "[IPAddress]")
00169 {
00170     SECTION("Without a port number.")
00171     {
00172         REQUIRE(IPAddress("192.168.32.128") == IPAddress(0xC0A82080));
00173     }
00174     SECTION("With a port number.")
00175     {
00176         REQUIRE(IPAddress("192.168.32.128:443") == IPAddress(0xC0A82080, 443));
00177     }
00178     SECTION("And will raise std::invalid_argument if it cannot be parsed.")
00179     {
00180         // Errors.
00181         REQUIRE_THROWS_AS(IPAddress("-1.2.3.4"), std::invalid_argument);
00182         REQUIRE_THROWS_AS(IPAddress("1.-2.3.4"), std::invalid_argument);
00183         REQUIRE_THROWS_AS(IPAddress("1.2.-3.4"), std::invalid_argument);
00184         REQUIRE_THROWS_AS(IPAddress("1.2.3.-4"), std::invalid_argument);

```

```

00185     REQUIRE_THROWS_AS(IPAddress("+1.2.3.4"), std::invalid_argument);
00186     REQUIRE_THROWS_AS(IPAddress("1.+2.3.4"), std::invalid_argument);
00187     REQUIRE_THROWS_AS(IPAddress("1.2.+3.4"), std::invalid_argument);
00188     REQUIRE_THROWS_AS(IPAddress("1.2.3.+4"), std::invalid_argument);
00189     REQUIRE_THROWS_AS(IPAddress("1."), std::invalid_argument);
00190     REQUIRE_THROWS_AS(IPAddress("."), std::invalid_argument);
00191     REQUIRE_THROWS_AS(IPAddress("1.2"), std::invalid_argument);
00192     REQUIRE_THROWS_AS(IPAddress("1.2."), std::invalid_argument);
00193     REQUIRE_THROWS_AS(IPAddress("1.2.3"), std::invalid_argument);
00194     REQUIRE_THROWS_AS(IPAddress("1.2.3."), std::invalid_argument);
00195     REQUIRE_THROWS_AS(IPAddress("1.2.3.4."), std::invalid_argument);
00196     REQUIRE_THROWS_AS(IPAddress("1.2.3.4.5"), std::invalid_argument);
00197     REQUIRE_THROWS_AS(IPAddress("a.2.3.4"), std::invalid_argument);
00198     REQUIRE_THROWS_AS(IPAddress("1.b.3.4"), std::invalid_argument);
00199     REQUIRE_THROWS_AS(IPAddress("1.2.c.4"), std::invalid_argument);
00200     REQUIRE_THROWS_AS(IPAddress("1.2.3.d"), std::invalid_argument);
00201     REQUIRE_THROWS_AS(
00202         IPAddress("192.168.32.128.443"), std::invalid_argument);
00203     REQUIRE_THROWS_AS(IPAddress("192.168.128:443"), std::invalid_argument);
00204     REQUIRE_THROWS_AS(
00205         IPAddress("192:168:32:128:443"), std::invalid_argument);
00206     REQUIRE_THROWS_AS(
00207         IPAddress("192.1b8.32.128:443"), std::invalid_argument);
00208 // Check error string.
00209     REQUIRE_THROWS_WITH(
00210         IPAddress("-1.2.3.4"), "Invalid IP address string.");
00211     REQUIRE_THROWS_WITH(
00212         IPAddress("1.-2.3.4"), "Invalid IP address string.");
00213     REQUIRE_THROWS_WITH(
00214         IPAddress("1.2.-3.4"), "Invalid IP address string.");
00215     REQUIRE_THROWS_WITH(
00216         IPAddress("1.2.3.-4"), "Invalid IP address string.");
00217     REQUIRE_THROWS_WITH(
00218         IPAddress("+1.2.3.4"), "Invalid IP address string.");
00219     REQUIRE_THROWS_WITH(
00220         IPAddress("1.+2.3.4"), "Invalid IP address string.");
00221     REQUIRE_THROWS_WITH(
00222         IPAddress("1.2.+3.4"), "Invalid IP address string.");
00223     REQUIRE_THROWS_WITH(
00224         IPAddress("1.2.3.+4"), "Invalid IP address string.");
00225     REQUIRE_THROWS_WITH(
00226         IPAddress("1."), "Invalid IP address string.");
00227     REQUIRE_THROWS_WITH(
00228         IPAddress("1."), "Invalid IP address string.");
00229     REQUIRE_THROWS_WITH(
00230         IPAddress("1.2"), "Invalid IP address string.");
00231     REQUIRE_THROWS_WITH(
00232         IPAddress("1.2."), "Invalid IP address string.");
00233     REQUIRE_THROWS_WITH(
00234         IPAddress("1.2.3"), "Invalid IP address string.");
00235     REQUIRE_THROWS_WITH(
00236         IPAddress("1.2.3."), "Invalid IP address string.");
00237     REQUIRE_THROWS_WITH(
00238         IPAddress("1.2.3.4."), "Invalid IP address string.");
00239     REQUIRE_THROWS_WITH(
00240         IPAddress("1.2.3.4.5"), "Invalid IP address string.");
00241     REQUIRE_THROWS_WITH(
00242         IPAddress("a.2.3.4"), "Invalid IP address string.");
00243     REQUIRE_THROWS_WITH(
00244         IPAddress("1.b.3.4"), "Invalid IP address string.");
00245     REQUIRE_THROWS_WITH(
00246         IPAddress("1.2.c.4"), "Invalid IP address string.");
00247     REQUIRE_THROWS_WITH(
00248         IPAddress("1.2.3.d"), "Invalid IP address string.");
00249     REQUIRE_THROWS_WITH(
00250         IPAddress("192.168.32.128.443"), "Invalid IP address string.");
00251     REQUIRE_THROWS_WITH(
00252         IPAddress("192.168.128:443"), "Invalid IP address string.");
00253     REQUIRE_THROWS_WITH(
00254         IPAddress("192:168:32:128:443"), "Invalid IP address string.");
00255     REQUIRE_THROWS_WITH(
00256         IPAddress("192.1b8.32.128:443"), "Invalid IP address string.");
00257 }
00258 SECTION("And ensures address and port are within range.")
00259 {
00260 // Errors
00261 // The following 2 tests had to be removed because they do not work on
00262 // systems that use 16 bit integers.
00263 // REQUIRE_THROWS_AS(IPAddress("1.2.3.4:-1"), std::out_of_range);
00264 // REQUIRE_THROWS_AS(IPAddress("1.2.3.4:65536"), std::out_of_range);
00265 REQUIRE_THROWS_AS(IPAddress("256.2.3.4"), std::out_of_range);

```

```
00266     REQUIRE_THROWS_AS(IPAddress("1.256.3.4"), std::out_of_range);
00267     REQUIRE_THROWS_AS(IPAddress("1.2.256.4"), std::out_of_range);
00268     REQUIRE_THROWS_AS(IPAddress("1.2.3.256"), std::out_of_range);
00269     // Check error string.
00270     REQUIRE_THROWS_WITH(
00271         IPAddress("256.2.3.4"),
00272         "Address octet (256) is outside of the allowed range (0 - 255).");
00273     REQUIRE_THROWS_WITH(
00274         IPAddress("1.256.3.4"),
00275         "Address octet (256) is outside of the allowed range (0 - 255).");
00276     REQUIRE_THROWS_WITH(
00277         IPAddress("1.2.256.4"),
00278         "Address octet (256) is outside of the allowed range (0 - 255).");
00279     REQUIRE_THROWS_WITH(
00280         IPAddress("1.2.3.256"),
00281         "Address octet (256) is outside of the allowed range (0 - 255).");
00282 }
00283 }
00284
00285
00286 TEST_CASE("IPAddress's are copyable.", "[IPAddress]")
00287 {
00288     IPAddress address_a(0x00000000, 0);
00289     IPAddress address_b(0xFFFFFFFF, 65535);
00290     IPAddress address_a_copy = address_a;
00291     IPAddress address_b_copy(address_b);
00292     REQUIRE(&address_a != &address_a_copy);
00293     REQUIRE(address_a == address_a_copy);
00294     REQUIRE(&address_b != &address_b_copy);
00295     REQUIRE(address_b == address_b_copy);
00296 }
00297
00298
00299 TEST_CASE("The port of an IP address can be changed during a copy.",
00300         "[IPAddress]")
00301 {
00302     IPAddress address_a(0x00000000, 0);
00303     IPAddress address_b(0xFFFFFFFF, 65535);
00304     IPAddress address_a_copy(address_a, 65535);
00305     IPAddress address_b_copy(address_b, 0);
00306     REQUIRE(&address_a != &address_a_copy);
00307     REQUIRE(address_a.address() == address_a_copy.address());
00308     REQUIRE(address_a.port() == 0);
00309     REQUIRE(address_a_copy.port() == 65535);
00310     REQUIRE(&address_b != &address_b_copy);
00311     REQUIRE(address_b.address() == address_b_copy.address());
00312     REQUIRE(address_b.port() == 65535);
00313     REQUIRE(address_b_copy.port() == 0);
00314     SECTION("And ensures port number is within range.")
00315     {
00316         REQUIRE_THROWS_AS(
00317             IPAddress(address_a, static_cast<unsigned int>(-1)),
00318             std::out_of_range);
00319         REQUIRE_THROWS_AS(
00320             IPAddress(address_b, static_cast<unsigned int>(-1)),
00321             std::out_of_range);
00322         REQUIRE_THROWS_AS(IPAddress(address_a, 65536), std::out_of_range);
00323         REQUIRE_THROWS_AS(IPAddress(address_b, 65536), std::out_of_range);
00324     }
00325 }
00326
00327
00328 TEST_CASE("IPAddress's are movable.", "[IPAddress]")
00329 {
00330     IPAddress address_a(0x00000000, 0);
00331     IPAddress address_b(0xFFFFFFFF, 65535);
00332     IPAddress address_a_moved = std::move(address_a);
00333     IPAddress address_b_moved(std::move(address_b));
00334     REQUIRE(address_a_moved == IPAddress(0x00000000, 0));
00335     REQUIRE(address_b_moved == IPAddress(0xFFFFFFFF, 65535));
00336 }
00337
00338
00339 TEST_CASE("IPAddress's are assignable.", "[IPAddress]")
00340 {
00341     IPAddress address_a(0x00000000, 0);
00342     IPAddress address_b(0xFFFFFFFF, 65535);
00343     REQUIRE(address_a == IPAddress(0x00000000, 0));
00344     address_a = address_b;
00345     REQUIRE(address_a == IPAddress(0xFFFFFFFF, 65535));
00346 }
```

```

00347
00348
00349 TEST_CASE("IPAddress's are assignable (by move semantics).", "[IPAddress]")
00350 {
00351     IPAddress address_a(0x00000000, 0);
00352     IPAddress address_b(0xFFFFFFFF, 65535);
00353     REQUIRE(address_a == IPAddress(0x00000000, 0));
00354     address_a = std::move(address_b);
00355     REQUIRE(address_a == IPAddress(0xFFFFFFFF, 65535));
00356 }
00357
00358
00359 TEST_CASE("IPAddress's are printable", "[IPAddress]")
00360 {
00361     SECTION("Without a port number.")
00362     {
00363         REQUIRE(str(IPAddress("192.168.32.128")) == "192.168.32.128");
00364     }
00365     SECTION("With a port number.")
00366     {
00367         REQUIRE(str(IPAddress("192.168.32.128:443")) == "192.168.32.128:443");
00368     }
00369 }
00370
00371
00372 TEST_CASE("dnslookup finds IPAddress's by domain name.", "[IPAddress]")
00373 {
00374     SECTION("When they are available.")
00375     {
00376         REQUIRE(dnslookup("localhost") == IPAddress("127.0.0.1"));
00377     }
00378     SECTION("And throws DNSLookupError otherwise.")
00379     {
00380         REQUIRE_THROWS_AS(dnslookup("abc.efg"), DNSLookupError);
00381     }
00382 }

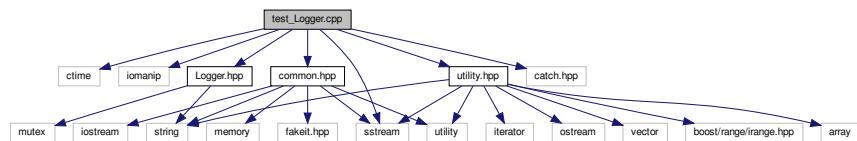
```

16.225 test_Logger.cpp File Reference

```

#include <ctime>
#include <iomanip>
#include <sstream>
#include <catch.hpp>
#include "Logger.hpp"
#include "utility.hpp"
#include "common.hpp"
Include dependency graph for test_Logger.cpp:

```



Functions

- **TEST_CASE** ("The [Logger](#) level can be set and retrieved with the static 'level' " "method", "[[Logger](#)]")
- **TEST_CASE** ("The [Logger](#) can log to stdout with the static 'log' method if the " "logger's level is the same or higher than the loglevel of the " "message.", "[[Logger](#)]")
- **TEST_CASE** ("All logged messages have a timestamp (this can sometimes fail if " "the clock ticks over a second during the test).", "[[Logger](#)]")

16.225.1 Function Documentation

16.225.1.1 TEST_CASE() [1/3]

```
TEST_CASE (
    "The Logger level can be set and retrieved with the static 'level' \"method\",
    "" [Logger] )
```

Definition at line 30 of file [test_Logger.cpp](#).

16.225.1.2 TEST_CASE() [2/3]

```
TEST_CASE (
    "The Logger can log to stdout with the static 'log' method if the \" logger's level
is the same or higher than the loglevel of the \" message.\",
    "" [Logger] )
```

Definition at line 45 of file [test_Logger.cpp](#).

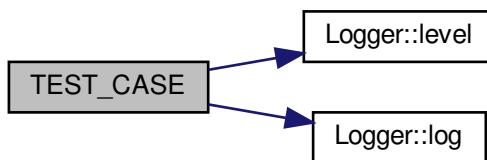
16.225.1.3 TEST_CASE() [3/3]

```
TEST_CASE (
    "All logged messages have a timestamp (this can sometimes fail if \" the clock ticks
over a second during the test).\",
    "" [Logger] )
```

Definition at line 99 of file [test_Logger.cpp](#).

References [Logger::level\(\)](#), and [Logger::log\(\)](#).

Here is the call graph for this function:



16.226 test_Logger.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <ctime>
00019 #include <iomanip>
00020 #include <sstream>
00021
00022 #include <catch.hpp>
00023
00024 #include "Logger.hpp"
00025 #include "utility.hpp"
00026
00027 #include "common.hpp"
00028
00029
00030 TEST_CASE("The Logger level can be set and retrieved with the static 'level' "
00031           "method", "[Logger]")
00032 {
00033     REQUIRE(Logger::level() == 0);
00034
00035     for (unsigned int i = 0; i <= 10; i++)
00036     {
00037         Logger::level(i);
00038         REQUIRE(Logger::level() == i);
00039     }
00040
00041     Logger::level(0);
00042 }
00043
00044
00045 TEST_CASE("The Logger can log to std::cout with the static 'log' method if the "
00046           "logger's level is the same or higher than the loglevel of the "
00047           "message.", "[Logger]")
00048 {
00049     MockCout mock_cout;
00050     SECTION("No message can be logged if the logger's level is 0.")
00051     {
00052         Logger::level(0);
00053         Logger::log("a log message");
00054         REQUIRE(mock_cout.buffer().empty());
00055         mock_cout.reset();
00056         Logger::log("another log message");
00057         REQUIRE(mock_cout.buffer().empty());
00058     }
00059     SECTION("The default level of the 'log' method is 1.")
00060     {
00061         Logger::level(0);
00062         Logger::log("not logged");
00063         REQUIRE(mock_cout.buffer().empty());
00064         Logger::level(1);
00065         Logger::log("a log message");
00066         REQUIRE(mock_cout.buffer().substr(21) == "a log message\n");
00067         mock_cout.reset();
00068         Logger::log("another log message");
00069         REQUIRE(mock_cout.buffer().substr(21) == "another log message\n");
00070     }
00071     SECTION("A level of 0 will be changed to level 1.")
00072     {
00073         Logger::level(0);
00074         Logger::log(0, "not logged");
00075         REQUIRE(mock_cout.buffer().empty());
00076         Logger::level(1);
00077         Logger::log(0, "a log message");
```

```

00078     REQUIRE(mock_cout.buffer().substr(21) == "a log message\n");
00079     mock_cout.reset();
00080     Logger::log(0, "another log message");
00081     REQUIRE(mock_cout.buffer().substr(21) == "another log message\n");
00082 }
00083 SECTION("The level can be set from 1 to 65535.")
00084 {
00085     Logger::level(0);
00086     Logger::log(65535, "not logged");
00087     REQUIRE(mock_cout.buffer().empty());
00088     Logger::level(65535);
00089     Logger::log(65535, "a log message");
00090     REQUIRE(mock_cout.buffer().substr(21) == "a log message\n");
00091     mock_cout.reset();
00092     Logger::log(65535, "another log message");
00093     REQUIRE(mock_cout.buffer().substr(21) == "another log message\n");
00094 }
00095 Logger::level(0);
00096 }
00097
00098
00099 TEST_CASE("All logged messages have a timestamp (this can sometimes fail if "
00100     "the clock ticks over a second during the test).", "[Logger]")
00101 {
00102     Logger::level(1);
00103     MockCout mock_cout;
00104     auto t = std::time(nullptr);
00105     auto tm = *std::localtime(&t);
00106     Logger::log("timestamped log message");
00107     REQUIRE(
00108         mock_cout.buffer() ==
00109         str(std::put_time(&tm, "%Y-%m-%d %H:%M:%S")) +
00110         " timestamped log message\n");
00111     Logger::level(0);
00112 }

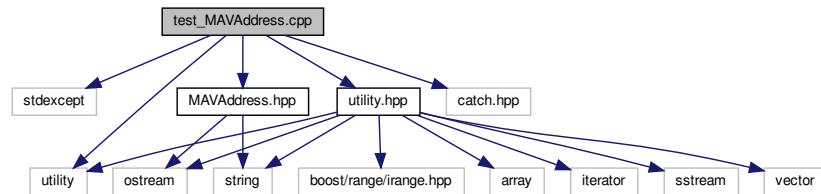
```

16.227 test_MAVAddress.cpp File Reference

```

#include <stdexcept>
#include <utility>
#include <catch.hpp>
#include "MAVAddress.hpp"
#include "utility.hpp"
Include dependency graph for test_MAVAddress.cpp:

```



Functions

- `TEST_CASE ("MAVAddress's store a system and component ID.", "[MAVAddress]")`
- `TEST_CASE ("MAVAddress's are comparable.", "[MAVAddress]")`
- `TEST_CASE ("MAVAddress's can be constructed from a numeric address.", "[MAVAddress]")`

- [TEST_CASE \("MAVAddress's can be constructed from System and Component ID's.", "\[**MAVAddress**\]""\)](#)
- [TEST_CASE \("MAVAddress's can be constructed from strings.", "\[**MAVAddress**\]""\)](#)
- [TEST_CASE \("MAVAddress's are copyable.", "\[**MAVAddress**\]""\)](#)
- [TEST_CASE \("MAVAddress's are movable.", "\[**MAVAddress**\]""\)](#)
- [TEST_CASE \("MAVAddress's are assignable.", "\[**MAVAddress**\]""\)](#)
- [TEST_CASE \("MAVAddress's are printable", "\[**MAVAddress**\]""\)](#)

Variables

- [MAVAddress address_b \(255, 255\)](#)
- [address_a = std::move\(address_b\)](#)

16.227.1 Function Documentation

16.227.1.1 TEST_CASE() [1/9]

```
TEST_CASE (
    "MAVAddress's store a system and component ID." ,
    "" [MAVAddress] )
```

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

16.227.1.2 TEST_CASE() [2/9]

```
TEST_CASE (
    "MAVAddress's are comparable." ,
    "" [MAVAddress] )
```

Definition at line [48](#) of file [test_MAVAddress.cpp](#).

16.227.1.3 TEST_CASE() [3/9]

```
TEST_CASE (
    "MAVAddress's can be constructed from a numeric address." ,
    "" [MAVAddress] )
```

Definition at line [145](#) of file [test_MAVAddress.cpp](#).

16.227.1.4 TEST_CASE() [4/9]

```
TEST_CASE (
    "MAVAddress's can be constructed from System and Component ID's." ,
    "" [MAVAddress] )
```

Definition at line 163 of file [test_MAVAddress.cpp](#).

16.227.1.5 TEST_CASE() [5/9]

```
TEST_CASE (
    "MAVAddress's can be constructed from strings." ,
    "" [MAVAddress] )
```

Definition at line 195 of file [test_MAVAddress.cpp](#).

16.227.1.6 TEST_CASE() [6/9]

```
TEST_CASE (
    "MAVAddress's are copyable." ,
    "" [MAVAddress] )
```

Definition at line 264 of file [test_MAVAddress.cpp](#).

References [address_a](#), and [address_b](#).

16.227.1.7 TEST_CASE() [7/9]

```
TEST_CASE (
    "MAVAddress's are movable." ,
    "" [MAVAddress] )
```

Definition at line 277 of file [test_MAVAddress.cpp](#).

References [address_a](#), and [address_b](#).

16.227.1.8 TEST_CASE() [8/9]

```
TEST_CASE (
    "MAVAddress's are assignable." ,
    "" [MAVAddress] )
```

Definition at line 288 of file [test_MAVAddress.cpp](#).

References [address_a](#), and [address_b](#).

16.227.1.9 TEST_CASE() [9/9]

```
TEST_CASE (
    "MAVAddress's are printable" ,
    "" [MAVAddress] )
```

Definition at line 308 of file [test_MAVAddress.cpp](#).

16.227.2 Variable Documentation

16.227.2.1 address_a

```
address_a = std::move(address\_b)
```

Definition at line 303 of file [test_MAVAddress.cpp](#).

16.227.2.2 address_b

```
MAVAddress address_b(255, 255)
```

Initial value:

```
{  
    MAVAddress address_a(0, 0)
```

16.228 test_MAVAddress.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <stdexcept>
00019 #include <utility>
00020
00021 #include <catch.hpp>
00022
00023 #include "MAVAddress.hpp"
00024 #include "utility.hpp"
00025
00026
00027 TEST_CASE("MAVAddress's store a system and component ID.", "[MAVAddress]")
00028 {
00029     MAVAddress a(0, 0);
00030     REQUIRE(a.system() == 0);
00031     REQUIRE(a.component() == 0);
00032     REQUIRE(a.address() == 0);
00033     MAVAddress b(255, 255);
00034     REQUIRE(b.system() == 255);
00035     REQUIRE(b.component() == 255);
00036     REQUIRE(b.address() == 0xFFFF);
00037     MAVAddress c(255, 0);
00038     REQUIRE(c.system() == 255);
00039     REQUIRE(c.component() == 0);
00040     REQUIRE(c.address() == 0xFF00);
00041     MAVAddress d(0, 255);
00042     REQUIRE(d.system() == 0);
00043     REQUIRE(d.component() == 255);
00044     REQUIRE(d.address() == 0x00FF);
00045 }
00046
00047
00048 TEST_CASE("MAVAddress's are comparable.", "[MAVAddress]")
00049 {
00050     SECTION("with ==")
00051     {
00052         REQUIRE(MAVAddress(0, 0) == MAVAddress(0, 0));
00053         REQUIRE(MAVAddress(128, 128) == MAVAddress(128, 128));
00054         REQUIRE(MAVAddress(255, 255) == MAVAddress(255, 255));
00055         REQUIRE_FALSE(MAVAddress(0, 0) == MAVAddress(0, 1));
00056         REQUIRE_FALSE(MAVAddress(0, 0) == MAVAddress(1, 0));
00057         REQUIRE_FALSE(MAVAddress(255, 255) == MAVAddress(255, 254));
00058         REQUIRE_FALSE(MAVAddress(255, 255) == MAVAddress(254, 255));
00059     }
00060     SECTION("with !=")
00061     {
00062         REQUIRE(MAVAddress(0, 0) != MAVAddress(0, 1));
00063         REQUIRE(MAVAddress(0, 0) != MAVAddress(1, 0));
00064         REQUIRE(MAVAddress(255, 255) != MAVAddress(255, 254));
00065         REQUIRE(MAVAddress(255, 255) != MAVAddress(254, 255));
00066         REQUIRE_FALSE(MAVAddress(0, 0) != MAVAddress(0, 0));
00067         REQUIRE_FALSE(MAVAddress(128, 128) != MAVAddress(128, 128));
00068         REQUIRE_FALSE(MAVAddress(255, 255) != MAVAddress(255, 255));
00069     }
00070     SECTION("with <")
00071     {
00072         REQUIRE(MAVAddress(0, 0) < MAVAddress(0, 1));
00073         REQUIRE(MAVAddress(0, 0) < MAVAddress(1, 0));
00074         REQUIRE(MAVAddress(0, 1) < MAVAddress(1, 0));
00075         REQUIRE(MAVAddress(255, 254) < MAVAddress(255, 255));
00076         REQUIRE(MAVAddress(254, 255) < MAVAddress(255, 255));
00077         REQUIRE(MAVAddress(0, 0) < MAVAddress(255, 255));
00078     }
00079 }
```

```

00078     REQUIRE_FALSE(MAVAddress(0, 1) < MAVAddress(0, 0));
00079     REQUIRE_FALSE(MAVAddress(1, 0) < MAVAddress(0, 0));
00080     REQUIRE_FALSE(MAVAddress(1, 0) < MAVAddress(0, 1));
00081     REQUIRE_FALSE(MAVAddress(255, 255) < MAVAddress(255, 254));
00082     REQUIRE_FALSE(MAVAddress(255, 255) < MAVAddress(254, 255));
00083     REQUIRE_FALSE(MAVAddress(0, 0) < MAVAddress(0, 0));
00084     REQUIRE_FALSE(MAVAddress(128, 128) < MAVAddress(128, 128));
00085     REQUIRE_FALSE(MAVAddress(255, 255) < MAVAddress(255, 255));
00086     REQUIRE_FALSE(MAVAddress(255, 255) < MAVAddress(0, 0));
00087 }
00088 SECTION("with >")
00089 {
00090     REQUIRE(MAVAddress(0, 1) > MAVAddress(0, 0));
00091     REQUIRE(MAVAddress(1, 0) > MAVAddress(0, 0));
00092     REQUIRE(MAVAddress(1, 0) > MAVAddress(0, 1));
00093     REQUIRE(MAVAddress(255, 255) > MAVAddress(255, 254));
00094     REQUIRE(MAVAddress(255, 255) > MAVAddress(254, 254));
00095     REQUIRE(MAVAddress(255, 255) > MAVAddress(0, 0));
00096     REQUIRE_FALSE(MAVAddress(0, 0) > MAVAddress(0, 1));
00097     REQUIRE_FALSE(MAVAddress(0, 0) > MAVAddress(1, 0));
00098     REQUIRE_FALSE(MAVAddress(0, 1) > MAVAddress(1, 0));
00099     REQUIRE_FALSE(MAVAddress(255, 254) > MAVAddress(255, 255));
00100    REQUIRE_FALSE(MAVAddress(254, 255) > MAVAddress(255, 255));
00101    REQUIRE_FALSE(MAVAddress(0, 0) > MAVAddress(0, 0));
00102    REQUIRE_FALSE(MAVAddress(128, 128) > MAVAddress(128, 128));
00103    REQUIRE_FALSE(MAVAddress(255, 255) > MAVAddress(255, 255));
00104    REQUIRE_FALSE(MAVAddress(0, 0) > MAVAddress(255, 255));
00105 }
00106 SECTION("with <=")
00107 {
00108     REQUIRE(MAVAddress(0, 0) <= MAVAddress(0, 0));
00109     REQUIRE(MAVAddress(128, 128) <= MAVAddress(128, 128));
00110     REQUIRE(MAVAddress(255, 255) <= MAVAddress(255, 255));
00111     REQUIRE(MAVAddress(0, 0) <= MAVAddress(0, 1));
00112     REQUIRE(MAVAddress(0, 0) <= MAVAddress(1, 0));
00113     REQUIRE(MAVAddress(0, 1) <= MAVAddress(1, 0));
00114     REQUIRE(MAVAddress(255, 254) <= MAVAddress(255, 255));
00115     REQUIRE(MAVAddress(254, 255) <= MAVAddress(255, 255));
00116     REQUIRE(MAVAddress(0, 0) <= MAVAddress(255, 255));
00117     REQUIRE_FALSE(MAVAddress(0, 1) <= MAVAddress(0, 0));
00118     REQUIRE_FALSE(MAVAddress(1, 0) <= MAVAddress(0, 0));
00119     REQUIRE_FALSE(MAVAddress(1, 0) <= MAVAddress(0, 1));
00120     REQUIRE_FALSE(MAVAddress(255, 255) <= MAVAddress(255, 254));
00121     REQUIRE_FALSE(MAVAddress(255, 255) <= MAVAddress(254, 255));
00122     REQUIRE_FALSE(MAVAddress(255, 255) <= MAVAddress(0, 0));
00123 }
00124 SECTION("with >=")
00125 {
00126     REQUIRE(MAVAddress(0, 0) >= MAVAddress(0, 0));
00127     REQUIRE(MAVAddress(128, 128) >= MAVAddress(128, 128));
00128     REQUIRE(MAVAddress(255, 255) >= MAVAddress(255, 255));
00129     REQUIRE(MAVAddress(0, 1) >= MAVAddress(0, 0));
00130     REQUIRE(MAVAddress(1, 0) >= MAVAddress(0, 0));
00131     REQUIRE(MAVAddress(1, 0) >= MAVAddress(0, 1));
00132     REQUIRE(MAVAddress(255, 255) >= MAVAddress(255, 254));
00133     REQUIRE(MAVAddress(255, 255) >= MAVAddress(254, 255));
00134     REQUIRE(MAVAddress(255, 255) >= MAVAddress(0, 0));
00135     REQUIRE_FALSE(MAVAddress(0, 0) >= MAVAddress(0, 1));
00136     REQUIRE_FALSE(MAVAddress(0, 0) >= MAVAddress(1, 0));
00137     REQUIRE_FALSE(MAVAddress(0, 1) >= MAVAddress(1, 0));
00138     REQUIRE_FALSE(MAVAddress(255, 254) >= MAVAddress(255, 255));
00139     REQUIRE_FALSE(MAVAddress(254, 255) >= MAVAddress(255, 255));
00140     REQUIRE_FALSE(MAVAddress(0, 0) >= MAVAddress(255, 255));
00141 }
00142 }
00143
00144
00145 TEST_CASE("MAVAddress's can be constructed from a numeric address.",
00146             "[MAVAddress]")
00147 {
00148     REQUIRE(MAVAddress(0x0000) == MAVAddress(0x0000));
00149     REQUIRE(MAVAddress(0x8000) == MAVAddress(0x8000));
00150     REQUIRE(MAVAddress(0x0080) == MAVAddress(0x0080));
00151     REQUIRE(MAVAddress(0xFF00) == MAVAddress(0xFF00));
00152     REQUIRE(MAVAddress(0x00FF) == MAVAddress(0x00FF));
00153     REQUIRE(MAVAddress(0xFFFF) == MAVAddress(0xFFFF));
00154     SECTION("And ensures System and Component ID's are within range.")
00155 {
00156     REQUIRE_THROWS_AS(MAVAddress(static_cast<unsigned int>(0x0000 - 1)),
00157                         std::out_of_range);
00158     REQUIRE_THROWS_AS(MAVAddress(0xFFFF + 1), std::out_of_range);

```

```

00159     }
00160 }
00161
00162
00163 TEST_CASE("MAVAddress's can be constructed from System and Component ID's.", "[MAVAddress]")
00164 {
00165     REQUIRE(MAVAddress(0, 0) == MAVAddress(0x0000));
00166     REQUIRE(MAVAddress(128, 0) == MAVAddress(0x8000));
00167     REQUIRE(MAVAddress(0, 128) == MAVAddress(0x0080));
00168     REQUIRE(MAVAddress(255, 0) == MAVAddress(0xFF00));
00169     REQUIRE(MAVAddress(0, 255) == MAVAddress(0x00FF));
00170     REQUIRE(MAVAddress(255, 255) == MAVAddress(0xFFFF));
00171     SECTION("And ensures System and Component ID's are within range.")
00172 {
00173     // Errors
00174     REQUIRE_THROWS_AS(MAVAddress(static_cast<unsigned int>(-1), 0),
00175                         std::out_of_range);
00176     REQUIRE_THROWS_AS(MAVAddress(0, static_cast<unsigned int>(-1)),
00177                         std::out_of_range);
00178     REQUIRE_THROWS_AS(MAVAddress(256, 255), std::out_of_range);
00179     REQUIRE_THROWS_AS(MAVAddress(255, 256), std::out_of_range);
00180     REQUIRE_THROWS_AS(MAVAddress(256, 256), std::out_of_range);
00181     // Error messages.
00182     REQUIRE_THROWS_WITH(
00183         MAVAddress(256, 255),
00184         "System ID (256) is outside of the allowed range (0 - 255).");
00185     REQUIRE_THROWS_WITH(
00186         MAVAddress(255, 256),
00187         "Component ID (256) is outside of the allowed range (0 - 255).");
00188     // NOTE: MAVAddress(256, 256) is not checked because the order of
00189     //       checking the inputs is not defined.
00190 }
00191 }
00192 }
00193
00194
00195 TEST_CASE("MAVAddress's can be constructed from strings.", "[MAVAddress]")
00196 {
00197     REQUIRE(MAVAddress("0.0") == MAVAddress(0x0000));
00198     REQUIRE(MAVAddress("128.0") == MAVAddress(0x8000));
00199     REQUIRE(MAVAddress("0.128") == MAVAddress(0x0080));
00200     REQUIRE(MAVAddress("255.0") == MAVAddress(0xFF00));
00201     REQUIRE(MAVAddress("0.255") == MAVAddress(0x00FF));
00202     REQUIRE(MAVAddress("255.255") == MAVAddress(0xFFFF));
00203     REQUIRE(MAVAddress("000.000") == MAVAddress(0x0000));
00204     REQUIRE(MAVAddress("001.001") == MAVAddress(0x0101));
00205     REQUIRE(MAVAddress("010.010") == MAVAddress(0x0A0A));
00206     REQUIRE(MAVAddress("192.168") == MAVAddress(192, 168));
00207     SECTION("And ensures the address string is valid.")
00208 {
00209     // Errors
00210     REQUIRE_THROWS_AS(MAVAddress("1"), std::invalid_argument);
00211     REQUIRE_THROWS_AS(MAVAddress("1."), std::invalid_argument);
00212     REQUIRE_THROWS_AS(MAVAddress("1.2"), std::invalid_argument);
00213     REQUIRE_THROWS_AS(MAVAddress("1.2.3"), std::invalid_argument);
00214     REQUIRE_THROWS_AS(MAVAddress("a.2.3"), std::invalid_argument);
00215     REQUIRE_THROWS_AS(MAVAddress("1.b.3"), std::invalid_argument);
00216     REQUIRE_THROWS_AS(MAVAddress("1.2.c"), std::invalid_argument);
00217     REQUIRE_THROWS_AS(MAVAddress("+1.0"), std::invalid_argument);
00218     REQUIRE_THROWS_AS(MAVAddress("0.+1"), std::invalid_argument);
00219     REQUIRE_THROWS_AS(MAVAddress("-1.0"), std::invalid_argument);
00220     REQUIRE_THROWS_AS(MAVAddress("0.-1"), std::invalid_argument);
00221     // Error message.
00222     REQUIRE_THROWS_WITH(
00223         MAVAddress("1"), "Invalid MAVLink address string.");
00224     REQUIRE_THROWS_WITH(
00225         MAVAddress("1."), "Invalid MAVLink address string.");
00226     REQUIRE_THROWS_WITH(
00227         MAVAddress("1.2"), "Invalid MAVLink address string.");
00228     REQUIRE_THROWS_WITH(
00229         MAVAddress("1.2.3"), "Invalid MAVLink address string.");
00230     REQUIRE_THROWS_WITH(
00231         MAVAddress("a.2.3"), "Invalid MAVLink address string.");
00232     REQUIRE_THROWS_WITH(
00233         MAVAddress("1.b.3"), "Invalid MAVLink address string.");
00234     REQUIRE_THROWS_WITH(
00235         MAVAddress("1.2.c"), "Invalid MAVLink address string.");
00236     REQUIRE_THROWS_WITH(
00237         MAVAddress("+1.0"), "Invalid MAVLink address string.");
00238     REQUIRE_THROWS_WITH(
00239         MAVAddress("0.+1"), "Invalid MAVLink address string.");

```

```

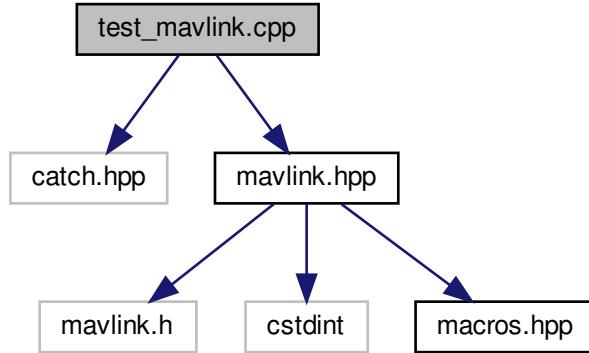
00240     REQUIRE_THROWS_WITH(
00241         MAVAddress("-1.0"), "Invalid MAVLink address string.");
00242     REQUIRE_THROWS_WITH(
00243         MAVAddress("0.-1"), "Invalid MAVLink address string.");
00244 }
00245 SECTION("And ensures System and Component ID's are within range.")
00246 {
00247     // Errors
00248     REQUIRE_THROWS_AS(MAVAddress("256.255"), std::out_of_range);
00249     REQUIRE_THROWS_AS(MAVAddress("255.256"), std::out_of_range);
00250     REQUIRE_THROWS_AS(MAVAddress("256.256"), std::out_of_range);
00251     // Error messages.
00252     REQUIRE_THROWS_WITH(
00253         MAVAddress("256.255"),
00254         "System ID (256) is outside of the allowed range (0 - 255).");
00255     REQUIRE_THROWS_WITH(
00256         MAVAddress("255.256"),
00257         "Component ID (256) is outside of the allowed range (0 - 255).");
00258     // NOTE: MAVAddress("256.256") is not checked because the order of
00259     //       checking the inputs is not defined.
00260 }
00261 }
00262
00263
00264 TEST_CASE("MAVAddress's are copyable.", "[MAVAddress]")
00265 {
00266     MAVAddress address_a(0, 0);
00267     MAVAddress address_b(255, 255);
00268     MAVAddress address_a_copy = address_a;
00269     MAVAddress address_b_copy(address_b);
00270     REQUIRE(&address_a != &address_a_copy);
00271     REQUIRE(address_a == address_a_copy);
00272     REQUIRE(&address_b != &address_b_copy);
00273     REQUIRE(address_b == address_b_copy);
00274 }
00275
00276
00277 TEST_CASE("MAVAddress's are movable.", "[MAVAddress]")
00278 {
00279     MAVAddress address_a(0, 0);
00280     MAVAddress address_b(255, 255);
00281     MAVAddress address_a_moved = std::move(address_a);
00282     MAVAddress address_b_moved(std::move(address_b));
00283     REQUIRE(address_a_moved == MAVAddress(0, 0));
00284     REQUIRE(address_b_moved == MAVAddress(255, 255));
00285 }
00286
00287
00288 TEST_CASE("MAVAddress's are assignable.", "[MAVAddress]")
00289 {
00290     MAVAddress address_a(0, 0);
00291     MAVAddress address_b(255, 255);
00292     REQUIRE(address_a == MAVAddress(0, 0));
00293     address_a = address_b;
00294     REQUIRE(address_a == MAVAddress(255, 255));
00295 }
00296
00297
00298 TEST_CASE("MAVAddress's are assignable (by move semantics).", "[MAVAddress]")
00299 {
00300     MAVAddress address_a(0, 0);
00301     MAVAddress address_b(255, 255);
00302     REQUIRE(address_a == MAVAddress(0, 0));
00303     address_a = std::move(address_b);
00304     REQUIRE(address_a == MAVAddress(255, 255));
00305 }
00306
00307
00308 TEST_CASE("MAVAddress's are printable", "[MAVAddress]")
00309 {
00310     REQUIRE(str(MAVAddress(192, 168)) == "192.168");
00311     REQUIRE(str(MAVAddress(32, 128)) == "32.128");
00312 }

```

16.229 test_mavlink.cpp File Reference

```
#include <catch.hpp>
```

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



Functions

- [TEST_CASE](#) ("message_info' returns the message info structure.", "[mavlink]")

16.229.1 Function Documentation

16.229.1.1 TEST_CASE()

```
TEST_CASE (
    "'message_info' returns the message info structure." ,
    "" [mavlink] )
```

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

16.230 test_mavlink.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
```

```

00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <catch.hpp>
00019
00020 #include "mavlink.hpp"
00021
00022
00023 TEST_CASE("'"message_info' returns the message info structure.", "[mavlink]")
00024 {
00025     SECTION("When given a numeric message ID.")
00026     {
00027         REQUIRE(mavlink::name(0) == "HEARTBEAT");
00028         REQUIRE(mavlink::name(4) == "PING");
00029         REQUIRE(mavlink::name(11) == "SET_MODE");
00030         REQUIRE(mavlink::name(41) == "MISSION_SET_CURRENT");
00031         REQUIRE(mavlink::name(131) == "ENCAPSULATED_DATA");
00032         REQUIRE(mavlink::name(321) == "PARAM_EXT_REQUEST_LIST");
00033     }
00034     SECTION("Throws and error when given an invalid message ID.")
00035     {
00036         // Currently #255 and #5000 are not valid message ID's.
00037         REQUIRE_THROWS_AS(mavlink::name(255), std::invalid_argument);
00038         REQUIRE_THROWS_WITH(mavlink::name(255), "Invalid packet ID (#255).");
00039         REQUIRE_THROWS_AS(mavlink::name(5000), std::invalid_argument);
00040         REQUIRE_THROWS_WITH(mavlink::name(5000), "Invalid packet ID (#5000).");
00041     }
00042     SECTION("When given a message name.")
00043     {
00044         REQUIRE(mavlink::id("HEARTBEAT") == 0);
00045         REQUIRE(mavlink::id("PING") == 4);
00046         REQUIRE(mavlink::id("SET_MODE") == 11);
00047         REQUIRE(mavlink::id("MISSION_SET_CURRENT") == 41);
00048         REQUIRE(mavlink::id("ENCAPSULATED_DATA") == 131);
00049         REQUIRE(mavlink::id("PARAM_EXT_REQUEST_LIST") == 321);
00050     }
00051     SECTION("Throws and error when given an invalid message name.")
00052     {
00053         REQUIRE_THROWS_AS(
00054             mavlink::id("CRAZY_MESSAGE_ID"), std::invalid_argument);
00055         REQUIRE_THROWS_WITH(
00056             mavlink::id("CRAZY_MESSAGE_ID"),
00057             "Invalid packet name (\\"CRAZY_MESSAGE_ID\\").");
00058     }
00059 }

```

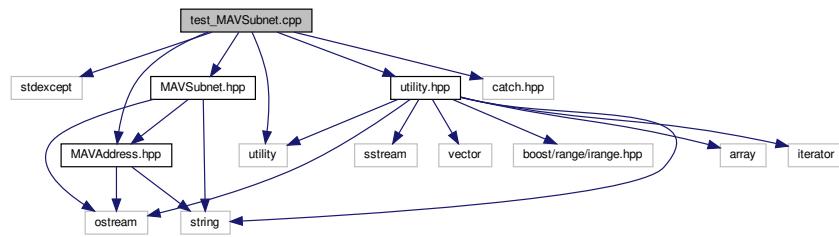
16.231 test_MAVSubnet.cpp File Reference

```

#include <stdexcept>
#include <utility>
#include <catch.hpp>
#include "MAVAddress.hpp"
#include "MAVSubnet.hpp"
#include "utility.hpp"

```

Include dependency graph for test_MAVSubnet.cpp:



Functions

- `TEST_CASE ("MAVSubnet's are comparable.", "[MAVSubnet]")`
- `TEST_CASE ("MAVSubnet's can be constructed from a MAVLink address " "and a numeric mask.", "[MAVSubnet]")`
- `TEST_CASE ("MAVSubnet's can be constructed from a MAVLink address, " "system mask, and component mask.", "[MAVSubnet]")`
- `TEST_CASE ("MAVSubnet's can be constructed from strings.", "[MAVSubnet]")`
- `TEST_CASE ("MAVSubnet's are copyable.", "[MAVSubnet]")`
- `TEST_CASE ("MAVSubnet's are movable.", "[MAVSubnet]")`
- `TEST_CASE ("MAVSubnet's are assignable.", "[MAVSubnet]")`
- `TEST_CASE ("MAVSubnet's are printable.", "[MAVSubnet]")`
- `TEST_CASE ("The 'contains' method determines if a MAVLink address " "is in the subnet.", "[MAVSubnet]")`

Variables

- `MAVSubnet subnet_b (MAVAddress(0xFFFF), 255, 255)`
- `subnet_a = std::move(subnet_b)`

16.231.1 Function Documentation

16.231.1.1 TEST_CASE() [1/9]

```
TEST_CASE (
    "MAVSubnet's are comparable." ,
    "" [MAVSubnet] )
```

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

16.231.1.2 TEST_CASE() [2/9]

```
TEST_CASE (
    "MAVSubnet's can be constructed from a MAVLink address " "and a numeric mask." ,
    "" [MAVSubnet] )
```

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

16.231.1.3 TEST_CASE() [3/9]

```
TEST_CASE (
    "MAVSubnet's can be constructed from a MAVLink address,
     " "system mask,
     and component mask." ,
    "" [MAVSubnet] )
```

Definition at line 96 of file [test_MAVSubnet.cpp](#).

16.231.1.4 TEST_CASE() [4/9]

```
TEST_CASE (
    "MAVSubnet's can be constructed from strings." ,
    "" [MAVSubnet] )
```

Definition at line 143 of file [test_MAVSubnet.cpp](#).

16.231.1.5 TEST_CASE() [5/9]

```
TEST_CASE (
    "MAVSubnet's are copyable." ,
    "" [MAVSubnet] )
```

Definition at line 285 of file [test_MAVSubnet.cpp](#).

References [subnet_a](#), and [subnet_b](#).

16.231.1.6 TEST_CASE() [6/9]

```
TEST_CASE (
    "MAVSubnet's are movable." ,
    "" [MAVSubnet] )
```

Definition at line 298 of file [test_MAVSubnet.cpp](#).

References [subnet_a](#), and [subnet_b](#).

16.231.1.7 TEST_CASE() [7/9]

```
TEST_CASE (
    "MAVSubnet's are assignable." ,
    "" [MAVSubnet] )
```

Definition at line 309 of file [test_MAVSubnet.cpp](#).

References [subnet_a](#), and [subnet_b](#).

16.231.1.8 TEST_CASE() [8/9]

```
TEST_CASE (
    "MAVSubnet's are printable." ,
    "" [MAVSubnet] )
```

Definition at line 329 of file [test_MAVSubnet.cpp](#).

16.231.1.9 TEST_CASE() [9/9]

```
TEST_CASE (
    "The 'contains' method determines if a MAVLink address " "is in the subnet." ,
    "" [MAVSubnet] )
```

Definition at line 362 of file [test_MAVSubnet.cpp](#).

16.231.2 Variable Documentation

16.231.2.1 subnet_a

```
subnet_a = std::move(subnet_b)
```

Definition at line 324 of file [test_MAVSubnet.cpp](#).

16.231.2.2 subnet_b

```
MAVSubnet subnet_b(MAVAddress(0xFFFF), 255, 255)
```

Initial value:

```
{
    MAVSubnet subnet_a(MAVAddress(0x0000), 0, 0)
```

16.232 test_MAVSubnet.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <stdexcept>
00019 #include <utility>
00020
00021 #include <catch.hpp>
00022
00023 #include "MAVAddress.hpp"
00024 #include "MAVSubnet.hpp"
00025 #include "utility.hpp"
00026
00027
00028 TEST_CASE("MAVSubnet's are comparable.", "[MAVSubnet]")
00029 {
00030     SECTION("with ==")
00031     {
00032         REQUIRE(MAVSubnet(MAVAddress(0x0000), 0x0000) ==
00033                 MAVSubnet(MAVAddress(0x0000), 0x0000));
00034         REQUIRE(MAVSubnet(MAVAddress(0x1234), 0x5678) ==
00035                 MAVSubnet(MAVAddress(0x1234), 0x5678));
00036         REQUIRE(MAVSubnet(MAVAddress(0xFFFF), 0xFFFF) ==
00037                 MAVSubnet(MAVAddress(0xFFFF), 0xFFFF));
00038         REQUIRE_FALSE(MAVSubnet(MAVAddress(0x0000), 0x0000) ==
00039                     MAVSubnet(MAVAddress(0x0001), 0x0000));
00040         REQUIRE_FALSE(MAVSubnet(MAVAddress(0x0000), 0x0000) ==
00041                     MAVSubnet(MAVAddress(0x0000), 0x0001));
00042         REQUIRE_FALSE(MAVSubnet(MAVAddress(0x1234), 0x5678) ==
00043                     MAVSubnet(MAVAddress(0x1235), 0x5678));
00044         REQUIRE_FALSE(MAVSubnet(MAVAddress(0x1234), 0x5678) ==
00045                     MAVSubnet(MAVAddress(0x1234), 0x5679));
00046         REQUIRE_FALSE(MAVSubnet(MAVAddress(0xFFFF), 0xFFFF) ==
```

```

00047             MAVSubnet(MAVAddress(0xFFFF), 0xFFFF));
00048     REQUIRE_FALSE(MAVSubnet(MAVAddress(0xFFFF), 0xFFFF) ==
00049                     MAVSubnet(MAVAddress(0xFFFF), 0xFFE));
00050 }
00051 SECTION("with !=")
00052 {
00053     REQUIRE(MAVSubnet(MAVAddress(0x0000), 0x0000) !=
00054             MAVSubnet(MAVAddress(0x0001), 0x0000));
00055     REQUIRE(MAVSubnet(MAVAddress(0x0000), 0x0000) !=
00056             MAVSubnet(MAVAddress(0x0000), 0x0001));
00057     REQUIRE(MAVSubnet(MAVAddress(0x1234), 0x5678) !=
00058             MAVSubnet(MAVAddress(0x1235), 0x5678));
00059     REQUIRE(MAVSubnet(MAVAddress(0x1234), 0x5678) !=
00060             MAVSubnet(MAVAddress(0x1234), 0x5679));
00061     REQUIRE(MAVSubnet(MAVAddress(0xFFFF), 0xFFFF) !=
00062             MAVSubnet(MAVAddress(0xFFFF), 0xFFFF));
00063     REQUIRE(MAVSubnet(MAVAddress(0xFFFF), 0xFFFF) !=
00064             MAVSubnet(MAVAddress(0xFFFF), 0xFFE));
00065     REQUIRE_FALSE(MAVSubnet(MAVAddress(0x0000), 0x0000) !=
00066                     MAVSubnet(MAVAddress(0x0000), 0x0000));
00067     REQUIRE_FALSE(MAVSubnet(MAVAddress(0x1234), 0x5678) !=
00068                     MAVSubnet(MAVAddress(0x1234), 0x5678));
00069     REQUIRE_FALSE(MAVSubnet(MAVAddress(0xFFFF), 0xFFFF) !=
00070                     MAVSubnet(MAVAddress(0xFFFF), 0xFFFF));
00071 }
00072 }
00073
00074
00075 TEST_CASE("MAVSubnet's can be constructed from a MAVLink address "
00076             "and a numeric mask.", "[MAVSubnet]")
00077 {
00078     REQUIRE(MAVSubnet(MAVAddress(0x0000), 0x0000) ==
00079             MAVSubnet(MAVAddress(0x0000), 0x0000));
00080     REQUIRE(MAVSubnet(MAVAddress(0x1234), 0x5678) ==
00081             MAVSubnet(MAVAddress(0x1234), 0x5678));
00082     REQUIRE(MAVSubnet(MAVAddress(0xFFFF), 0xFFFF) ==
00083             MAVSubnet(MAVAddress(0xFFFF), 0xFFFF));
00084 SECTION("And ensures the mask is within range (0x0000 - 0xFFFF).")
00085 {
00086     REQUIRE_THROWS_AS(
00087         MAVSubnet(MAVAddress(0x0000),
00088                 static_cast<unsigned int>(0x0000 - 1)),
00089         std::out_of_range);
00090     REQUIRE_THROWS_AS(
00091         MAVSubnet(MAVAddress(0x0000), 0xFFFF + 1), std::out_of_range);
00092 }
00093 }
00094
00095
00096 TEST_CASE("MAVSubnet's can be constructed from a MAVLink address, "
00097             "system mask, and component mask.", "[MAVSubnet]")
00098 {
00099     REQUIRE(MAVSubnet(MAVAddress(0x0000), 0, 0) ==
00100             MAVSubnet(MAVAddress(0x0000), 0));
00101     REQUIRE(MAVSubnet(MAVAddress(0x0000), 255, 0) ==
00102             MAVSubnet(MAVAddress(0x0000), 0xFF00));
00103     REQUIRE(MAVSubnet(MAVAddress(0x0000), 0, 255) ==
00104             MAVSubnet(MAVAddress(0x0000), 0x00FF));
00105     REQUIRE(MAVSubnet(MAVAddress(0xFFFF), 255, 255) ==
00106             MAVSubnet(MAVAddress(0xFFFF), 0xFFFF));
00107     REQUIRE(MAVSubnet(MAVAddress(0x1234), 64, 128) ==
00108             MAVSubnet(MAVAddress(0x1234), 0x4080));
00109     REQUIRE(MAVSubnet(MAVAddress(0x1234), 128, 64) ==
00110             MAVSubnet(MAVAddress(0x1234), 0x8040));
00111 SECTION("And ensures the system and component masks are within range "
00112             "(0x00 - 0xFF).")
00113 {
00114     auto addr = MAVAddress(0x0000);
00115     // Errors
00116     REQUIRE_THROWS_AS(
00117         MAVSubnet(addr, static_cast<unsigned int>(-1), 0),
00118         std::out_of_range);
00119     REQUIRE_THROWS_AS(
00120         MAVSubnet(addr, 0, static_cast<unsigned int>(-1)),
00121         std::out_of_range);
00122     REQUIRE_THROWS_AS(
00123         MAVSubnet(addr,
00124                 static_cast<unsigned int>(-1),
00125                 static_cast<unsigned int>(-1)),
00126         std::out_of_range);
00127     REQUIRE_THROWS_AS(MAVSubnet(addr, 256, 255), std::out_of_range);

```

```

00128     REQUIRE_THROWS_AS(MAVSubnet(addr, 255, 256), std::out_of_range);
00129     REQUIRE_THROWS_AS(MAVSubnet(addr, 256, 256), std::out_of_range);
00130     // Error messages.
00131     REQUIRE_THROWS_WITH(
00132         MAVSubnet(addr, 256, 255),
00133         "System mask (0x100) is outside of the allowed range "
00134         "(0x00 - 0xFF).");
00135     REQUIRE_THROWS_WITH(
00136         MAVSubnet(addr, 255, 256),
00137         "Component mask (0x100) is outside of the allowed range "
00138         "(0x00 - 0xFF).");
00139 }
00140 }
00141
00142
00143 TEST_CASE("MAVSubnet's can be constructed from strings.", "[MAVSubnet]")
00144 {
00145     auto addr = MAVAddress(255, 16);
00146     SECTION("Using no mask (exact address).")
00147     {
00148         REQUIRE(MAVSubnet("255.16") == MAVSubnet(addr, 255, 255));
00149     }
00150     SECTION("Using long notation.")
00151     {
00152         REQUIRE(MAVSubnet("255.16:123.234") == MAVSubnet(addr, 123, 234));
00153         REQUIRE(MAVSubnet("255.16:0.0") == MAVSubnet(addr, 0, 0));
00154         REQUIRE(MAVSubnet("255.16:128.0") == MAVSubnet(addr, 128, 0));
00155         REQUIRE(MAVSubnet("255.16:0.240") == MAVSubnet(addr, 0, 240));
00156         REQUIRE(MAVSubnet("255.16:128.240") == MAVSubnet(addr, 128, 240));
00157     }
00158     SECTION("Using forward slash notation.")
00159     {
00160         REQUIRE(MAVSubnet("255.16/0") == MAVSubnet(addr, 0b0000000000000000));
00161         REQUIRE(MAVSubnet("255.16/1") == MAVSubnet(addr, 0b1000000000000000));
00162         REQUIRE(MAVSubnet("255.16/2") == MAVSubnet(addr, 0b1100000000000000));
00163         REQUIRE(MAVSubnet("255.16/3") == MAVSubnet(addr, 0b1110000000000000));
00164         REQUIRE(MAVSubnet("255.16/4") == MAVSubnet(addr, 0b1111000000000000));
00165         REQUIRE(MAVSubnet("255.16/5") == MAVSubnet(addr, 0b1111100000000000));
00166         REQUIRE(MAVSubnet("255.16/6") == MAVSubnet(addr, 0b1111100000000000));
00167         REQUIRE(MAVSubnet("255.16/7") == MAVSubnet(addr, 0b1111110000000000));
00168         REQUIRE(MAVSubnet("255.16/8") == MAVSubnet(addr, 0b1111111000000000));
00169         REQUIRE(MAVSubnet("255.16/9") == MAVSubnet(addr, 0b1111111100000000));
00170         REQUIRE(MAVSubnet("255.16/10") == MAVSubnet(addr, 0b1111111110000000));
00171         REQUIRE(MAVSubnet("255.16/11") == MAVSubnet(addr, 0b1111111111000000));
00172         REQUIRE(MAVSubnet("255.16/12") == MAVSubnet(addr, 0b1111111111100000));
00173         REQUIRE(MAVSubnet("255.16/13") == MAVSubnet(addr, 0b1111111111110000));
00174         REQUIRE(MAVSubnet("255.16/14") == MAVSubnet(addr, 0b1111111111111000));
00175         REQUIRE(MAVSubnet("255.16/15") == MAVSubnet(addr, 0b1111111111111100));
00176         REQUIRE(MAVSubnet("255.16/16") == MAVSubnet(addr, 0b1111111111111111));
00177     }
00178     SECTION("Using backslash notation.")
00179     {
00180         REQUIRE(MAVSubnet("255.16\\0") == MAVSubnet(addr, 0b0000000000000000));
00181         REQUIRE(MAVSubnet("255.16\\1") == MAVSubnet(addr, 0b0000000010000000));
00182         REQUIRE(MAVSubnet("255.16\\2") == MAVSubnet(addr, 0b0000000001100000));
00183         REQUIRE(MAVSubnet("255.16\\3") == MAVSubnet(addr, 0b0000000001110000));
00184         REQUIRE(MAVSubnet("255.16\\4") == MAVSubnet(addr, 0b0000000001111000));
00185         REQUIRE(MAVSubnet("255.16\\5") == MAVSubnet(addr, 0b0000000001111100));
00186         REQUIRE(MAVSubnet("255.16\\6") == MAVSubnet(addr, 0b0000000001111110));
00187         REQUIRE(MAVSubnet("255.16\\7") == MAVSubnet(addr, 0b00000000011111110));
00188         REQUIRE(MAVSubnet("255.16\\8") == MAVSubnet(addr, 0b00000000011111111));
00189     }
00190     SECTION("And ensures the subnet string is valid.")
00191     {
00192         // Errors
00193         REQUIRE_THROWS_AS(MAVSubnet("255.16 255.256"), std::invalid_argument);
00194         REQUIRE_THROWS_AS(MAVSubnet("255.16-256.255"), std::invalid_argument);
00195         REQUIRE_THROWS_AS(MAVSubnet("255.16+256.255"), std::invalid_argument);
00196         REQUIRE_THROWS_AS(MAVSubnet("255.16:1"), std::invalid_argument);
00197         REQUIRE_THROWS_AS(MAVSubnet("255.16:1."), std::invalid_argument);
00198         REQUIRE_THROWS_AS(MAVSubnet("255.16:1.2."), std::invalid_argument);
00199         REQUIRE_THROWS_AS(MAVSubnet("255.16:1.2.3"), std::invalid_argument);
00200         REQUIRE_THROWS_AS(MAVSubnet("255.16:a.2.3"), std::invalid_argument);
00201         REQUIRE_THROWS_AS(MAVSubnet("255.16:1.b.3"), std::invalid_argument);
00202         REQUIRE_THROWS_AS(MAVSubnet("255.16:1.c.3"), std::invalid_argument);
00203         REQUIRE_THROWS_AS(MAVSubnet("255.16:+1.0"), std::invalid_argument);
00204         REQUIRE_THROWS_AS(MAVSubnet("255.16:0.+1"), std::invalid_argument);
00205         REQUIRE_THROWS_AS(MAVSubnet("255.16:-1.0"), std::invalid_argument);
00206         REQUIRE_THROWS_AS(MAVSubnet("255.16:0.-1"), std::invalid_argument);
00207         // Error messages.
00208         REQUIRE_THROWS_WITH(

```

```

00209     MAVSubnet("255.16 255.256"),
00210     "Invalid MAVLink subnet: \"255.16 255.256\".");
00211 REQUIRE_THROWS_WITH(
00212     MAVSubnet("255.16-256.255"),
00213     "Invalid MAVLink subnet: \"255.16-256.255\".");
00214 REQUIRE_THROWS_WITH(
00215     MAVSubnet("255.16+256.255"),
00216     "Invalid MAVLink subnet: \"255.16+256.255\".");
00217 REQUIRE_THROWS_WITH(
00218     MAVSubnet("255.16:1"), "Invalid MAVLink subnet: \"255.16:1\".");
00219 REQUIRE_THROWS_WITH(
00220     MAVSubnet("255.16:1."), "Invalid MAVLink subnet: \"255.16:1.\\".");
00221 REQUIRE_THROWS_WITH(
00222     MAVSubnet("255.16:1.2"),
00223     "Invalid MAVLink subnet: \"255.16:1.2.\".");
00224 REQUIRE_THROWS_WITH(
00225     MAVSubnet("255.16:1.2.3"),
00226     "Invalid MAVLink subnet: \"255.16:1.2.3\".");
00227 REQUIRE_THROWS_WITH(
00228     MAVSubnet("255.16:a.2.3"),
00229     "Invalid MAVLink subnet: \"255.16:a.2.3\".");
00230 REQUIRE_THROWS_WITH(
00231     MAVSubnet("255.16:1.b.3"),
00232     "Invalid MAVLink subnet: \"255.16:1.b.3\".");
00233 REQUIRE_THROWS_WITH(
00234     MAVSubnet("255.16:1.2.c"),
00235     "Invalid MAVLink subnet: \"255.16:1.2.c\".");
00236 REQUIRE_THROWS_WITH(
00237     MAVSubnet("255.16:+1.0"),
00238     "Invalid MAVLink subnet: \"255.16:+1.0\".");
00239 REQUIRE_THROWS_WITH(
00240     MAVSubnet("255.16:0.+1"),
00241     "Invalid MAVLink subnet: \"255.16:0.+1\".");
00242 REQUIRE_THROWS_WITH(
00243     MAVSubnet("255.16:-1.0"),
00244     "Invalid MAVLink subnet: \"255.16:-1.0\".");
00245 REQUIRE_THROWS_WITH(
00246     MAVSubnet("255.16:0.-1"),
00247     "Invalid MAVLink subnet: \"255.16:0.-1\".");
00248 }
00249 SECTION("And ensures mask is within range.")
00250 {
00251     // Errors
00252     REQUIRE_THROWS_AS(MAVSubnet("255.16:256.255"), std::out_of_range);
00253     REQUIRE_THROWS_AS(MAVSubnet("255.16:255.256"), std::out_of_range);
00254     // Error messages.
00255     REQUIRE_THROWS_WITH(
00256         MAVSubnet("255.16:256.255"),
00257         "System ID (256) is outside of the allowed range (0 - 255).");
00258     REQUIRE_THROWS_WITH(
00259         MAVSubnet("255.16:255.256"),
00260         "Component ID (256) is outside of the allowed range (0 - 255).");
00261 }
00262 SECTION("And ensures forward slash mask is within range.")
00263 {
00264     // Errors
00265     REQUIRE_THROWS_AS(MAVSubnet("255.16/-1"), std::out_of_range);
00266     REQUIRE_THROWS_AS(MAVSubnet("255.16/17"), std::out_of_range);
00267     // Error messages.
00268     REQUIRE_THROWS_WITH(
00269         MAVSubnet("255.16/17"),
00270         "Forward slash mask (17) is outside of allowed range (0 - 16).");
00271 }
00272 SECTION("And ensures backslash mask is within range.")
00273 {
00274     // Errors
00275     REQUIRE_THROWS_AS(MAVSubnet("255.16/-1"), std::out_of_range);
00276     REQUIRE_THROWS_AS(MAVSubnet("255.16\|-1"), std::out_of_range);
00277     // Error messages.
00278     REQUIRE_THROWS_WITH(
00279         MAVSubnet("255.16\|9"),
00280         "Backslash mask (9) is outside of allowed range (0 - 8).");
00281 }
00282 }
00283
00284
00285 TEST_CASE("MAVSubnet's are copyable.", "[MAVSubnet]")
00286 {
00287     MAVSubnet subnet_a(MAVAddress(0x0000), 0, 0);
00288     MAVSubnet subnet_b(MAVAddress(0xFFFF), 255, 255);
00289     MAVSubnet subnet_a_copy = subnet_a;

```

```

00290     MAVSubnet subnet_b_copy(subnet_b);
00291     REQUIRE(&subnet_a != &subnet_a_copy);
00292     REQUIRE(subnet_a == subnet_a_copy);
00293     REQUIRE(&subnet_b != &subnet_b_copy);
00294     REQUIRE(subnet_b == subnet_b_copy);
00295 }
00296
00297
00298 TEST_CASE("MAVSubnet's are movable.", "[MAVSubnet]")
00299 {
00300     MAVSubnet subnet_a(MAVAddress(0x0000), 0, 0);
00301     MAVSubnet subnet_b(MAVAddress(0xFFFF), 255, 255);
00302     MAVSubnet subnet_a_moved = std::move(subnet_a);
00303     MAVSubnet subnet_b_moved(std::move(subnet_b));
00304     REQUIRE(subnet_a_moved == MAVSubnet(MAVAddress(0x0000), 0, 0));
00305     REQUIRE(subnet_b_moved == MAVSubnet(MAVAddress(0xFFFF), 255, 255));
00306 }
00307
00308
00309 TEST_CASE("MAVSubnet's are assignable.", "[MAVSubnet]")
00310 {
00311     MAVSubnet subnet_a(MAVAddress(0x0000), 0, 0);
00312     MAVSubnet subnet_b(MAVAddress(0xFFFF), 255, 255);
00313     REQUIRE(subnet_a == MAVSubnet(MAVAddress(0x0000), 0, 0));
00314     subnet_a = subnet_b;
00315     REQUIRE(subnet_a == MAVSubnet(MAVAddress(0xFFFF), 255, 255));
00316 }
00317
00318
00319 TEST_CASE("MAVSubnet's are assignable (by move semantics).", "[MAVSubnet]")
00320 {
00321     MAVSubnet subnet_a(MAVAddress(0x0000), 0, 0);
00322     MAVSubnet subnet_b(MAVAddress(0xFFFF), 255, 255);
00323     REQUIRE(subnet_a == MAVSubnet(MAVAddress(0x0000), 0, 0));
00324     subnet_a = std::move(subnet_b);
00325     REQUIRE(subnet_a == MAVSubnet(MAVAddress(0xFFFF), 255, 255));
00326 }
00327
00328
00329 TEST_CASE("MAVSubnet's are printable.", "[MAVSubnet]")
00330 {
00331     auto addr = MAVAddress(255, 16);
00332     REQUIRE(str(MAVSubnet(addr, 123, 234)) == "255.16:123.234");
00333     REQUIRE(str(MAVSubnet(addr, 128, 240)) == "255.16:128.240");
00334     REQUIRE(str(MAVSubnet(addr, 0b0000000000000000)) == "255.16/0");
00335     REQUIRE(str(MAVSubnet(addr, 0b1000000000000000)) == "255.16/1");
00336     REQUIRE(str(MAVSubnet(addr, 0b1100000000000000)) == "255.16/2");
00337     REQUIRE(str(MAVSubnet(addr, 0b1110000000000000)) == "255.16/3");
00338     REQUIRE(str(MAVSubnet(addr, 0b1111000000000000)) == "255.16/4");
00339     REQUIRE(str(MAVSubnet(addr, 0b1111100000000000)) == "255.16/5");
00340     REQUIRE(str(MAVSubnet(addr, 0b1111110000000000)) == "255.16/6");
00341     REQUIRE(str(MAVSubnet(addr, 0b1111111000000000)) == "255.16/7");
00342     REQUIRE(str(MAVSubnet(addr, 0b1111111100000000)) == "255.16/8");
00343     REQUIRE(str(MAVSubnet(addr, 0b1111111110000000)) == "255.16/9");
00344     REQUIRE(str(MAVSubnet(addr, 0b1111111111000000)) == "255.16/10");
00345     REQUIRE(str(MAVSubnet(addr, 0b1111111111100000)) == "255.16/11");
00346     REQUIRE(str(MAVSubnet(addr, 0b1111111111110000)) == "255.16/12");
00347     REQUIRE(str(MAVSubnet(addr, 0b1111111111111000)) == "255.16/13");
00348     REQUIRE(str(MAVSubnet(addr, 0b1111111111111100)) == "255.16/14");
00349     REQUIRE(str(MAVSubnet(addr, 0b1111111111111110)) == "255.16/15");
00350     REQUIRE(str(MAVSubnet(addr, 0b1111111111111111)) == "255.16");
00351     REQUIRE(str(MAVSubnet(addr, 0b000000010000000)) == "255.16\1");
00352     REQUIRE(str(MAVSubnet(addr, 0b000000011000000)) == "255.16\2");
00353     REQUIRE(str(MAVSubnet(addr, 0b000000011100000)) == "255.16\3");
00354     REQUIRE(str(MAVSubnet(addr, 0b000000011110000)) == "255.16\4");
00355     REQUIRE(str(MAVSubnet(addr, 0b000000011111000)) == "255.16\5");
00356     REQUIRE(str(MAVSubnet(addr, 0b000000011111100)) == "255.16\6");
00357     REQUIRE(str(MAVSubnet(addr, 0b0000000111111100)) == "255.16\7");
00358     REQUIRE(str(MAVSubnet(addr, 0b000000011111111)) == "255.16\8");
00359 }
00360
00361
00362 TEST_CASE("The 'contains' method determines if a MAVLink address "
00363         "is in the subnet.",
00364         "[MAVSubnet]")
00365 {
00366     REQUIRE(MAVSubnet("0.0:0.0").contains(MAVAddress("0.0")));
00367     REQUIRE(MAVSubnet("0.0:0.0").contains(MAVAddress("255.255")));
00368     REQUIRE(MAVSubnet("0.0:255.255").contains(MAVAddress("0.0")));
00369     REQUIRE_FALSE(MAVSubnet("0.0:255.255").contains(MAVAddress("1.1")));
00370     REQUIRE_FALSE(MAVSubnet("0.0:255.255").contains(MAVAddress("255.255")));

```

```

00371     SECTION("With subnet 192.0/14")
00372     {
00373         MAVSubnet subnet("192.0/14");
00374         REQUIRE(subnet.contains(MAVAddress("192.0")));
00375         REQUIRE(subnet.contains(MAVAddress("192.1")));
00376         REQUIRE(subnet.contains(MAVAddress("192.2")));
00377         REQUIRE(subnet.contains(MAVAddress("192.3")));
00378         REQUIRE_FALSE(subnet.contains(MAVAddress("192.4")));
00379         REQUIRE_FALSE(subnet.contains(MAVAddress("192.5")));
00380         REQUIRE_FALSE(subnet.contains(MAVAddress("192.255")));
00381         REQUIRE_FALSE(subnet.contains(MAVAddress("191.0")));
00382         REQUIRE_FALSE(subnet.contains(MAVAddress("193.1")));
00383         REQUIRE_FALSE(subnet.contains(MAVAddress("0.2")));
00384         REQUIRE_FALSE(subnet.contains(MAVAddress("255.3")));
00385     }
00386     SECTION("With subnet 192.0\6")
00387     {
00388         MAVSubnet subnet("192.0\6");
00389         REQUIRE(subnet.contains(MAVAddress("192.0")));
00390         REQUIRE(subnet.contains(MAVAddress("192.1")));
00391         REQUIRE(subnet.contains(MAVAddress("192.2")));
00392         REQUIRE(subnet.contains(MAVAddress("192.3")));
00393         REQUIRE_FALSE(subnet.contains(MAVAddress("192.4")));
00394         REQUIRE_FALSE(subnet.contains(MAVAddress("192.5")));
00395         REQUIRE_FALSE(subnet.contains(MAVAddress("192.255")));
00396         REQUIRE(subnet.contains(MAVAddress("191.0")));
00397         REQUIRE(subnet.contains(MAVAddress("193.1")));
00398         REQUIRE(subnet.contains(MAVAddress("0.2")));
00399         REQUIRE(subnet.contains(MAVAddress("255.3")));
00400     }
00401 }

```

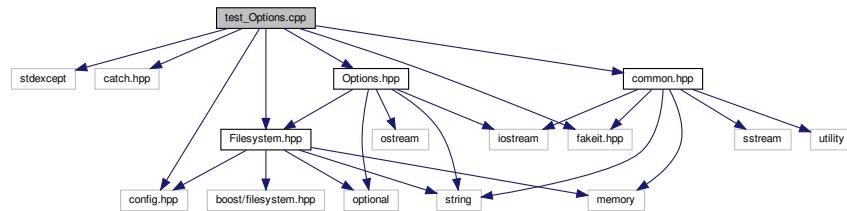
16.233 test_Options.cpp File Reference

```

#include <stdexcept>
#include "catch.hpp"
#include "fakeit.hpp"
#include "config.hpp"
#include "Filesystem.hpp"
#include "Options.hpp"
#include "common.hpp"

```

Include dependency graph for test_Options.cpp:



Functions

- **TEST_CASE** ("find_config' returns the path to the highest priority " "configuration file.", "[[Options](#)]"")
- **TEST_CASE** ("Options's class prints the help message.", "[[Options](#)]"")
- **TEST_CASE** ("Options's class prints version information when given the " "--version' flag.", "[[Options](#)]"")
- **TEST_CASE** ("Options's class will use the given configuration file " "(--config flag).", "[[Options](#)]"")
- **TEST_CASE** ("Options's class finds the configuration file.", "[[Options](#)]"")
- **TEST_CASE** ("Options's class sets run to false and ast to true when the --ast " "flag is given.", "[[Options](#)]"")
- **TEST_CASE** ("Option's class has a loglevel option", "[[Options](#)]"")

16.233.1 Function Documentation

16.233.1.1 TEST_CASE() [1/7]

```
TEST_CASE (
    "'find_config' returns the path to the highest priority configuration file." ,
    "" [Options] )
```

Definition at line 30 of file [test_Options.cpp](#).

16.233.1.2 TEST_CASE() [2/7]

```
TEST_CASE (
    "Options's class prints the help message." ,
    "" [Options] )
```

Definition at line 133 of file [test_Options.cpp](#).

16.233.1.3 TEST_CASE() [3/7]

```
TEST_CASE (
    "Options's class prints version information when given the '--version' flag." ,
    "" [Options] )
```

Definition at line 162 of file [test_Options.cpp](#).

16.233.1.4 TEST_CASE() [4/7]

```
TEST_CASE (
    "Options's class will use the given configuration file \"(--config flag).\" ,
    "" [Options] )
```

Definition at line 184 of file [test_Options.cpp](#).

16.233.1.5 TEST_CASE() [5/7]

```
TEST_CASE (
    "Options's class finds the configuration file." ,
    "" [Options] )
```

Definition at line 245 of file [test_Options.cpp](#).

16.233.1.6 TEST_CASE() [6/7]

```
TEST_CASE (
    "Options's class sets run to false and ast to true when the --ast \"flag is given."
,
    "" [Options] )
```

Definition at line 403 of file [test_Options.cpp](#).

16.233.1.7 TEST_CASE() [7/7]

```
TEST_CASE (
    "Option's class has a loglevel option" ,
    "" [Options] )
```

Definition at line 427 of file [test_Options.cpp](#).

16.234 test_Options.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <stdexcept>
00019
00020 #include "catch.hpp"
00021 #include "fakeit.hpp"
00022
00023 #include "config.hpp"
00024 #include "Filesystem.hpp"
00025 #include "Options.hpp"
```

```

00026
00027 #include "common.hpp"
00028
00029
00030 TEST_CASE("'find_config' returns the path to the highest priority "
00031     "configuration file.", "[Options]")
00032 {
00033     std::vector<Filesystem::path> paths;
00034     fakeit::Mock<Filesystem> fs_mock;
00035     SECTION("First found is given by MAVTABLES_CONFIG_PATH environment "
00036         "variable.")
00037     {
00038         fakeit::When(Method(fs_mock, exists)).AlwaysDo([&](auto p)
00039         {
00040             paths.push_back(p);
00041             return p == Filesystem::path("mtbls.conf");
00042         });
00043         setenv("MAVTABLES_CONFIG_PATH", "mtbls.conf", true);
00044         auto config_file = find_config(fs_mock.get());
00045         REQUIRE(config_file.has_value());
00046         REQUIRE(config_file.value() == "mtbls.conf");
00047         unsetenv("MAVTABLES_CONFIG_PATH");
00048         fakeit::Verify(Method(fs_mock, exists)).Exactly(1);
00049         REQUIRE(paths.size() == 1);
00050         REQUIRE(paths[0] == Filesystem::path("mtbls.conf"));
00051     }
00052     SECTION("First found is .mavtablesrc in current directory.")
00053     {
00054         fakeit::When(Method(fs_mock, exists)).AlwaysDo([&](auto p)
00055         {
00056             paths.push_back(p);
00057             return p == Filesystem::path(".mavtablesrc");
00058         });
00059         setenv("MAVTABLES_CONFIG_PATH", "mtbls.conf", true);
00060         auto config_file = find_config(fs_mock.get());
00061         REQUIRE(config_file.has_value());
00062         REQUIRE(config_file.value() == ".mavtablesrc");
00063         unsetenv("MAVTABLES_CONFIG_PATH");
00064         fakeit::Verify(Method(fs_mock, exists)).Exactly(2);
00065         REQUIRE(paths.size() == 2);
00066         REQUIRE(paths[0] == Filesystem::path("mtbls.conf"));
00067         REQUIRE(paths[1] == Filesystem::path(".mavtablesrc"));
00068     }
00069     SECTION("First found is .mavtablesrc in HOME directory.")
00070     {
00071         Filesystem::path home_path(std::getenv("HOME"));
00072         home_path /= Filesystem::path(".mavtablesrc");
00073         fakeit::When(Method(fs_mock, exists)).AlwaysDo([&](auto p)
00074         {
00075             paths.push_back(p);
00076             return p == home_path;
00077         });
00078         setenv("MAVTABLES_CONFIG_PATH", "mtbls.conf", true);
00079         auto config_file = find_config(fs_mock.get());
00080         REQUIRE(config_file.has_value());
00081         REQUIRE(config_file.value() == home_path);
00082         unsetenv("MAVTABLES_CONFIG_PATH");
00083         fakeit::Verify(Method(fs_mock, exists)).Exactly(3);
00084         REQUIRE(paths.size() == 3);
00085         REQUIRE(paths[0] == Filesystem::path("mtbls.conf"));
00086         REQUIRE(paths[1] == Filesystem::path(".mavtablesrc"));
00087         REQUIRE(paths[2] == Filesystem::path(home_path));
00088     }
00089     SECTION("First found is PREFIX/etc/mavtable.conf.")
00090     {
00091         Filesystem::path home_path(std::getenv("HOME"));
00092         home_path /= Filesystem::path(".mavtablesrc");
00093         fakeit::When(Method(fs_mock, exists)).AlwaysDo([&](auto p)
00094         {
00095             paths.push_back(p);
00096             return p == (PREFIX "/etc/mavtable.conf");
00097         });
00098         setenv("MAVTABLES_CONFIG_PATH", "mtbls.conf", true);
00099         auto config_file = find_config(fs_mock.get());
00100         REQUIRE(config_file.has_value());
00101         REQUIRE(config_file.value() == (PREFIX "/etc/mavtable.conf"));
00102         unsetenv("MAVTABLES_CONFIG_PATH");
00103         fakeit::Verify(Method(fs_mock, exists)).Exactly(4);
00104         REQUIRE(paths.size() == 4);
00105         REQUIRE(paths[0] == Filesystem::path("mtbls.conf"));
00106         REQUIRE(paths[1] == Filesystem::path(".mavtablesrc"));

```

```

00107     REQUIRE(paths[2] == Filesystem::path(home_path));
00108     REQUIRE(paths[3] == Filesystem::path(PREFIX "/etc/mavtables.conf"));
00109 }
00110 SECTION("Failed to find any configuration file.")
00111 {
00112     Filesystem::path home_path(std::getenv("HOME"));
00113     home_path /= Filesystem::path(".mavtablesrc");
00114     fakeit::When(Method(fs_mock, exists)).AlwaysDo([&](auto p)
00115     {
00116         paths.push_back(p);
00117         return false;
00118     });
00119     setenv("MAVTABLES_CONFIG_PATH", "mtbls.conf", true);
00120     auto config_file = find_config(fs_mock.get());
00121     REQUIRE_FALSE(config_file.has_value());
00122     unsetenv("MAVTABLES_CONFIG_PATH");
00123     fakeit::Verify(Method(fs_mock, exists)).Exactly(4);
00124     REQUIRE(paths.size() == 4);
00125     REQUIRE(paths[0] == Filesystem::path("mtbls.conf"));
00126     REQUIRE(paths[1] == Filesystem::path("./mavtablesrc"));
00127     REQUIRE(paths[2] == Filesystem::path(home_path));
00128     REQUIRE(paths[3] == Filesystem::path(PREFIX "/etc/mavtables.conf"));
00129 }
00130 }
00131
00132
00133 TEST_CASE("Options's class prints the help message.", "[Options]")
00134 {
00135     MockCout mock_cout;
00136     std::string help_message =
00137         "usage: <program name here>:\n"
00138         " -h [ --help ]           print this message\n"
00139         " --config arg            specify configuration file\n"
00140         " --ast                   print AST of configuration file (do not run)\n"
00141         " --version               print version and license information\n"
00142         " --loglevel arg          level of logging, between 0 and 3\n";
00143 SECTION("When given the '-h' flag.")
00144 {
00145     int argc = 2;
00146     const char *argv[2] = {"<program name here>", "-h"};
00147     Options options(argc, argv);
00148     REQUIRE(mock_cout.buffer() == help_message);
00149     REQUIRE_FALSE(options);
00150 }
00151 SECTION("When given the '--help' flag.")
00152 {
00153     int argc = 2;
00154     const char *argv[2] = {"<program name here>", "--help"};
00155     Options options(argc, argv);
00156     REQUIRE(mock_cout.buffer() == help_message);
00157     REQUIRE_FALSE(options);
00158 }
00159 }
00160
00161
00162 TEST_CASE("Options's class prints version information when given the "
00163         "'--version' flag.", "[Options]")
00164 {
00165     MockCout mock_cout;
00166     int argc = 2;
00167     const char *argv[2] = {"<program name here>", "--version"};
00168     Options options(argc, argv);
00169     REQUIRE(
00170         mock_cout.buffer() ==
00171         "mavtables (SHAMU Project) v" + std::to_string(VERSION_MAJOR) +
00172         "." + std::to_string(VERSION_MINOR) +
00173         "." + std::to_string(VERSION_PATCH) +
00174         "\nCopyright (C) 2018 Michael R. Shannon\n"
00175         "\n"
00176         "License: GPL v2.0 or any later version.\n"
00177         "This is free software; see the source for copying conditions. "
00178         "There is NO WARRANTY; not even for MERCHANTABILITY or FITNESS FOR A "
00179         "PARTICULAR PURPOSE.\n";
00180     REQUIRE_FALSE(options);
00181 }
00182
00183
00184 TEST_CASE("Options's class will use the given configuration file "
00185         "(--config flag).", "[Options]")
00186 {
00187     MockCout mock_cout;

```

```

00188     std::vector<Filesystem::path> paths;
00189     fakeit::Mock<Filesystem> fs_mock;
00190     SECTION("File found.")
00191     {
00192         // Setup mocks.
00193         fakeit::When(Method(fs_mock, exists)).AlwaysDo([&] (auto p)
00194         {
00195             paths.push_back(p);
00196             return p == Filesystem::path("examples/test.conf");
00197         });
00198         // Construct Options object.
00199         int argc = 3;
00200         const char *argv[3] = {"mavtables", "--config", "examples/test.conf"};
00201         Options options(argc, argv, fs_mock.get());
00202         // Verify Options object.
00203         REQUIRE(options.config_file() == "examples/test.conf");
00204         REQUIRE(options);
00205         REQUIRE_FALSE(options.ast());
00206         REQUIRE(options.run());
00207         // Verify printing.
00208         REQUIRE(mock_cout.buffer().empty());
00209         // Verify exists calls.
00210         fakeit::Verify(Method(fs_mock, exists)).Exactly(1);
00211         REQUIRE(paths.size() == 1);
00212         REQUIRE(paths[0] == Filesystem::path("examples/test.conf"));
00213     }
00214     SECTION("File not found (throws error).")
00215     {
00216         // Setup mocks.
00217         fakeit::When(Method(fs_mock, exists)).AlwaysDo([&] (auto p)
00218         {
00219             paths.push_back(p);
00220             return false;
00221         });
00222         // Construct Options object.
00223         int argc = 3;
00224         const char *argv[3] =
00225         {
00226             "mavtables", "--config", "non_existant_file.conf"
00227         };
00228         // Verify Options object.
00229         REQUIRE_THROWS_AS(
00230             Options(argc, argv, fs_mock.get()), std::runtime_error);
00231         REQUIRE_THROWS_WITH(
00232             Options(argc, argv, fs_mock.get()),
00233             "mavtables could not locate a configuration file");
00234         // Verify printing.
00235         REQUIRE(mock_cout.buffer().empty());
00236         // Verify exists calls.
00237         fakeit::Verify(Method(fs_mock, exists)).Exactly(2);
00238         REQUIRE(paths.size() == 2);
00239         REQUIRE(paths[0] == Filesystem::path("non_existant_file.conf"));
00240         REQUIRE(paths[1] == Filesystem::path("non_existant_file.conf"));
00241     }
00242 }
00243
00244
00245 TEST_CASE("Options's class finds the configuration file.", "[Options]")
00246 {
00247     MockCOut mock_cout;
00248     std::vector<Filesystem::path> paths;
00249     fakeit::Mock<Filesystem> fs_mock;
00250     SECTION("First found is given by MAVTABLES_CONFIG_PATH environment "
00251             "variable.")
00252     {
00253         // Setup mocks.
00254         fakeit::When(Method(fs_mock, exists)).AlwaysDo([&] (auto p)
00255         {
00256             paths.push_back(p);
00257             return p == Filesystem::path("mtbls.conf");
00258         });
00259         setenv("MAVTABLES_CONFIG_PATH", "mtbls.conf", true);
00260         // Construct Options object.
00261         int argc = 1;
00262         const char *argv[2] = {"mavtables"};
00263         Options options(argc, argv, fs_mock.get());
00264         unsetenv("MAVTABLES_CONFIG_PATH");
00265         // Verify Options object.
00266         REQUIRE(options.config_file() == "mtbls.conf");
00267         REQUIRE(options);
00268         REQUIRE_FALSE(options.ast());

```

```

00269     REQUIRE(options.run());
00270     // Verify printing.
00271     REQUIRE(mock_cout.buffer().empty());
00272     // Verify exists calls.
00273     fakeit::Verify(Method(fs_mock, exists)).Exactly(1);
00274     REQUIRE(paths.size() == 1);
00275     REQUIRE(paths[0] == Filesystem::path("mtbls.conf"));
00276 }
00277 SECTION("First found is .mavtablesrc in current directory.")
00278 {
00279     // Setup mocks.
00280     fakeit::When(Method(fs_mock, exists)).AlwaysDo([&](auto p)
00281     {
00282         paths.push_back(p);
00283         return p == Filesystem::path(".mavtablesrc");
00284     });
00285     setenv("MAVTABLES_CONFIG_PATH", "mtbls.conf", true);
00286     // Construct Options object.
00287     int argc = 1;
00288     const char *argv[2] = {"mavtables"};
00289     Options options(argc, argv, fs_mock.get());
00290     unsetenv("MAVTABLES_CONFIG_PATH");
00291     // Verify Options object.
00292     REQUIRE(options.config_file() == ".mavtablesrc");
00293     REQUIRE(options);
00294     REQUIRE_FALSE(options.ast());
00295     REQUIRE(options.run());
00296     // Verify printing.
00297     REQUIRE(mock_cout.buffer().empty());
00298     // Verify exists calls.
00299     fakeit::Verify(Method(fs_mock, exists)).Exactly(2);
00300     REQUIRE(paths.size() == 2);
00301     REQUIRE(paths[0] == Filesystem::path("mtbls.conf"));
00302     REQUIRE(paths[1] == Filesystem::path(".mavtablesrc"));
00303 }
00304 SECTION("First found is .mavtablesrc in HOME directory.")
00305 {
00306     // Setup mocks.
00307     Filesystem::path home_path(std::getenv("HOME"));
00308     home_path /= Filesystem::path(".mavtablesrc");
00309     fakeit::When(Method(fs_mock, exists)).AlwaysDo([&](auto p)
00310     {
00311         paths.push_back(p);
00312         return p == home_path;
00313     });
00314     setenv("MAVTABLES_CONFIG_PATH", "mtbls.conf", true);
00315     // Construct Options object.
00316     int argc = 1;
00317     const char *argv[2] = {"mavtables"};
00318     Options options(argc, argv, fs_mock.get());
00319     unsetenv("MAVTABLES_CONFIG_PATH");
00320     // Verify Options object.
00321     REQUIRE(options.config_file() == home_path);
00322     REQUIRE(options);
00323     REQUIRE_FALSE(options.ast());
00324     REQUIRE(options.run());
00325     // Verify printing.
00326     REQUIRE(mock_cout.buffer().empty());
00327     // Verify exists calls.
00328     fakeit::Verify(Method(fs_mock, exists)).Exactly(3);
00329     REQUIRE(paths.size() == 3);
00330     REQUIRE(paths[0] == Filesystem::path("mtbls.conf"));
00331     REQUIRE(paths[1] == Filesystem::path(".mavtablesrc"));
00332     REQUIRE(paths[2] == Filesystem::path(home_path));
00333 }
00334 SECTION("First found is PREFIX/etc/mavtables.conf.")
00335 {
00336     // Setup mocks.
00337     Filesystem::path home_path(std::getenv("HOME"));
00338     home_path /= Filesystem::path(".mavtablesrc");
00339     fakeit::When(Method(fs_mock, exists)).AlwaysDo([&](auto p)
00340     {
00341         paths.push_back(p);
00342         return p == (PREFIX "/etc/mavtables.conf");
00343     });
00344     setenv("MAVTABLES_CONFIG_PATH", "mtbls.conf", true);
00345     // Construct Options object.
00346     int argc = 1;
00347     const char *argv[2] = {"mavtables"};
00348     Options options(argc, argv, fs_mock.get());
00349     unsetenv("MAVTABLES_CONFIG_PATH");

```

```

00350     // Verify Options object.
00351     REQUIRE(options.config_file() == (PREFIX "/etc/mavtables.conf"));
00352     REQUIRE(options);
00353     REQUIRE_FALSE(options.ast());
00354     REQUIRE(options.run());
00355     // Verify printing.
00356     REQUIRE(mock_cout.buffer().empty());
00357     // Verify exists calls.
00358     fakeit::Verify(Method(fs_mock, exists)).Exactly(4);
00359     REQUIRE(paths.size() == 4);
00360     REQUIRE(paths[0] == Filesystem::path("mtbls.conf"));
00361     REQUIRE(paths[1] == Filesystem::path(".mavtablesrc"));
00362     REQUIRE(paths[2] == Filesystem::path(home_path));
00363     REQUIRE(paths[3] == Filesystem::path(PREFIX "/etc/mavtables.conf"));
00364 }
00365 SECTION("Failed to find any configuration file.")
00366 {
00367     // Setup mocks.
00368     Filesystem::path home_path(std::getenv("HOME"));
00369     home_path /= Filesystem::path(".mavtablesrc");
00370     fakeit::When(Method(fs_mock, exists)).AlwaysDo([&](auto p)
00371     {
00372         paths.push_back(p);
00373         return false;
00374     });
00375     setenv("MAVTABLES_CONFIG_PATH", "mtbls.conf", true);
00376     // Construct Options object.
00377     int argc = 1;
00378     const char *argv[2] = {"mavtables"};
00379     REQUIRE_THROWS_AS(
00380         Options(argc, argv, fs_mock.get()),
00381         std::runtime_error);
00382     REQUIRE_THROWS_WITH(
00383         Options(argc, argv, fs_mock.get()),
00384         "mavtables could not locate a configuration file");
00385     unsetenv("MAVTABLES_CONFIG_PATH");
00386     // Verify printing.
00387     REQUIRE(mock_cout.buffer().empty());
00388     // Verify exists calls.
00389     fakeit::Verify(Method(fs_mock, exists)).Exactly(8);
00390     REQUIRE(paths.size() == 8);
00391     REQUIRE(paths[0] == Filesystem::path("mtbls.conf"));
00392     REQUIRE(paths[1] == Filesystem::path(".mavtablesrc"));
00393     REQUIRE(paths[2] == Filesystem::path(home_path));
00394     REQUIRE(paths[3] == Filesystem::path(PREFIX "/etc/mavtables.conf"));
00395     REQUIRE(paths[4] == Filesystem::path("mtbls.conf"));
00396     REQUIRE(paths[5] == Filesystem::path(".mavtablesrc"));
00397     REQUIRE(paths[6] == Filesystem::path(home_path));
00398     REQUIRE(paths[7] == Filesystem::path(PREFIX "/etc/mavtables.conf"));
00399 }
00400 }
00401
00402
00403 TEST_CASE("Options's class sets run to false and ast to true when the --ast "
00404             "flag is given.", "[Options]")
00405 {
00406     MockCOut mock_cout;
00407     // Setup mocks.
00408     fakeit::Mock<Filesystem> fs_mock;
00409     fakeit::When(Method(fs_mock, exists)).AlwaysReturn(true);
00410     // Construct Options object.
00411     int argc = 4;
00412     const char *argv[4] =
00413     {
00414         "mavtables", "--ast", "--config", "test/mavtables.conf"
00415     };
00416     Options options(argc, argv);
00417     // Verify Options object.
00418     REQUIRE(options.config_file() == "test/mavtables.conf");
00419     REQUIRE(options);
00420     REQUIRE(options.ast());
00421     REQUIRE_FALSE(options.run());
00422     // Verify printing.
00423     REQUIRE(mock_cout.buffer().empty());
00424 }
00425
00426
00427 TEST_CASE("Option's class has a loglevel option", "[Options]")
00428 {
00429     MockCOut mock_cout;
00430     // Setup mocks.

```

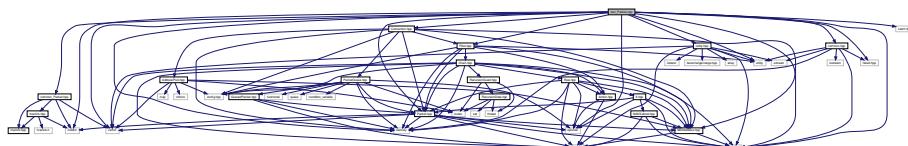
```

00431     fakeit::Mock<Filesystem> fs_mock;
00432     fakeit::When(Method(fs_mock, exists)).AlwaysReturn(true);
00433     SECTION("defaults to loglevel 0")
00434     {
00435         // Construct Options object.
00436         int argc = 3;
00437         const char *argv[3] = {"mavtables", "--config", "test/mavtables.conf"};
00438         Options options(argc, argv);
00439         // Verify Options object.
00440         REQUIRE(options.loglevel() == 0);
00441         // Verify printing.
00442         REQUIRE(mock_cout.buffer().empty());
00443     }
00444     SECTION("sets the loglevel when the --loglevel is given")
00445     {
00446         // Construct Options object.
00447         int argc = 5;
00448         const char *argv[5] =
00449         {
00450             "mavtables", "--loglevel", "3", "--config", "test/mavtables.conf"
00451         };
00452         Options options(argc, argv);
00453         // Verify Options object.
00454         REQUIRE(options.loglevel() == 3);
00455         // Verify printing.
00456         REQUIRE(mock_cout.buffer().empty());
00457     }
00458 }
```

16.235 test_Packet.cpp File Reference

```

#include <cstdint>
#include <memory>
#include <optional>
#include <string>
#include <utility>
#include <vector>
#include <catch.hpp>
#include <fakeit.hpp>
#include "Connection.hpp"
#include "Filter.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "utility.hpp"
#include "common.hpp"
#include "common_Packet.hpp"
Include dependency graph for test_Packet.cpp:
```



Functions

- [TEST_CASE \("Packet's can be constructed.", "\[Packet\]"\)](#)
- [TEST_CASE \("Packet's are comparable.", "\[Packet\]"\)](#)

- [TEST_CASE \("Packet's are copyable.", "\[Packet\]"\)](#)
- [TEST_CASE \("Packet's are movable.", "\[Packet\]"\)](#)
- [TEST_CASE \("Packet's are assignable.", "\[Packet\]"\)](#)
- [PacketTestClass packet_b \({0, 7, 7, 3, 4}\)](#)
- [TEST_CASE \("Packet's contain raw packet data and make it accessible.", "\[Packet\]"\)](#)
- [TEST_CASE \("Packet's have a version.", "\[Packet\]"\)](#)
- [TEST_CASE \("Packet's have an ID.", "\[Packet\]"\)](#)
- [TEST_CASE \("Packet's have a name.", "\[Packet\]"\)](#)
- [TEST_CASE \("Packet's have a source address.", "\[Packet\]"\)](#)
- [TEST_CASE \("Packet's optionally have a destination address.", "\[Packet\]"\)](#)
- [TEST_CASE \("Packet's optionally have a source connection.", "\[Packet\]"\)](#)
- [TEST_CASE \("Packet's are printable.", "\[Packet\]"\)](#)

Variables

- [packet_a = std::move\(packet_b\)](#)

16.235.1 Function Documentation

16.235.1.1 packet_b()

```
PacketTestClass packet_b (
    {0, 7, 7, 3, 4} )
```

Here is the caller graph for this function:



16.235.1.2 TEST_CASE() [1/13]

```
TEST_CASE (
    "Packet's can be constructed." ,
    "" [Packet] )
```

Definition at line 90 of file [test_Packet.cpp](#).

16.235.1.3 TEST_CASE() [2/13]

```
TEST_CASE (
    "Packet's are comparable." ,
    "" [Packet] )
```

Definition at line 99 of file [test_Packet.cpp](#).

16.235.1.4 TEST_CASE() [3/13]

```
TEST_CASE (
    "Packet's are copyable." ,
    "" [Packet] )
```

Definition at line 117 of file [test_Packet.cpp](#).

16.235.1.5 TEST_CASE() [4/13]

```
TEST_CASE (
    "Packet's are movable." ,
    "" [Packet] )
```

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

16.235.1.6 TEST_CASE() [5/13]

```
TEST_CASE (
    "Packet's are assignable." ,
    "" [Packet] )
```

Definition at line 133 of file [test_Packet.cpp](#).

References [packet_a](#), and [packet_b\(\)](#).

Here is the call graph for this function:



16.235.1.7 TEST_CASE() [6/13]

```
TEST_CASE (
    "Packet's contain raw packet data and make it accessible." ,
    "" [Packet] )
```

Definition at line 153 of file [test_Packet.cpp](#).

16.235.1.8 TEST_CASE() [7/13]

```
TEST_CASE (
    "Packet's have a version." ,
    "" [Packet] )
```

Definition at line 165 of file [test_Packet.cpp](#).

16.235.1.9 TEST_CASE() [8/13]

```
TEST_CASE (
    "Packet's have an ID." ,
    "" [Packet] )
```

Definition at line 171 of file [test_Packet.cpp](#).

16.235.1.10 TEST_CASE() [9/13]

```
TEST_CASE (
    "Packet's have a name." ,
    "" [Packet] )
```

Definition at line 177 of file [test_Packet.cpp](#).

16.235.1.11 TEST_CASE() [10/13]

```
TEST_CASE (
    "Packet's have a source address." ,
    "" [Packet] )
```

Definition at line 183 of file [test_Packet.cpp](#).

16.235.1.12 TEST_CASE() [11/13]

```
TEST_CASE (
    "Packet's optionally have a destination address." ,
    "" [Packet] )
```

Definition at line 189 of file [test_Packet.cpp](#).

16.235.1.13 TEST_CASE() [12/13]

```
TEST_CASE (
    "Packet's optionally have a source connection." ,
    "" [Packet] )
```

Definition at line 195 of file [test_Packet.cpp](#).

16.235.1.14 TEST_CASE() [13/13]

```
TEST_CASE (
    "Packet's are printable." ,
    "" [Packet] )
```

Definition at line 215 of file [test_Packet.cpp](#).

16.235.2 Variable Documentation**16.235.2.1 packet_a**

```
packet_a = std::move(packet_b)
```

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

16.236 test_Packet.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <cstdint>
00019 #include <memory>
00020 #include <optional>
00021 #include <string>
00022 #include <utility>
00023 #include <vector>
00024
00025 #include <catch.hpp>
00026 #include <fakeit.hpp>
00027
00028 #include "Connection.hpp"
00029 #include "Filter.hpp"
00030 #include "MAVAddress.hpp"
00031 #include "Packet.hpp"
00032 #include "utility.hpp"
00033
00034 #include "common.hpp"
00035 #include "common_Packet.hpp"
00036
00037
00038 namespace
00039 {
00040
00041 #ifdef __clang__
00042     #pragma clang diagnostic push
00043     #pragma clang diagnostic ignored "-Wweak-vtables"
00044 #endif
00045
00046     // Subclass of Packet used for testing the abstract class Packet.
00047     class PacketTestClass : public Packet
00048     {
00049         public:
00050             PacketTestClass(const PacketTestClass &other) = default;
00051             PacketTestClass(PacketTestClass &&other) = default;
00052             PacketTestClass(std::vector<uint8_t> data)
00053                 : Packet(std::move(data))
00054             {
00055             }
00056             // LCOV_EXCL_START
00057             ~PacketTestClass() = default;
00058             // LCOV_EXCL_STOP
00059             virtual ::Packet::Version version() const
00060             {
00061                 return ::Packet::V1;
00062             }
00063             virtual unsigned long id() const
00064             {
00065                 return 42;
00066             }
00067             virtual std::string name() const
00068             {
00069                 return "MISSION_CURRENT";
00070             }
00071             virtual MAVAddress source() const
00072             {
00073                 return MAVAddress("3.14");
00074             }
00075             virtual std::optional<MAVAddress> dest() const
00076             {
00077                 return MAVAddress("2.71");
```

```
00078         }
00079         PacketTestClass &operator=(const PacketTestClass &other) = default;
00080         PacketTestClass &operator=(PacketTestClass &&other) = default;
00081     };
00082
00083 #ifdef __clang__
00084     #pragma clang diagnostic pop
00085 #endif
00086
00087 }
00088
00089
00090 TEST_CASE("Packet's can be constructed.", "[Packet]")
00091 {
00092     std::vector<uint8_t> data = {0, 7, 7, 3, 4};
00093     REQUIRE_NOTHROW(PacketTestClass({}));
00094     REQUIRE_NOTHROW(PacketTestClass({1, 3, 3, 7}));
00095     REQUIRE_NOTHROW(PacketTestClass(data));
00096 }
00097
00098
00099 TEST_CASE("Packet's are comparable.", "[Packet]")
00100 {
00101     SECTION("with ==")
00102     {
00103         REQUIRE(PacketTestClass({1, 3, 3, 7}) == PacketTestClass({1, 3, 3, 7}));
00104         REQUIRE_FALSE(
00105             PacketTestClass({1, 3, 3, 7}) == PacketTestClass({0, 7, 7, 3, 4}));
00106     }
00107     SECTION("with !=")
00108     {
00109         REQUIRE(
00110             PacketTestClass({1, 3, 3, 7}) != PacketTestClass({0, 7, 7, 3, 4}));
00111         REQUIRE_FALSE(
00112             PacketTestClass({1, 3, 3, 7}) != PacketTestClass({1, 3, 3, 7}));
00113     }
00114 }
00115
00116
00117 TEST_CASE("Packet's are copyable.", "[Packet]")
00118 {
00119     PacketTestClass original({1, 3, 3, 7});
00120     PacketTestClass copy(original);
00121     REQUIRE(copy == PacketTestClass({1, 3, 3, 7}));
00122 }
00123
00124
00125 TEST_CASE("Packet's are movable.", "[Packet]")
00126 {
00127     PacketTestClass original({1, 3, 3, 7});
00128     PacketTestClass moved(std::move(original));
00129     REQUIRE(moved == PacketTestClass({1, 3, 3, 7}));
00130 }
00131
00132
00133 TEST_CASE("Packet's are assignable.", "[Packet]")
00134 {
00135     PacketTestClass packet_a({1, 3, 3, 7});
00136     PacketTestClass packet_b({0, 7, 7, 3, 4});
00137     REQUIRE(packet_a == PacketTestClass({1, 3, 3, 7}));
00138     packet_a = packet_b;
00139     REQUIRE(packet_a == PacketTestClass({0, 7, 7, 3, 4}));
00140 }
00141
00142
00143 TEST_CASE("Packet's are assignable (by move semantics).", "[Packet]")
00144 {
00145     PacketTestClass packet_a({1, 3, 3, 7});
00146     PacketTestClass packet_b({0, 7, 7, 3, 4});
00147     REQUIRE(packet_a == PacketTestClass({1, 3, 3, 7}));
00148     packet_a = std::move(packet_b);
00149     REQUIRE(packet_a == PacketTestClass({0, 7, 7, 3, 4}));
00150 }
00151
00152
00153 TEST_CASE("Packet's contain raw packet data and make it accessible.",
00154         "[Packet]")
00155 {
00156     REQUIRE(
00157         PacketTestClass({1, 3, 3, 7}).data() ==
00158             std::vector<uint8_t>({1, 3, 3, 7}));
```

```

00159     REQUIRE_FALSE(
00160         PacketTestClass({1, 3, 3, 7}).data() ==
00161             std::vector<uint8_t>({1, 0, 5, 3}));
00162 }
00163
00164
00165 TEST_CASE("Packet's have a version.", "[Packet]")
00166 {
00167     REQUIRE(PacketTestClass({}).version() == Packet::V1);
00168 }
00169
00170
00171 TEST_CASE("Packet's have an ID.", "[Packet]")
00172 {
00173     REQUIRE(PacketTestClass({}).id() == 42);
00174 }
00175
00176
00177 TEST_CASE("Packet's have a name.", "[Packet]")
00178 {
00179     REQUIRE(PacketTestClass({}).name() == "MISSION_CURRENT");
00180 }
00181
00182
00183 TEST_CASE("Packet's have a source address.", "[Packet]")
00184 {
00185     REQUIRE(PacketTestClass({}).source() == MAVAddress("3.14"));
00186 }
00187
00188
00189 TEST_CASE("Packet's optionally have a destination address.", "[Packet]")
00190 {
00191     REQUIRE(PacketTestClass({}).dest().value() == MAVAddress("2.71"));
00192 }
00193
00194
00195 TEST_CASE("Packet's optionally have a source connection.", "[Packet]")
00196 {
00197     SECTION("Defaults to nullptr.")
00198     {
00199         REQUIRE(PacketTestClass({}).connection() == nullptr);
00200     }
00201     SECTION("Can be set with the 'connection' method.")
00202     {
00203         fakeit::Mock<Filter> mock_filter;
00204         auto filter = mock_shared(mock_filter);
00205         auto conn = std::make_shared<Connection>("SOURCE", filter);
00206         PacketTestClass packet({});
00207         packet.connection(conn);
00208         REQUIRE(packet.connection() != nullptr);
00209         REQUIRE(packet.connection() == conn);
00210         REQUIRE(str(*packet.connection()) == "SOURCE");
00211     }
00212 }
00213
00214
00215 TEST_CASE("Packet's are printable.", "[Packet]")
00216 {
00217     REQUIRE(
00218         str(PacketTestClass({})) ==
00219         "MISSION_CURRENT (#42) from 3.14 to 2.71 (v1.0)");
00220 }

```

16.237 test_PacketParser.cpp File Reference

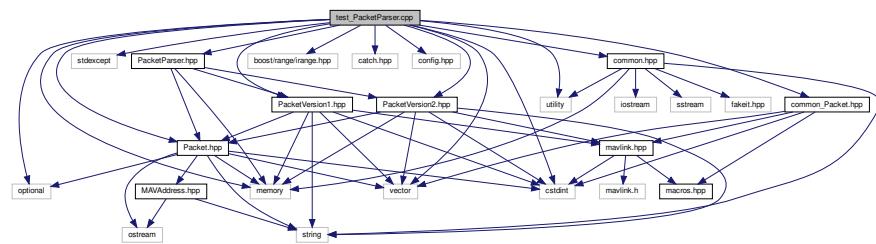
```

#include <cstdint>
#include <memory>
#include <optional>
#include <stdexcept>
#include <utility>
#include <vector>
#include <boost/range/irange.hpp>

```

```
#include <catch.hpp>
#include "config.hpp"
#include "Packet.hpp"
#include "PacketParser.hpp"
#include "PacketVersion1.hpp"
#include "PacketVersion2.hpp"
#include "common.hpp"
#include "common_Packet.hpp"

Include dependency graph for test_PacketParser.cpp:
```



Functions

- [TEST_CASE \("PacketParser's can be constructed.", "\[PacketParser\]"\)](#)
- [TEST_CASE \("PacketParser's can parse packets with 'parse_byte'.", "\[PacketParser\]"\)](#)
- [TEST_CASE \("PacketParser's can be cleared with 'clear'.", "\[PacketParser\]"\)](#)
- [TEST_CASE \("PacketParser's keep track of how many bytes they have parsed of " "the current packet.", "\[PacketParser\]"\)](#)

16.237.1 Function Documentation

16.237.1.1 TEST_CASE() [1/4]

```
TEST_CASE (
    "PacketParser's can be constructed." ,
    "" [PacketParser] )
```

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

16.237.1.2 TEST_CASE() [2/4]

```
TEST_CASE (
    "PacketParser's can parse packets with 'parse_byte'." ,
    "" [PacketParser] )
```

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

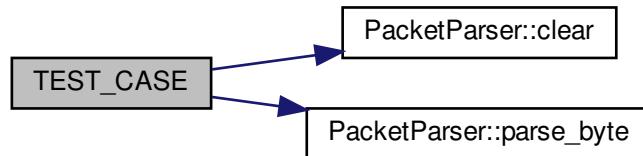
16.237.1.3 TEST_CASE() [3/4]

```
TEST_CASE (
    "PacketParser's can be cleared with 'clear'." ,
    "" [PacketParser] )
```

Definition at line 193 of file [test_PacketParser.cpp](#).

References [PacketParser::clear\(\)](#), and [PacketParser::parse_byte\(\)](#).

Here is the call graph for this function:



16.237.1.4 TEST_CASE() [4/4]

```
TEST_CASE (
    "PacketParser's keep track of how many bytes they have parsed of " "the current packet."
    ,
    "" [PacketParser] )
```

Definition at line 206 of file [test_PacketParser.cpp](#).

16.238 test_PacketParser.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
```

```
00016
00017
00018 #include <cstdint>
00019 #include <memory>
00020 #include <optional>
00021 #include <stdexcept>
00022 #include <utility>
00023 #include <vector>
00024
00025 #include <boost/range/irange.hpp>
00026 #include <catch.hpp>
00027
00028 #include "config.hpp"
00029 #include "Packet.hpp"
00030 #include "PacketParser.hpp"
00031 #include "PacketVersion1.hpp"
00032 #include "PacketVersion2.hpp"
00033
00034 #include "common.hpp"
00035 #include "common_Packet.hpp"
00036
00037
00038 namespace
00039 {
00040
00041     // Forward defines of local functions.
00042     std::unique_ptr<Packet> test_packet_parser(PacketParser &parser,
00043         const std::vector<uint8_t> &data, size_t packet_end);
00044     void add_bytes(std::vector<uint8_t> &data, size_t num_bytes);
00045
00046
00047     /** Test a PacketParser instance on data.
00048     *
00049     * Uses the REQUIRE macro from Catch C++ internally for testing.
00050     *
00051     * \param parser The instance of the packet parser to use.
00052     * \param data The data to give to the parser.
00053     * \param packet_end The byte (1 indexed) that the packet is supposed to
00054     *     end on.
00055     *
00056     * \returns The packet parsed from the given \p data.
00057     */
00058     std::unique_ptr<Packet> test_packet_parser(PacketParser &parser,
00059         const std::vector<uint8_t> &data, size_t packet_end)
00060     {
00061         std::unique_ptr<Packet> packet;
00062
00063         for (auto i : boost::irange(static_cast<size_t>(0), packet_end - 1))
00064         {
00065             REQUIRE(parser.parse_byte(data[i]) == nullptr);
00066         }
00067
00068         REQUIRE((packet = parser.parse_byte(data[packet_end - 1])) != nullptr);
00069
00070         for (auto i : boost::irange(packet_end, data.size()))
00071         {
00072             REQUIRE(parser.parse_byte(data[i]) == nullptr);
00073         }
00074
00075         return packet;
00076     }
00077
00078
00079     /** Add bytes to either side of a vector.
00080     *
00081     * Used for testing.
00082     *
00083     * \param data A reference to the vector to modify.
00084     * \param num_bytes Number of bytes to add on either end the \p data
00085     *     vector.
00086     */
00087     void add_bytes(std::vector<uint8_t> &data, size_t num_bytes)
00088     {
00089         for (auto i : boost::irange(static_cast<size_t>(0), num_bytes))
00090         {
00091             data.insert(data.begin(), static_cast<uint8_t>(i));
00092         }
00093
00094         for (auto i : boost::irange(static_cast<size_t>(0), num_bytes))
00095         {
00096             data.push_back(static_cast<uint8_t>(i));
```

```

00097         }
00098     }
00099
00100 }
00101
00102
00103 TEST_CASE("PacketParser's can be constructed.", "[PacketParser]")
00104 {
00105     REQUIRE_NO_THROW(PacketParser());
00106 }
00107
00108
00109 TEST_CASE("PacketParser's can parse packets with 'parse_byte'.",
00110         "[PacketParser]")
00111 {
00112     PacketParser parser;
00113     SECTION("Can parse v1.0 packets.")
00114     {
00115         auto data = to_vector(PingV1());
00116         add_bytes(data, 3);
00117         auto packet = test_packet_parser(parser, data, sizeof(PingV1) + 3);
00118         REQUIRE(*packet == packet_v1::Packet(to_vector(PingV1())));
00119     }
00120     SECTION("Can parse v2.0 packets.")
00121     {
00122         auto data = to_vector(PingV2());
00123         add_bytes(data, 3);
00124         auto packet = test_packet_parser(parser, data, sizeof(PingV2) + 3);
00125         REQUIRE(*packet == packet_v2::Packet(to_vector(PingV2())));
00126     }
00127     SECTION("Can parse v2.0 packets with signature.")
00128     {
00129         auto data = to_vector_with_sig(PingV2());
00130         add_bytes(data, 3);
00131         auto packet = test_packet_parser(
00132             parser, data,
00133             sizeof(PingV2) + packet_v2::SIGNATURE_LENGTH + 3);
00134         REQUIRE(*packet == packet_v2::Packet(to_vector_with_sig(PingV2())));
00135     }
00136     SECTION("Prints error and clears buffer if message ID is invalid.")
00137     {
00138         // std::stringstream buffer;
00139         // std::streambuf *sbuf = std::cerr.rdbuf();
00140         // std::cerr.rdbuf(buffer.rdbuf());
00141         MockCerr mock_cerr;
00142         auto data = to_vector(PingV1());
00143         data[5] = 255;
00144         std::unique_ptr<Packet> packet;
00145
00146         for (auto byte : data)
00147         {
00148             packet = parser.parse_byte(byte);
00149         }
00150
00151         REQUIRE(packet == nullptr);
00152         REQUIRE(
00153             mock_cerr.buffer() ==
00154             "Packet ID (#255) is not part of the ''"
00155             MAVLINK_DIALECT "' MAVLink dialect.\n");
00156         REQUIRE(parser.bytes_parsed() == 0);
00157         // std::cerr.rdbuf(sbuf);
00158     }
00159     SECTION("Can parse multiple packets back to back.")
00160     {
00161         auto data1 = to_vector(PingV1());
00162         auto data2 = to_vector(PingV2());
00163         auto data3 = to_vector_with_sig(PingV2());
00164         auto packet1 = test_packet_parser(parser, data1, sizeof(PingV1));
00165         auto packet2 = test_packet_parser(parser, data2, sizeof(PingV2));
00166         auto packet3 = test_packet_parser(
00167             parser, data3,
00168             sizeof(PingV2) + packet_v2::SIGNATURE_LENGTH);
00169         REQUIRE(*packet1 == packet_v1::Packet(to_vector(PingV1())));
00170         REQUIRE(*packet2 == packet_v2::Packet(to_vector(PingV2())));
00171         REQUIRE(*packet3 == packet_v2::Packet(to_vector_with_sig(PingV2())));
00172     }
00173     SECTION("Can parse multiple packets with random bytes interspersed.")
00174     {
00175         auto data1 = to_vector(PingV1());
00176         auto data2 = to_vector(PingV2());
00177         auto data3 = to_vector_with_sig(PingV2());

```

```

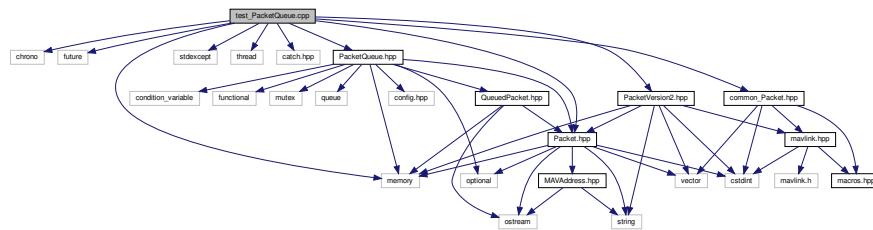
00178     add_bytes(data1, 3);
00179     add_bytes(data2, 3);
00180     add_bytes(data3, 3);
00181     auto packet1 = test_packet_parser(parser, data1, sizeof(PingV1) + 3);
00182     auto packet2 = test_packet_parser(parser, data2, sizeof(PingV2) + 3);
00183     auto packet3 = test_packet_parser(
00184         parser, data3,
00185         sizeof(PingV2) + packet_v2::SIGNATURE_LENGTH + 3);
00186     REQUIRE(*packet1 == packet_v1::Packet(to_vector(PingV1())));
00187     REQUIRE(*packet2 == packet_v2::Packet(to_vector(PingV2())));
00188     REQUIRE(*packet3 == packet_v2::Packet(to_vector_with_sig(PingV2())));
00189 }
00190 }
00191
00192
00193 TEST_CASE("PacketParser's can be cleared with 'clear'.", "[PacketParser]")
00194 {
00195     PacketParser parser;
00196     parser.parse_byte(0xFE);
00197     parser.parse_byte(14);
00198     parser.parse_byte(3);
00199     parser.clear();
00200     auto data = to_vector(PingV1());
00201     auto packet = test_packet_parser(parser, data, sizeof(PingV1));
00202     REQUIRE(*packet == packet_v1::Packet(to_vector(PingV1())));
00203 }
00204
00205
00206 TEST_CASE("PacketParser's keep track of how many bytes they have parsed of "
00207             "the current packet.", "[PacketParser]")
00208 {
00209     PacketParser parser;
00210     REQUIRE(parser.bytes_parsed() == 0);
00211     parser.parse_byte(1);
00212     REQUIRE(parser.bytes_parsed() == 0);
00213     parser.parse_byte(2);
00214     REQUIRE(parser.bytes_parsed() == 0);
00215     parser.parse_byte(3);
00216     REQUIRE(parser.bytes_parsed() == 0);
00217     parser.parse_byte(0xFD);
00218     REQUIRE(parser.bytes_parsed() == 1);
00219     parser.parse_byte(14);
00220     REQUIRE(parser.bytes_parsed() == 2);
00221     parser.parse_byte(3);
00222     REQUIRE(parser.bytes_parsed() == 3);
00223     parser.clear();
00224     REQUIRE(parser.bytes_parsed() == 0);
00225     parser.parse_byte(1);
00226     REQUIRE(parser.bytes_parsed() == 0);
00227 }
```

16.239 test_PacketQueue.cpp File Reference

```

#include <chrono>
#include <future>
#include <memory>
#include <stdexcept>
#include <thread>
#include <catch.hpp>
#include <Packet.hpp>
#include <PacketQueue.hpp>
#include "PacketVersion2.hpp"
#include "common_Packet.hpp"
```

Include dependency graph for test_PacketQueue.cpp:



Functions

- [TEST_CASE \("PacketQueue's can be constructed.", "\[AddressPool\]"\)](#)
- [TEST_CASE \("PacketQueue's 'push' adds a packet to the queue.", "\[PacketQueue\]"\)](#)
- [TEST_CASE \("PacketQueue's 'empty' method determines if the queue is empty or " "not.", "\[PacketQueue\]"\)](#)
- [TEST_CASE \("PacketQueue's can be managed with 'push' and 'pop' methods.", "\[PacketQueue\]"\)](#)
- [TEST_CASE \("PacketQueue's 'pop' method blocks by default.", "\[PacketQueue\]"\)](#)
- [TEST_CASE \("PacketQueue's 'pop' method optionally has a timeout.", "\[PacketQueue\]"\)](#)
- [TEST_CASE \("PacketQueue's 'pop' method is non blocking when given a 0 second " "timeout.", "\[PacketQueue\]"\)](#)

16.239.1 Function Documentation

16.239.1.1 TEST_CASE() [1/7]

```
TEST_CASE (
    "PacketQueue's can be constructed." ,
    "" [AddressPool] )
```

Definition at line 36 of file [test_PacketQueue.cpp](#).

16.239.1.2 TEST_CASE() [2/7]

```
TEST_CASE (
    "PacketQueue's 'push' adds a packet to the queue." ,
    "" [PacketQueue] )
```

Definition at line 51 of file [test_PacketQueue.cpp](#).

16.239.1.3 TEST_CASE() [3/7]

```
TEST_CASE (
    "PacketQueue's 'empty' method determines if the queue is empty or \"not.\" ,
    "" [PacketQueue] )
```

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

References [ping](#).

16.239.1.4 TEST_CASE() [4/7]

```
TEST_CASE (
    "PacketQueue's can be managed with 'push' and 'pop' methods." ,
    "" [PacketQueue] )
```

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

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:

**16.239.1.5 TEST_CASE()** [5/7]

```
TEST_CASE (
    "PacketQueue's 'pop' method blocks by default." ,
    "" [PacketQueue] )
```

Definition at line 205 of file [test_PacketQueue.cpp](#).

References [ping](#).

16.239.1.6 TEST_CASE() [6/7]

```
TEST_CASE (
    "PacketQueue's 'pop' method optionally has a timeout." ,
    "" [PacketQueue] )
```

Definition at line 240 of file [test_PacketQueue.cpp](#).

References [ping](#).

16.239.1.7 TEST_CASE() [7/7]

```
TEST_CASE (
    "PacketQueue's 'pop' method is non blocking when given a 0 second \"timeout.\" ,
    "" [PacketQueue] )
```

Definition at line 288 of file [test_PacketQueue.cpp](#).

References [ping](#).

16.240 test_PacketQueue.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <chrono>
00019 #include <future>
00020 #include <memory>
00021 #include <stdexcept>
00022 #include <thread>
00023
00024 #include <catch.hpp>
00025
00026 #include <Packet.hpp>
00027 #include <PacketQueue.hpp>
00028 #include "PacketVersion2.hpp"
00029
00030 #include "common_Packet.hpp"
00031
00032
00033 using namespace std::chrono_literals;
00034
00035
00036 TEST_CASE("PacketQueue's can be constructed.", "[AddressPool]")
00037 {
00038     SECTION("Without a push callback.")
```

```
00039     {
00040         REQUIRE_NOTHROW(PacketQueue());
00041     }
00042 SECTION("With a push callback.")
00043 {
00044     PacketQueue pq([]() {});
00045     auto ping = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00046     pq.push(ping);
00047 }
00048 }
00049
00050
00051 TEST_CASE("PacketQueue's 'push' adds a packet to the queue.", "[PacketQueue]")
00052 {
00053     SECTION("Ensures the packet is not null.")
00054     {
00055         PacketQueue queue;
00056         REQUIRE_THROWS_AS(queue.push(nullptr), std::invalid_argument);
00057         REQUIRE_THROWS_WITH(
00058             queue.push(nullptr), "Given packet pointer is null.");
00059     }
00060 SECTION("Calls the push callback.")
00061 {
00062     auto ping = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00063     bool called = false;
00064     PacketQueue queue([&]()
00065     {
00066         called = true;
00067     });
00068     REQUIRE_FALSE(called);
00069     queue.push(ping);
00070     REQUIRE(called);
00071 }
00072 }
00073
00074
00075 TEST_CASE("PacketQueue's 'empty' method determines if the queue is empty or "
00076             "not.", "[PacketQueue]")
00077 {
00078     auto ping = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00079     PacketQueue queue;
00080     REQUIRE(queue.empty());
00081     queue.push(ping);
00082     REQUIRE_FALSE(queue.empty());
00083 }
00084
00085
00086 TEST_CASE("PacketQueue's can be managed with 'push' and 'pop' methods.",
00087             "[PacketQueue]")
00088 {
00089     auto heartbeat =
00090         std::make_shared<packet_v2::Packet>(to_vector(HeartbeatV2()));
00091     auto ping =
00092         std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00093     auto set_mode =
00094         std::make_shared<packet_v2::Packet>(to_vector(SetModeV2()));
00095     auto mission_set_current =
00096         std::make_shared<packet_v2::Packet>(to_vector(MissionSetCurrentV2()));
00097     auto encapsulated_data =
00098         std::make_shared<packet_v2::Packet>(to_vector(EncapsulatedDataV2()));
00099     auto param_ext_request_list =
00100         std::make_shared<packet_v2::Packet>(to_vector(ParamExtRequestListV2()));
00101     PacketQueue queue;
00102 SECTION("Maintains order among the same priority")
00103 {
00104     queue.push(heartbeat);
00105     queue.push(ping, 0);
00106     queue.push(set_mode);
00107     queue.push(mission_set_current, 0);
00108     queue.push(encapsulated_data);
00109     queue.push(param_ext_request_list, 0);
00110     // HEARTBEAT
00111     auto packet = queue.pop(0s);
00112     REQUIRE(packet != nullptr);
00113     REQUIRE(*packet == *heartbeat);
00114     // PING
00115     packet = queue.pop(0s);
00116     REQUIRE(packet != nullptr);
00117     REQUIRE(*packet == *ping);
00118     // SET_MODE
00119     packet = queue.pop(0s);
```

```

00120     REQUIRE(packet != nullptr);
00121     REQUIRE(*packet == *set_mode);
00122     // MISSION_SET_CURRENT
00123     packet = queue.pop(0s);
00124     REQUIRE(packet != nullptr);
00125     REQUIRE(*packet == *mission_set_current);
00126     // ENCAPSULATED_DATA
00127     packet = queue.pop(0s);
00128     REQUIRE(packet != nullptr);
00129     REQUIRE(*packet == *encapsulated_data);
00130     // PARAM_EXT_REQUEST_LIST
00131     packet = queue.pop(0s);
00132     REQUIRE(packet != nullptr);
00133     REQUIRE(*packet == *param_ext_request_list);
00134 }
00135 SECTION("Maintains priority order.")
00136 {
00137     queue.push(heartbeat, -1);
00138     queue.push(ping, 0);
00139     queue.push(set_mode, 1);
00140     queue.push(mission_set_current, -3);
00141     queue.push(encapsulated_data, -2);
00142     queue.push(param_ext_request_list, 3);
00143     // PARAM_EXT_REQUEST_LIST
00144     auto packet = queue.pop(0s);
00145     REQUIRE(packet != nullptr);
00146     REQUIRE(*packet == *param_ext_request_list);
00147     // SET_MODE
00148     packet = queue.pop(0s);
00149     REQUIRE(packet != nullptr);
00150     REQUIRE(*packet == *set_mode);
00151     // PING
00152     packet = queue.pop(0s);
00153     REQUIRE(packet != nullptr);
00154     REQUIRE(*packet == *ping);
00155     // HEARTBEAT
00156     packet = queue.pop(0s);
00157     REQUIRE(packet != nullptr);
00158     REQUIRE(*packet == *heartbeat);
00159     // ENCAPSULATED_DATA
00160     packet = queue.pop(0s);
00161     REQUIRE(packet != nullptr);
00162     REQUIRE(*packet == *encapsulated_data);
00163     // MISSION_SET_CURRENT
00164     packet = queue.pop(0s);
00165     REQUIRE(packet != nullptr);
00166     REQUIRE(*packet == *mission_set_current);
00167 }
00168 SECTION("Maintains order among the same priority as well as "
00169         "priority order.")
00170 {
00171     queue.push(heartbeat);
00172     queue.push(ping, 2);
00173     queue.push(set_mode);
00174     queue.push(mission_set_current, 2);
00175     queue.push(encapsulated_data);
00176     queue.push(param_ext_request_list, 2);
00177     // PING
00178     auto packet = queue.pop();
00179     REQUIRE(packet != nullptr);
00180     REQUIRE(*packet == *ping);
00181     // MISSION_SET_CURRENT
00182     packet = queue.pop(0s);
00183     REQUIRE(packet != nullptr);
00184     REQUIRE(*packet == *mission_set_current);
00185     // PARAM_EXT_REQUEST_LIST
00186     packet = queue.pop(0s);
00187     REQUIRE(packet != nullptr);
00188     REQUIRE(*packet == *param_ext_request_list);
00189     // HEARTBEAT
00190     packet = queue.pop(0s);
00191     REQUIRE(packet != nullptr);
00192     REQUIRE(*packet == *heartbeat);
00193     // SET_MODE
00194     packet = queue.pop(0s);
00195     REQUIRE(packet != nullptr);
00196     REQUIRE(*packet == *set_mode);
00197     // ENCAPSULATED_DATA
00198     packet = queue.pop(0s);
00199     REQUIRE(packet != nullptr);
00200     REQUIRE(*packet == *encapsulated_data);

```

```
00201      }
00202  }
00203
00204
00205 TEST_CASE("PacketQueue's 'pop' method blocks by default.", "[PacketQueue]")
00206 {
00207     auto ping = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00208     PacketQueue queue;
00209     SECTION("And will be released when a packet becomes available.")
00210     {
00211         auto future = std::async(std::launch::async, [&]()
00212         {
00213             return queue.pop();
00214         });
00215         REQUIRE(future.wait_for(0s) != std::future_status::ready);
00216         queue.push(ping);
00217         auto result = future.get();
00218         REQUIRE(result != nullptr);
00219         REQUIRE(*result == *ping);
00220     }
00221     SECTION("And will be released when the 'close' method is called.")
00222     {
00223         auto future1 = std::async(std::launch::async, [&]()
00224         {
00225             return queue.pop();
00226         });
00227         auto future2 = std::async(std::launch::async, [&]()
00228         {
00229             return queue.pop();
00230         });
00231         REQUIRE(future1.wait_for(0s) != std::future_status::ready);
00232         REQUIRE(future2.wait_for(0s) != std::future_status::ready);
00233         queue.close();
00234         REQUIRE(future1.get() == nullptr);
00235         REQUIRE(future2.get() == nullptr);
00236     }
00237 }
00238
00239
00240 TEST_CASE("PacketQueue's 'pop' method optionally has a timeout.",
00241             "[PacketQueue]")
00242 {
00243     auto ping = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00244     PacketQueue queue;
00245     SECTION("And will be released when a packet becomes available.")
00246     {
00247         auto future = std::async(std::launch::async, [&]()
00248         {
00249             return queue.pop(10s);
00250         });
00251         auto status = future.wait_for(0s);
00252         REQUIRE(status != std::future_status::ready);
00253         queue.push(ping);
00254         auto result = future.get();
00255         REQUIRE(result != nullptr);
00256         REQUIRE(*result == *ping);
00257     }
00258     SECTION("And will be released when the timeout expires.")
00259     {
00260         auto future = std::async(std::launch::async, [&]()
00261         {
00262             return queue.pop(1ms);
00263         });
00264         auto status = future.wait_for(0s);
00265         REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00266         REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00267         REQUIRE(future.get() == nullptr);
00268     }
00269     SECTION("And will be released when the 'close' method is called.")
00270     {
00271         auto future1 = std::async(std::launch::async, [&]()
00272         {
00273             return queue.pop(10s);
00274         });
00275         auto future2 = std::async(std::launch::async, [&]()
00276         {
00277             return queue.pop(10s);
00278         });
00279         REQUIRE(future1.wait_for(0s) != std::future_status::ready);
00280         REQUIRE(future2.wait_for(0s) != std::future_status::ready);
00281         queue.close();
```

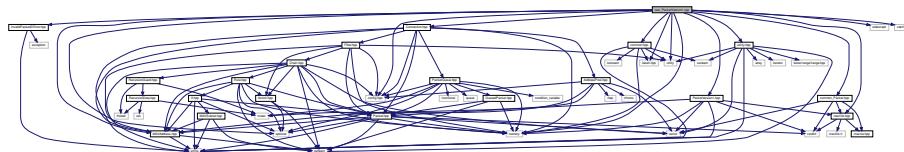
```

00282     REQUIRE(future1.get() == nullptr);
00283     REQUIRE(future2.get() == nullptr);
00284 }
00285 }
00286
00287
00288 TEST_CASE("PacketQueue's 'pop' method is non blocking when given a 0 second "
00289             "timeout.", "[PacketQueue]")
00290 {
00291     auto ping = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00292     PacketQueue queue;
00293     SECTION("Returns the packet when it is available.")
00294     {
00295         queue.push(ping);
00296         auto packet = queue.pop(0s);
00297         REQUIRE(packet != nullptr);
00298         REQUIRE(*packet == *ping);
00299     }
00300     SECTION("Returns nullptr when the queue is empty.")
00301     {
00302         REQUIRE(queue.pop(0s) == nullptr);
00303     }
00304 }
```

16.241 test_PacketVersion1.cpp File Reference

```

#include <memory>
#include <optional>
#include <stdexcept>
#include <utility>
#include <catch.hpp>
#include <fakeit.hpp>
#include "config.hpp"
#include "Connection.hpp"
#include "InvalidPacketIDError.hpp"
#include "MAVAddress.hpp"
#include "mavlink.hpp"
#include "PacketVersion1.hpp"
#include "utility.hpp"
#include "common.hpp"
#include "common_Packet.hpp"
Include dependency graph for test_PacketVersion1.cpp:
```



Functions

- [TEST_CASE](#) ("packet_v1::header_complete' determines whether the given bytes " "at least represent a complete header.", "[packet_v1]")
- [TEST_CASE](#) ("packet_v1::header' returns a structure pointer to the given " "[header](#) data.", "[packet_v1]")
- [TEST_CASE](#) ("packet_v1::packet_complete' determines whether the given bytes " "represent a complete packet.", "[packet_v1]")

- [TEST_CASE \("packet_v1::Packet's can be constructed.", "\[packet_v1::Packet\]"\)](#)
- [TEST_CASE \("packet_v1::Packet's are comparable.", "\[packet_v1::Packet\]"\)](#)
- [TEST_CASE \("packet_v1::Packet's are copyable.", "\[packet_v1::Packet\]"\)](#)
- [TEST_CASE \("packet_v1::Packet's are movable.", "\[packet_v1::Packet\]"\)](#)
- [TEST_CASE \("packet_v1::Packet's are assignable.", "\[Packet\]"\)](#)
- [TEST_CASE \("packet_v1::Packet's contain raw packet data and make it accessible.", "\[packet_v1::Packet\]"\)](#)
- [TEST_CASE \("packet_v1::Packet's have a version.", "\[packet_v1::Packet\]"\)](#)
- [TEST_CASE \("packet_v1::Packet's have an ID.", "\[packet_v1::Packet\]"\)](#)
- [TEST_CASE \("packet_v1::Packet's have a name.", "\[packet_v1::Packet\]"\)](#)
- [TEST_CASE \("packet_v1::Packet's have a source address.", "\[packet_v1::Packet\]"\)](#)
- [TEST_CASE \("packet_v1::Packet's optionally have a destination address.", "\[packet_v1::Packet\]"\)](#)
- [TEST_CASE \("packet_v1::Packet's optionally have a source connection.", "\[packet_v1::Packet\]"\)](#)
- [TEST_CASE \("packet_v1::Packet's are printable.", "\[packet_v1::Packet\]"\)](#)

Variables

- [packet_v1::Packet packet_b \(to_vector\(SetModeV1\(\)\)\)](#)
- [packet_a = packet_b](#)

16.241.1 Function Documentation

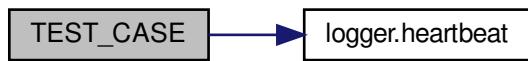
16.241.1.1 TEST_CASE() [1/16]

```
TEST_CASE (
    "'packet_v1::header_complete' determines whether the given bytes " "at least represent
a complete header." ,
    "" [packet_v1] )
```

Definition at line 38 of file [test_PacketVersion1.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



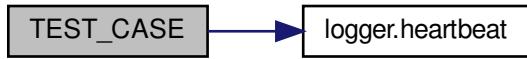
16.241.1.2 TEST_CASE() [2/16]

```
TEST_CASE (
    "'packet_v1::header' returns a structure pointer to the given \"header\" data." ,
    "" [packet_v1] )
```

Definition at line 80 of file [test_PacketVersion1.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.241.1.3 TEST_CASE() [3/16]

```
TEST_CASE (
    "'packet_v1::packet_complete' determines whether the given bytes \"represent a complete
packet.\" ,
    "" [packet_v1] )
```

Definition at line 154 of file [test_PacketVersion1.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



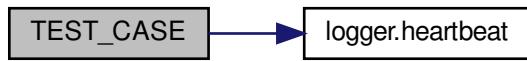
16.241.1.4 TEST_CASE() [4/16]

```
TEST_CASE (
    "packet_v1::Packet's can be constructed." ,
    "" [packet_v1::Packet] )
```

Definition at line 204 of file [test_PacketVersion1.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:

**16.241.1.5 TEST_CASE() [5/16]**

```
TEST_CASE (
    "packet_v1::Packet's are comparable." ,
    "" [packet_v1::Packet] )
```

Definition at line 285 of file [test_PacketVersion1.cpp](#).

16.241.1.6 TEST_CASE() [6/16]

```
TEST_CASE (
    "packet_v1::Packet's are copyable." ,
    "" [packet_v1::Packet] )
```

Definition at line 308 of file [test_PacketVersion1.cpp](#).

16.241.1.7 TEST_CASE() [7/16]

```
TEST_CASE (
    "packet_v1::Packet's are movable." ,
    "" [packet_v1::Packet] )
```

Definition at line 316 of file [test_PacketVersion1.cpp](#).

16.241.1.8 TEST_CASE() [8/16]

```
TEST_CASE (
    "packet_v1::Packet's are assignable." ,
    "" [Packet] )
```

Definition at line 324 of file [test_PacketVersion1.cpp](#).

References [packet_a](#), and [packet_b](#).

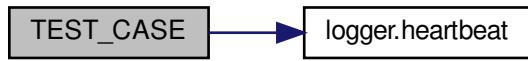
16.241.1.9 TEST_CASE() [9/16]

```
TEST_CASE (
    "packet_v1::Packet's contain raw packet data and make it accessible." ,
    "" [packet_v1::Packet] )
```

Definition at line 344 of file [test_PacketVersion1.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.241.1.10 TEST_CASE() [10/16]

```
TEST_CASE (
    "packet_v1::Packet's have a version." ,
    "" [packet_v1::Packet] )
```

Definition at line 358 of file [test_PacketVersion1.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



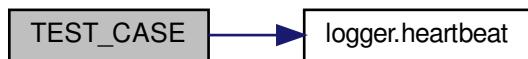
16.241.1.11 TEST_CASE() [11/16]

```
TEST_CASE (
    "packet_v1::Packet's have an ID." ,
    "" [packet_v1::Packet] )
```

Definition at line 372 of file [test_PacketVersion1.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.241.1.12 TEST_CASE() [12/16]

```
TEST_CASE (
    "packet_v1::Packet's have a name." ,
    "" [packet_v1::Packet] )
```

Definition at line 385 of file [test_PacketVersion1.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:

**16.241.1.13 TEST_CASE()** [13/16]

```
TEST_CASE (
    "packet_v1::Packet's have a source address." ,
    "" [packet_v1::Packet] )
```

Definition at line 398 of file [test_PacketVersion1.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.241.1.14 TEST_CASE() [14/16]

```
TEST_CASE (
    "packet_v1::Packet's optionally have a destination address." ,
    "" [packet_v1::Packet] )
```

Definition at line 412 of file [test_PacketVersion1.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



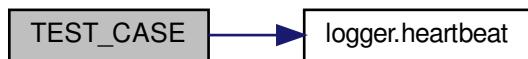
16.241.1.15 TEST_CASE() [15/16]

```
TEST_CASE (
    "packet_v1::Packet's optionally have a source connection." ,
    "" [packet_v1::Packet] )
```

Definition at line 429 of file [test_PacketVersion1.cpp](#).

References [logger::heartbeat\(\)](#).

Here is the call graph for this function:



16.241.1.16 TEST_CASE() [16/16]

```
TEST_CASE (
    "packet_v1::Packet's are printable." ,
    "" [packet_v1::Packet] )
```

Definition at line 450 of file [test_PacketVersion1.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.241.2 Variable Documentation

16.241.2.1 packet_a

```
packet_a = packet\_b
```

Definition at line 339 of file [test_PacketVersion1.cpp](#).

16.241.2.2 packet_b

```
packet_v1::Packet packet_b(to_vector(SetModeV1()))
```

Initial value:

```
{  
    packet_v1::Packet packet_a(to_vector(PingV1()))
```

16.242 test_PacketVersion1.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <memory>
00019 #include <optional>
00020 #include <stdexcept>
00021 #include <utility>
00022
00023 #include <catch.hpp>
00024 #include <fakeit.hpp>
00025
00026 #include "config.hpp"
00027 #include "Connection.hpp"
00028 #include "InvalidPacketIDError.hpp"
00029 #include "MAVAddress.hpp"
00030 #include "mavlink.hpp"
00031 #include "PacketVersion1.hpp"
00032 #include "utility.hpp"
00033
00034 #include "common.hpp"
00035 #include "common_Packet.hpp"
00036
00037
00038 TEST_CASE("'packet_v1::header_complete' determines whether the given bytes "
00039             "at least represent a complete header.", "[packet_v1]")
00040 {
00041     auto heartbeat = to_vector(HeartbeatV1());
00042     auto ping = to_vector(PingV1());
00043     auto set_mode = to_vector(SetModeV1());
00044     auto encapsulated_data = to_vector(EncapsulatedDataV1());
00045     SECTION("Returns true when at least a complete header is given.")
00046     {
00047         heartbeat.resize(6);
00048         ping.resize(10);
00049         set_mode.resize(15);
00050         REQUIRE(packet_v1::header_complete(heartbeat));
00051         REQUIRE(packet_v1::header_complete(ping));
00052         REQUIRE(packet_v1::header_complete(set_mode));
00053         REQUIRE(packet_v1::header_complete(encapsulated_data));
00054     }
00055     SECTION("Returns false when an incomplete header is given.")
00056     {
00057         heartbeat.resize(5);
00058         ping.resize(4);
00059         set_mode.resize(3);
00060         encapsulated_data.resize(0);
00061         REQUIRE_FALSE(packet_v1::header_complete(
00062             heartbeat));
00063         REQUIRE_FALSE(packet_v1::header_complete(ping));
00064         REQUIRE_FALSE(packet_v1::header_complete(set_mode));
00065     }
00066     SECTION("Returns false when the magic byte is wrong.")
00067     {
00068         heartbeat.front() = 0xAD;
00069         ping.front() = 0xBC;
00070         set_mode.front() = 0xFD;
00071         encapsulated_data.front() = 0xFD;
00072         REQUIRE_FALSE(packet_v1::header_complete(
00073             heartbeat));
00074         REQUIRE_FALSE(packet_v1::header_complete(ping));
00075         REQUIRE_FALSE(packet_v1::header_complete(set_mode));
00076         REQUIRE_FALSE(packet_v1::header_complete(encapsulated_data));
```

```

00076      }
00077  }
00078
00079
00080 TEST_CASE("'packet_v1::header' returns a structure pointer to the given "
00081         "header data.", "[packet_v1]")
00082 {
00083     auto heartbeat = to_vector(HeartbeatV1());
00084     auto ping = to_vector(PingV1());
00085     auto set_mode = to_vector(SetModeV1());
00086     auto encapsulated_data = to_vector(EncapsulatedDataV1());
00087     SECTION("Header contains a magic value.")
00088     {
00089         REQUIRE(packet_v1::header(heartbeat)->magic == 0xFE);
00090         REQUIRE(packet_v1::header(ping)->magic == 0xFE);
00091         REQUIRE(packet_v1::header(set_mode)->magic == 0xFE);
00092         REQUIRE(packet_v1::header(encapsulated_data)->magic == 0xFE);
00093     }
00094     SECTION("Header stores the payload length.")
00095     {
00096         REQUIRE(packet_v1::header(heartbeat)->len == 9);
00097         REQUIRE(packet_v1::header(ping)->len == 14);
00098         REQUIRE(packet_v1::header(set_mode)->len == 6);
00099         REQUIRE(packet_v1::header(encapsulated_data)->len == 255);
00100 }
00101 SECTION("Header has a sequence number.")
00102 {
00103     REQUIRE(packet_v1::header(heartbeat)->seq == 0xFE);
00104     REQUIRE(packet_v1::header(ping)->seq == 0xFE);
00105     REQUIRE(packet_v1::header(set_mode)->seq == 0xFE);
00106     REQUIRE(packet_v1::header(encapsulated_data)->seq == 0xFE);
00107 }
00108 SECTION("Header has a system ID.")
00109 {
00110     REQUIRE(packet_v1::header(heartbeat)->sysid == 127);
00111     REQUIRE(packet_v1::header(ping)->sysid == 192);
00112     REQUIRE(packet_v1::header(set_mode)->sysid == 172);
00113     REQUIRE(packet_v1::header(encapsulated_data)->sysid == 224);
00114 }
00115 SECTION("Header has a component ID.")
00116 {
00117     REQUIRE(packet_v1::header(heartbeat)->compid == 1);
00118     REQUIRE(packet_v1::header(ping)->compid == 168);
00119     REQUIRE(packet_v1::header(set_mode)->compid == 0);
00120     REQUIRE(packet_v1::header(encapsulated_data)->compid == 255);
00121 }
00122 SECTION("Header has a message ID.")
00123 {
00124     REQUIRE(packet_v1::header(heartbeat)->msgid == 0);
00125     REQUIRE(packet_v1::header(ping)->msgid == 4);
00126     REQUIRE(packet_v1::header(set_mode)->msgid == 11);
00127     REQUIRE(packet_v1::header(encapsulated_data)->msgid == 131);
00128 }
00129 SECTION("Returns nullptr when an incomplete header is given.")
00130 {
00131     heartbeat.resize(5);
00132     ping.resize(4);
00133     set_mode.resize(3);
00134     encapsulated_data.resize(0);
00135     REQUIRE(packet_v1::header(heartbeat) == nullptr);
00136     REQUIRE(packet_v1::header(ping) == nullptr);
00137     REQUIRE(packet_v1::header(set_mode) == nullptr);
00138     REQUIRE(packet_v1::header(encapsulated_data) == nullptr);
00139 }
00140 SECTION("Returns nullptr when the magic byte is wrong.")
00141 {
00142     heartbeat.front() = 0xAD;
00143     ping.front() = 0xBC;
00144     set_mode.front() = 0xFD;
00145     encapsulated_data.front() = 0xFD;
00146     REQUIRE(packet_v1::header(heartbeat) == nullptr);
00147     REQUIRE(packet_v1::header(ping) == nullptr);
00148     REQUIRE(packet_v1::header(set_mode) == nullptr);
00149     REQUIRE(packet_v1::header(encapsulated_data) == nullptr);
00150 }
00151 }
00152
00153
00154 TEST_CASE("'packet_v1::packet_complete' determines whether the given bytes "
00155         "represent a complete packet.", "[packet_v1]")
00156 {

```

```
00157     auto heartbeat = to_vector(HeartbeatV1());
00158     auto ping = to_vector(PingV1());
00159     auto set_mode = to_vector(SetModeV1());
00160     auto encapsulated_data = to_vector(EncapsulatedDataV1());
00161     SECTION("Returns true when a complete packet is given.")
00162     {
00163         REQUIRE(packet_v1::packet_complete(heartbeat));
00164         REQUIRE(packet_v1::packet_complete(ping));
00165         REQUIRE(packet_v1::packet_complete(set_mode));
00166         REQUIRE(packet_v1::packet_complete(encapsulated_data));
00167     }
00168     SECTION("Returns false when the magic byte is wrong.")
00169     {
00170         heartbeat.front() = 0xAD;
00171         ping.front() = 0xBC;
00172         set_mode.front() = 0xFD;
00173         encapsulated_data.front() = 0xFD;
00174         REQUIRE_FALSE(packet_v1::packet_complete(
00175             heartbeat));
00176         REQUIRE_FALSE(packet_v1::packet_complete(ping));
00177         REQUIRE_FALSE(packet_v1::packet_complete(set_mode));
00178         REQUIRE_FALSE(packet_v1::packet_complete(encapsulated_data));
00179     }
00180     SECTION("Returns false when the packet is too short.")
00181     {
00182         heartbeat.pop_back();
00183         ping.pop_back();
00184         set_mode.pop_back();
00185         encapsulated_data.pop_back();
00186         REQUIRE_FALSE(packet_v1::packet_complete(
00187             heartbeat));
00188         REQUIRE_FALSE(packet_v1::packet_complete(ping));
00189         REQUIRE_FALSE(packet_v1::packet_complete(set_mode));
00190         REQUIRE_FALSE(packet_v1::packet_complete(encapsulated_data));
00191     }
00192     SECTION("Returns false when the packet is too long.")
00193     {
00194         heartbeat.push_back(0x00);
00195         ping.push_back(0x00);
00196         set_mode.push_back(0x00);
00197         encapsulated_data.push_back(0x00);
00198         REQUIRE_FALSE(packet_v1::packet_complete(
00199             heartbeat));
00200         REQUIRE_FALSE(packet_v1::packet_complete(ping));
00201         REQUIRE_FALSE(packet_v1::packet_complete(set_mode));
00202         REQUIRE_FALSE(packet_v1::packet_complete(encapsulated_data));
00203     }
00204 TEST_CASE("packet_v1::Packet's can be constructed.", "[packet_v1::Packet]")
00205 {
00206     HeartbeatV1 heartbeat;
00207     PingV1 ping;
00208     SetModeV1 set_mode;
00209     EncapsulatedDataV1 encapsulated_data;
00210     SECTION("With proper arguments.")
00211     {
00212         REQUIRE_NO_THROW(packet_v1::Packet(to_vector(heartbeat)));
00213         REQUIRE_NO_THROW(packet_v1::Packet(to_vector(ping)));
00214         REQUIRE_NO_THROW(packet_v1::Packet(to_vector(set_mode)));
00215         REQUIRE_NO_THROW(packet_v1::Packet(to_vector(encapsulated_data)));
00216     }
00217     SECTION("And ensures a complete header is given.")
00218     {
00219         REQUIRE_THROWS_AS(packet_v1::Packet({}), std::length_error);
00220         REQUIRE_THROWS_WITH(packet_v1::Packet({}), "Packet is empty.");
00221         REQUIRE_THROWS_AS(
00222             packet_v1::Packet({0xFE, 2, 3, 4, 5}), std::length_error);
00223         REQUIRE_THROWS_WITH(
00224             packet_v1::Packet({0xFE, 2, 3, 4, 5}),
00225             "Packet (5 bytes) is shorter than a v1.0 header (6 bytes).");
00226     }
00227     SECTION("And ensures packets begins with the magic byte (0xFE).")
00228     {
00229         ping.magic = 0xAD;
00230         REQUIRE_THROWS_AS(
00231             packet_v1::Packet(to_vector(ping)), std::invalid_argument);
00232         REQUIRE_THROWS_WITH(
00233             packet_v1::Packet(to_vector(ping)),
00234             "Invalid packet starting byte (0xAD), "
00235         );
00236     }
00237 }
```

```

00235         "v1.0 packets should start with 0xFE.");
00236     }
00237     SECTION("And ensures the message ID is valid.")
00238     {
00239         ping.msgid = 255; // ID 255 is not currently valid.
00240         REQUIRE_THROWS_AS(
00241             packet_v1::Packet(to_vector(ping)),
00242             InvalidPacketIDError);
00243         REQUIRE_THROWS_WITH(
00244             packet_v1::Packet(to_vector(ping)),
00245             "Packet ID (#255) is not part of the '" +
00246             MAVLINK_DIALECT "' MAVLink dialect.");
00247     }
00248     SECTION("And ensures the packet is the correct length.")
00249     {
00250         // HEARTBEAT (no target system/component).
00251         auto heartbeat_data = to_vector(heartbeat);
00252         heartbeat_data.pop_back();
00253         REQUIRE_THROWS_AS(
00254             packet_v1::Packet(heartbeat_data), std::length_error);
00255         REQUIRE_THROWS_WITH(
00256             packet_v1::Packet(heartbeat_data),
00257             "Packet is 16 bytes, should be 17 bytes.");
00258         // PING (with target system/component).
00259         auto ping_data = to_vector(ping);
00260         ping_data.push_back(0x00);
00261         REQUIRE_THROWS_AS(
00262             packet_v1::Packet(ping_data), std::length_error);
00263         REQUIRE_THROWS_WITH(
00264             packet_v1::Packet(ping_data),
00265             "Packet is 23 bytes, should be 22 bytes.");
00266         // SET_MODE (target system only, no target component).
00267         auto set_mode_data = to_vector(set_mode);
00268         set_mode_data.pop_back();
00269         REQUIRE_THROWS_AS(
00270             packet_v1::Packet(set_mode_data), std::length_error);
00271         REQUIRE_THROWS_WITH(
00272             packet_v1::Packet(set_mode_data),
00273             "Packet is 13 bytes, should be 14 bytes.");
00274         // ENCAPSULATED_DATA (longest packet).
00275         auto encapsulated_data_data = to_vector(encapsulated_data);
00276         encapsulated_data_data.push_back(0x00);
00277         REQUIRE_THROWS_AS(
00278             packet_v1::Packet(encapsulated_data_data), std::length_error);
00279         REQUIRE_THROWS_WITH(
00280             packet_v1::Packet(encapsulated_data_data),
00281             "Packet is 264 bytes, should be 263 bytes.");
00282     }
00283
00284
00285 TEST_CASE("packet_v1::Packet's are comparable.", "[packet_v1::Packet]")
00286 {
00287     SECTION("with ==")
00288     {
00289         REQUIRE(
00290             packet_v1::Packet(to_vector(PingV1())) ==
00291             packet_v1::Packet(to_vector(PingV1())));
00292         REQUIRE_FALSE(
00293             packet_v1::Packet(to_vector(PingV1())) ==
00294             packet_v1::Packet(to_vector(SetModeV1())));
00295     }
00296     SECTION("with !=")
00297     {
00298         REQUIRE(
00299             packet_v1::Packet(to_vector(PingV1())) !=
00300             packet_v1::Packet(to_vector(SetModeV1())));
00301         REQUIRE_FALSE(
00302             packet_v1::Packet(to_vector(PingV1())) !=
00303             packet_v1::Packet(to_vector(PingV1())));
00304     }
00305 }
00306
00307
00308 TEST_CASE("packet_v1::Packet's are copyable.", "[packet_v1::Packet]")
00309 {
00310     packet_v1::Packet original(to_vector(PingV1()));
00311     packet_v1::Packet copy(original);
00312     REQUIRE(copy == packet_v1::Packet(to_vector(PingV1())));
00313 }
00314

```

```
00315
00316 TEST_CASE("packet_v1::Packet's are movable.", "[packet_v1::Packet]")
00317 {
00318     packet_v1::Packet original(to_vector(PingV1()));
00319     packet_v1::Packet moved(std::move(original));
00320     REQUIRE(moved == packet_v1::Packet(to_vector(PingV1())));
00321 }
00322
00323
00324 TEST_CASE("packet_v1::Packet's are assignable.", "[Packet]")
00325 {
00326     packet_v1::Packet packet_a(to_vector(PingV1()));
00327     packet_v1::Packet packet_b(to_vector(SetModeV1()));
00328     REQUIRE(packet_a == packet_v1::Packet(to_vector(PingV1())));
00329     packet_a = packet_b;
00330     REQUIRE(packet_b == packet_v1::Packet(to_vector(SetModeV1())));
00331 }
00332
00333
00334 TEST_CASE("packet_v1::Packet's are assignable (by move semantics).", "[Packet]")
00335 {
00336     packet_v1::Packet packet_a(to_vector(PingV1()));
00337     packet_v1::Packet packet_b(to_vector(SetModeV1()));
00338     REQUIRE(packet_a == packet_v1::Packet(to_vector(PingV1())));
00339     packet_a = packet_b;
00340     REQUIRE(packet_b == packet_v1::Packet(to_vector(SetModeV1())));
00341 }
00342
00343
00344 TEST_CASE("packet_v1::Packet's contain raw packet data and make it accessible.", "[packet_v1::Packet]")
00345
00346 {
00347     auto heartbeat = to_vector(HeartbeatV1());
00348     auto ping = to_vector(PingV1());
00349     auto set_mode = to_vector(SetModeV1());
00350     auto encapsulated_data = to_vector(EncapsulatedDataV1());
00351     REQUIRE(packet_v1::Packet(heartbeat).data() ==
00352         heartbeat);
00353     REQUIRE(packet_v1::Packet(ping).data() == ping);
00354     REQUIRE(packet_v1::Packet(set_mode).data() == set_mode);
00355     REQUIRE(packet_v1::Packet(encapsulated_data).data() == encapsulated_data);
00356 }
00357
00358 TEST_CASE("packet_v1::Packet's have a version.", "[packet_v1::Packet]")
00359 {
00360     auto heartbeat = to_vector(HeartbeatV1());
00361     auto ping = to_vector(PingV1());
00362     auto set_mode = to_vector(SetModeV1());
00363     auto encapsulated_data = to_vector(EncapsulatedDataV1());
00364     // All should read 0x0100 for v1.0.
00365     REQUIRE(packet_v1::Packet(heartbeat).version() ==
00366         Packet::V1);
00367     REQUIRE(packet_v1::Packet(ping).version() == Packet::V1);
00368     REQUIRE(packet_v1::Packet(set_mode).version() == 0x100);
00369 }
00370
00371
00372 TEST_CASE("packet_v1::Packet's have an ID.", "[packet_v1::Packet]")
00373 {
00374     auto heartbeat = to_vector(HeartbeatV1());
00375     auto ping = to_vector(PingV1());
00376     auto set_mode = to_vector(SetModeV1());
00377     auto encapsulated_data = to_vector(EncapsulatedDataV1());
00378     REQUIRE(packet_v1::Packet(heartbeat).id() == 0);
00379     REQUIRE(packet_v1::Packet(ping).id() == 4);
00380     REQUIRE(packet_v1::Packet(set_mode).id() == 11);
00381     REQUIRE(packet_v1::Packet(encapsulated_data).id() == 131);
00382 }
00383
00384
00385 TEST_CASE("packet_v1::Packet's have a name.", "[packet_v1::Packet]")
00386 {
00387     auto heartbeat = to_vector(HeartbeatV1());
00388     auto ping = to_vector(PingV1());
00389     auto set_mode = to_vector(SetModeV1());
00390     auto encapsulated_data = to_vector(EncapsulatedDataV1());
00391     REQUIRE(packet_v1::Packet(heartbeat).name() == "HEARTBEAT");
00392     REQUIRE(packet_v1::Packet(ping).name() == "PING");
00393     REQUIRE(packet_v1::Packet(set_mode).name() == "SET_MODE");
```

```

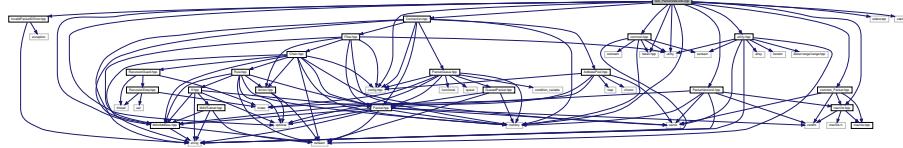
00394     REQUIRE(packet_v1::Packet(encapsulated_data).name() == "ENCAPSULATED_DATA");
00395 }
00396
00397
00398 TEST_CASE("packet_v1::Packet's have a source address.", "[packet_v1::Packet]")
00399 {
00400     auto heartbeat = to_vector(HeartbeatV1());
00401     auto ping = to_vector(PingV1());
00402     auto set_mode = to_vector(SetModeV1());
00403     auto encapsulated_data = to_vector(EncapsulatedDataV1());
00404     REQUIRE(packet_v1::Packet(heartbeat).source() ==
00405         MAVAddress("127.1"));
00406     REQUIRE(packet_v1::Packet(ping).source() == MAVAddress("192.168"));
00407     REQUIRE(packet_v1::Packet(set_mode).source() == MAVAddress("172.0"));
00408     REQUIRE(
00409         packet_v1::Packet(encapsulated_data).source() ==
00410         MAVAddress("224.255"));
00411
00412 TEST_CASE("packet_v1::Packet's optionally have a destination address.",
00413             "[packet_v1::Packet]")
00414 {
00415     auto heartbeat = to_vector(HeartbeatV1());
00416     auto ping = to_vector(PingV1());
00417     auto set_mode = to_vector(SetModeV1());
00418     auto encapsulated_data = to_vector(EncapsulatedDataV1());
00419     REQUIRE_THROWS_AS(
00420         packet_v1::Packet(heartbeat).dest().value(), std::bad_optional_access);
00421     REQUIRE(packet_v1::Packet(ping).dest().value() ==
00422         MAVAddress("127.1"));
00423     REQUIRE(packet_v1::Packet(set_mode).dest().value() ==
00424         MAVAddress("123.0"));
00425     REQUIRE_THROWS_AS(
00426         packet_v1::Packet(encapsulated_data).dest().value(),
00427         std::bad_optional_access);
00428
00429 TEST_CASE("packet_v1::Packet's optionally have a source connection.",
00430             "[packet_v1::Packet]")
00431 {
00432     auto heartbeat = packet_v1::Packet(to_vector(HeartbeatV1()));
00433     SECTION("Defaults to nullptr.")
00434     {
00435         REQUIRE(heartbeat.connection() == nullptr);
00436     }
00437     SECTION("Can be set with the 'connection' method.")
00438     {
00439         fakeit::Mock<Filter> mock_filter;
00440         auto filter = mock_shared(mock_filter);
00441         auto conn = std::make_shared<Connection>("SOURCE", filter);
00442         heartbeat.connection(conn);
00443         REQUIRE(heartbeat.connection() != nullptr);
00444         REQUIRE(heartbeat.connection() == conn);
00445         REQUIRE(str(*heartbeat.connection()) == "SOURCE");
00446     }
00447 }
00448
00449
00450 TEST_CASE("packet_v1::Packet's are printable.", "[packet_v1::Packet]")
00451 {
00452     auto heartbeat = to_vector(HeartbeatV1());
00453     auto ping = to_vector(PingV1());
00454     auto set_mode = to_vector(SetModeV1());
00455     auto encapsulated_data = to_vector(EncapsulatedDataV1());
00456     REQUIRE(
00457         str(packet_v1::Packet(heartbeat)) ==
00458         "HEARTBEAT (#0) from 127.1 (v1.0)");
00459     REQUIRE(
00460         str(packet_v1::Packet(ping)) ==
00461         "PING (#4) from 192.168 to 127.1 (v1.0)");
00462     REQUIRE(
00463         str(packet_v1::Packet(set_mode)) ==
00464         "SET_MODE (#11) from 172.0 to 123.0 (v1.0)");
00465     REQUIRE(
00466         str(packet_v1::Packet(encapsulated_data)) ==
00467         "ENCAPSULATED_DATA (#131) from 224.255 (v1.0)");
00468 }

```

16.243 test_PacketVersion2.cpp File Reference

```
#include <memory>
#include <optional>
#include <stdexcept>
#include <utility>
#include <catch.hpp>
#include <fakeit.hpp>
#include "config.hpp"
#include "Connection.hpp"
#include "InvalidPacketIDError.hpp"
#include "MAVAddress.hpp"
#include "mavlink.hpp"
#include "PacketVersion2.hpp"
#include "utility.hpp"
#include "common.hpp"
#include "common_Packet.hpp"
```

Include dependency graph for test_PacketVersion2.cpp:



Functions

- [`TEST_CASE` \("packet_v2::header_complete' determines whether the given bytes " "at least represent a complete header.", "\[packet_v2\]"\)](#)
- [`TEST_CASE` \("packet_v2::header' returns a structure pointer to the given " "header data.", "\[packet_v2\]"\)](#)
- [`TEST_CASE` \("packet_v2::packet_complete' determines whether the given bytes " "represent a complete packet.", "\[packet_v2\]"\)](#)
- [`TEST_CASE` \("packet_v2::is_signed' determines whether the given bytes " "represent a signed packet.", "\[packet_v2\]"\)](#)
- [`TEST_CASE` \("packet_v2::Packet's can be constructed.", "\[packet_v2::Packet\]"\)](#)
- [`TEST_CASE` \("packet_v2::Packet's are comparable.", "\[packet_v2::Packet\]"\)](#)
- [`TEST_CASE` \("packet_v2::Packet's are copyable.", "\[packet_v2::Packet\]"\)](#)
- [`TEST_CASE` \("packet_v2::Packet's are movable.", "\[packet_v2::Packet\]"\)](#)
- [`TEST_CASE` \("packet_v2::Packet's are assignable.", "\[Packet\]"\)](#)
- [`TEST_CASE` \("packet_v2::Packet's contain raw packet data and make it accessible.", "\[packet_v2::Packet\]"\)](#)
- [`TEST_CASE` \("packet_v2::Packet's have a version.", "\[packet_v2::Packet\]"\)](#)
- [`TEST_CASE` \("packet_v2::Packet's have an ID.", "\[packet_v2::Packet\]"\)](#)
- [`TEST_CASE` \("packet_v2::Packet's have a name.", "\[packet_v2::Packet\]"\)](#)
- [`TEST_CASE` \("packet_v2::Packet's have a source address.", "\[packet_v2::Packet\]"\)](#)
- [`TEST_CASE` \("packet_v2::Packet's optionally have a destination address.", "\[packet_v2::Packet\]"\)](#)
- [`TEST_CASE` \("packet_v2::Packet's optionally have a source connection.", "\[packet_v2::Packet\]"\)](#)
- [`TEST_CASE` \("packet_v2::Packet's are printable.", "\[packet_v2::Packet\]"\)](#)

Variables

- `packet_v2::Packet packet_b (to_vector(SetModeV2())))`
- `packet_a = packet_b`

16.243.1 Function Documentation

16.243.1.1 TEST_CASE() [1/17]

```
TEST_CASE (
    "'packet_v2::header_complete' determines whether the given bytes " "at least represent
a complete header." ,
    "" [packet_v2] )
```

Definition at line 38 of file [test_PacketVersion2.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



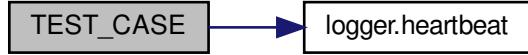
16.243.1.2 TEST_CASE() [2/17]

```
TEST_CASE (
    "'packet_v2::header' returns a structure pointer to the given " "header" data." ,
    "" [packet_v2] )
```

Definition at line 94 of file [test_PacketVersion2.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.243.1.3 TEST_CASE() [3/17]

```
TEST_CASE (
    "'packet_v2::packet_complete' determines whether the given bytes " "represent a complete
packet." ,
    "" [packet_v2] )
```

Definition at line 214 of file [test_PacketVersion2.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.243.1.4 TEST_CASE() [4/17]

```
TEST_CASE (
    "'packet_v2::is_signed' determines whether the given bytes " "represent a signed
packet." ,
    "" [packet_v2] )
```

Definition at line 280 of file [test_PacketVersion2.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.243.1.5 TEST_CASE() [5/17]

```
TEST_CASE (
    "packet_v2::Packet's can be constructed." ,
    "" [packet_v2::Packet] )
```

Definition at line 346 of file [test_PacketVersion2.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.243.1.6 TEST_CASE() [6/17]

```
TEST_CASE (
    "packet_v2::Packet's are comparable." ,
    "" [packet_v2::Packet] )
```

Definition at line 515 of file [test_PacketVersion2.cpp](#).

16.243.1.7 TEST_CASE() [7/17]

```
TEST_CASE (
    "packet_v2::Packet's are copyable." ,
    "" [packet_v2::Packet] )
```

Definition at line 544 of file [test_PacketVersion2.cpp](#).

16.243.1.8 TEST_CASE() [8/17]

```
TEST_CASE (
    "packet_v2::Packet's are movable." ,
    "" [packet_v2::Packet] )
```

Definition at line 552 of file [test_PacketVersion2.cpp](#).

16.243.1.9 TEST_CASE() [9/17]

```
TEST_CASE (
    "packet_v2::Packet's are assignable." ,
    "" [Packet] )
```

Definition at line 560 of file [test_PacketVersion2.cpp](#).

References [packet_a](#), and [packet_b](#).

16.243.1.10 TEST_CASE() [10/17]

```
TEST_CASE (
    "packet_v2::Packet's contain raw packet data and make it accessible." ,
    "" [packet_v2::Packet] )
```

Definition at line 580 of file [test_PacketVersion2.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.243.1.11 TEST_CASE() [11/17]

```
TEST_CASE (
    "packet_v2::Packet's have a version." ,
    "" [packet_v2::Packet] )
```

Definition at line 602 of file [test_PacketVersion2.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:

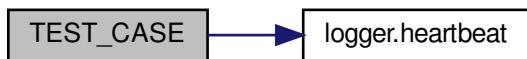
**16.243.1.12 TEST_CASE()** [12/17]

```
TEST_CASE (
    "packet_v2::Packet's have an ID." ,
    "" [packet_v2::Packet] )
```

Definition at line 620 of file [test_PacketVersion2.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.243.1.13 TEST_CASE() [13/17]

```
TEST_CASE (
    "packet_v2::Packet's have a name." ,
    "" [packet_v2::Packet] )
```

Definition at line 637 of file [test_PacketVersion2.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



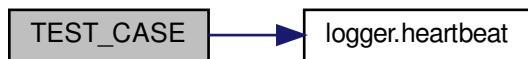
16.243.1.14 TEST_CASE() [14/17]

```
TEST_CASE (
    "packet_v2::Packet's have a source address." ,
    "" [packet_v2::Packet] )
```

Definition at line 657 of file [test_PacketVersion2.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



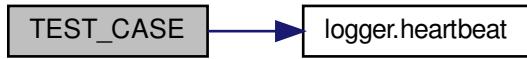
16.243.1.15 TEST_CASE() [15/17]

```
TEST_CASE (
    "packet_v2::Packet's optionally have a destination address." ,
    "" [packet_v2::Packet] )
```

Definition at line 678 of file [test_PacketVersion2.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:

**16.243.1.16 TEST_CASE()** [16/17]

```
TEST_CASE (
    "packet_v2::Packet's optionally have a source connection." ,
    "" [packet_v2::Packet] )
```

Definition at line 703 of file [test_PacketVersion2.cpp](#).

References [logger::heartbeat\(\)](#).

Here is the call graph for this function:



16.243.1.17 TEST_CASE() [17/17]

```
TEST_CASE (
    "packet_v2::Packet's are printable." ,
    "" [packet_v2::Packet] )
```

Definition at line 724 of file [test_PacketVersion2.cpp](#).

References [logger::heartbeat\(\)](#), and [ping](#).

Here is the call graph for this function:



16.243.2 Variable Documentation

16.243.2.1 packet_a

```
packet_a = packet\_b
```

Definition at line 575 of file [test_PacketVersion2.cpp](#).

16.243.2.2 packet_b

```
packet_v2::Packet packet_b(to_vector(SetModeV2()))
```

Initial value:

```
{  
    packet_v2::Packet packet_a(to_vector(PingV2()))
```

16.244 test_PacketVersion2.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <memory>
00019 #include <optional>
00020 #include <stdexcept>
00021 #include <utility>
00022
00023 #include <catch.hpp>
00024 #include <fakeit.hpp>
00025
00026 #include "config.hpp"
00027 #include "Connection.hpp"
00028 #include "InvalidPacketIDError.hpp"
00029 #include "MAVAddress.hpp"
00030 #include "mavlink.hpp"
00031 #include "PacketVersion2.hpp"
00032 #include "utility.hpp"
00033
00034 #include "common.hpp"
00035 #include "common_Packet.hpp"
00036
00037
00038 TEST_CASE("'packet_v2::header_complete' determines whether the given bytes "
00039             "at least represent a complete header.", "[packet_v2]")
00040 {
00041     auto heartbeat = to_vector_with_sig(HeartbeatV2());
00042     auto ping = to_vector(PingV2());
00043     auto set_mode = to_vector_with_sig(SetModeV2());
00044     auto mission_set_current = to_vector(MissionSetCurrentV2());
00045     auto encapsulated_data = to_vector_with_sig(EncapsulatedDataV2());
00046     auto param_ext_request_list = to_vector(ParamExtRequestListV2());
00047     SECTION("Returns true when at least a complete header is given.")
00048     {
00049         heartbeat.resize(10);
00050         ping.resize(15);
00051         set_mode.resize(20);
00052         mission_set_current.resize(25);
00053         encapsulated_data.resize(30);
00054         REQUIRE(packet_v2::header_complete(heartbeat));
00055         REQUIRE(packet_v2::header_complete(ping));
00056         REQUIRE(packet_v2::header_complete(set_mode));
00057         REQUIRE(packet_v2::header_complete(mission_set_current));
00058         REQUIRE(packet_v2::header_complete(encapsulated_data));
00059         REQUIRE(packet_v2::header_complete(param_ext_request_list));
00060     }
00061     SECTION("Returns false when an incomplete header is given.")
00062     {
00063         heartbeat.resize(5);
00064         ping.resize(4);
00065         set_mode.resize(3);
00066         mission_set_current.resize(2);
00067         encapsulated_data.resize(1);
00068         param_ext_request_list.resize(0);
00069         REQUIRE_FALSE(packet_v2::header_complete(
00070             heartbeat));
00071         REQUIRE_FALSE(packet_v2::header_complete(ping));
00072         REQUIRE_FALSE(packet_v2::header_complete(set_mode));
00073         REQUIRE_FALSE(packet_v2::header_complete(mission_set_current));
00074         REQUIRE_FALSE(packet_v2::header_complete(encapsulated_data));
00075         REQUIRE_FALSE(packet_v2::header_complete(param_ext_request_list));
00076     }
00077     SECTION("Returns false when the magic byte is wrong.")

```

```

00077     {
00078         heartbeat.front() = 0xAD;
00079         ping.front() = 0xBC;
00080         set_mode.front() = 0xFE;
00081         mission_set_current.front() = 0xFE;
00082         encapsulated_data.front() = 0xFE;
00083         param_ext_request_list.front() = 0xFE;
00084         REQUIRE_FALSE(packet_v2::header_complete(
00085             heartbeat));
00085         REQUIRE_FALSE(packet_v2::header_complete(ping));
00086         REQUIRE_FALSE(packet_v2::header_complete(set_mode));
00087         REQUIRE_FALSE(packet_v2::header_complete(mission_set_current));
00088         REQUIRE_FALSE(packet_v2::header_complete(encapsulated_data));
00089         REQUIRE_FALSE(packet_v2::header_complete(param_ext_request_list));
00090     }
00091 }
00092
00093
00094 TEST_CASE("packet_v2::header' returns a structure pointer to the given "
00095             "header data.", "[packet_v2]")
00096 {
00097     auto heartbeat = to_vector_with_sig(HeartbeatV2());
00098     auto ping = to_vector(PingV2());
00099     auto set_mode = to_vector_with_sig(SetModeV2());
00100    auto mission_set_current = to_vector(MissionSetCurrentV2());
00101    auto encapsulated_data = to_vector_with_sig(EncapsulatedDataV2());
00102    auto param_ext_request_list = to_vector(ParamExtRequestListV2());
00103    SECTION("Header contains a magic value.")
00104    {
00105        REQUIRE(packet_v2::header(heartbeat)->magic == 0xFD);
00106        REQUIRE(packet_v2::header(ping)->magic == 0xFD);
00107        REQUIRE(packet_v2::header(set_mode)->magic == 0xFD);
00108        REQUIRE(packet_v2::header(mission_set_current)->magic == 0xFD);
00109        REQUIRE(packet_v2::header(encapsulated_data)->magic == 0xFD);
00110        REQUIRE(packet_v2::header(param_ext_request_list)->magic == 0xFD);
00111    }
00112    SECTION("Header stores the packet length.")
00113    {
00114        REQUIRE(packet_v2::header(heartbeat)->len == 9);
00115        REQUIRE(packet_v2::header(ping)->len == 14);
00116        REQUIRE(packet_v2::header(set_mode)->len == 6);
00117        REQUIRE(packet_v2::header(mission_set_current)->len == 2);
00118        REQUIRE(packet_v2::header(encapsulated_data)->len == 255);
00119        REQUIRE(packet_v2::header(param_ext_request_list)->len == 2);
00120    }
00121    SECTION("Header has incompatibility flags.")
00122    {
00123        REQUIRE((packet_v2::header(heartbeat)->incompat_flags &
00124                  MAVLINK_IFLAG_SIGNED) == true);
00125        REQUIRE((packet_v2::header(ping)->incompat_flags &
00126                  MAVLINK_IFLAG_SIGNED) == false);
00127        REQUIRE((packet_v2::header(set_mode)->incompat_flags &
00128                  MAVLINK_IFLAG_SIGNED) == true);
00129        REQUIRE((packet_v2::header(mission_set_current)->incompat_flags &
00130                  MAVLINK_IFLAG_SIGNED) == false);
00131        REQUIRE((packet_v2::header(encapsulated_data)->incompat_flags &
00132                  MAVLINK_IFLAG_SIGNED) == true);
00133        REQUIRE((packet_v2::header(param_ext_request_list)->incompat_flags &
00134                  MAVLINK_IFLAG_SIGNED) == false);
00135    }
00136    SECTION("Header has compatibility flags.")
00137    {
00138        REQUIRE(packet_v2::header(heartbeat)->compat_flags == 0);
00139        REQUIRE(packet_v2::header(ping)->compat_flags == 0);
00140        REQUIRE(packet_v2::header(set_mode)->compat_flags == 0);
00141        REQUIRE(packet_v2::header(mission_set_current)->compat_flags == 0);
00142        REQUIRE(packet_v2::header(encapsulated_data)->compat_flags == 0);
00143        REQUIRE(packet_v2::header(param_ext_request_list)->compat_flags == 0);
00144    }
00145    SECTION("Header has a sequence number.")
00146    {
00147        REQUIRE(packet_v2::header(heartbeat)->seq == 0xFD);
00148        REQUIRE(packet_v2::header(ping)->seq == 0xFD);
00149        REQUIRE(packet_v2::header(set_mode)->seq == 0xFD);
00150        REQUIRE(packet_v2::header(mission_set_current)->seq == 0xFD);
00151        REQUIRE(packet_v2::header(encapsulated_data)->seq == 0xFD);
00152        REQUIRE(packet_v2::header(param_ext_request_list)->seq == 0xFD);
00153    }
00154    SECTION("Header has a system ID.")
00155    {
00156        REQUIRE(packet_v2::header(heartbeat)->sysid == 127);

```

```

00157     REQUIRE(packet_v2::header(ping)->sysid == 192);
00158     REQUIRE(packet_v2::header(set_mode)->sysid == 172);
00159     REQUIRE(packet_v2::header(mission_set_current)->sysid == 255);
00160     REQUIRE(packet_v2::header(encapsulated_data)->sysid == 224);
00161     REQUIRE(packet_v2::header(param_ext_request_list)->sysid == 0);
00162 }
00163 SECTION("Header has a component ID.")
00164 {
00165     REQUIRE(packet_v2::header(heartbeat)->compid == 1);
00166     REQUIRE(packet_v2::header(ping)->compid == 168);
00167     REQUIRE(packet_v2::header(set_mode)->compid == 0);
00168     REQUIRE(packet_v2::header(mission_set_current)->compid == 0);
00169     REQUIRE(packet_v2::header(encapsulated_data)->compid == 255);
00170     REQUIRE(packet_v2::header(param_ext_request_list)->compid == 255);
00171 }
00172 SECTION("Header has a message ID.")
00173 {
00174     REQUIRE(packet_v2::header(heartbeat)->msgid == 0);
00175     REQUIRE(packet_v2::header(ping)->msgid == 4);
00176     REQUIRE(packet_v2::header(set_mode)->msgid == 11);
00177     REQUIRE(packet_v2::header(mission_set_current)->msgid == 41);
00178     REQUIRE(packet_v2::header(encapsulated_data)->msgid == 131);
00179     REQUIRE(packet_v2::header(param_ext_request_list)->msgid == 321);
00180 }
00181 SECTION("Returns nullptr when an incomplete header is given.")
00182 {
00183     heartbeat.resize(5);
00184     ping.resize(4);
00185     set_mode.resize(3);
00186     mission_set_current.resize(2);
00187     encapsulated_data.resize(1);
00188     param_ext_request_list.resize(0);
00189     REQUIRE(packet_v2::header(heartbeat) == nullptr);
00190     REQUIRE(packet_v2::header(ping) == nullptr);
00191     REQUIRE(packet_v2::header(set_mode) == nullptr);
00192     REQUIRE(packet_v2::header(mission_set_current) == nullptr);
00193     REQUIRE(packet_v2::header(encapsulated_data) == nullptr);
00194     REQUIRE(packet_v2::header(param_ext_request_list) == nullptr);
00195 }
00196 SECTION("Returns nullptr when the magic byte is wrong.")
00197 {
00198     heartbeat.front() = 0xAD;
00199     ping.front() = 0xBC;
00200     set_mode.front() = 0xFE;
00201     mission_set_current.front() = 0xFE;
00202     encapsulated_data.front() = 0xFE;
00203     param_ext_request_list.front() = 0xFE;
00204     REQUIRE(packet_v2::header(heartbeat) == nullptr);
00205     REQUIRE(packet_v2::header(ping) == nullptr);
00206     REQUIRE(packet_v2::header(set_mode) == nullptr);
00207     REQUIRE(packet_v2::header(mission_set_current) == nullptr);
00208     REQUIRE(packet_v2::header(encapsulated_data) == nullptr);
00209     REQUIRE(packet_v2::header(param_ext_request_list) == nullptr);
00210 }
00211 }
00212
00213
00214 TEST_CASE("'packet_v2::packet_complete' determines whether the given bytes "
00215 "represent a complete packet.", "[packet_v2]")
00216 {
00217     auto heartbeat = to_vector_with_sig(HeartbeatV2());
00218     auto ping = to_vector(PingV2());
00219     auto set_mode = to_vector_with_sig(SetModeV2());
00220     auto mission_set_current = to_vector(MissionSetCurrentV2());
00221     auto encapsulated_data = to_vector_with_sig(EncapsulatedDataV2());
00222     auto param_ext_request_list = to_vector(ParamExtRequestListV2());
00223     SECTION("Returns true when a complete packet is given.")
00224 {
00225     REQUIRE(packet_v2::packet_complete(heartbeat));
00226     REQUIRE(packet_v2::packet_complete(ping));
00227     REQUIRE(packet_v2::packet_complete(set_mode));
00228     REQUIRE(packet_v2::packet_complete(mission_set_current));
00229     REQUIRE(packet_v2::packet_complete(encapsulated_data));
00230     REQUIRE(packet_v2::packet_complete(param_ext_request_list));
00231 }
00232 SECTION("Returns false when the magic byte is wrong.")
00233 {
00234     heartbeat.front() = 0xAD;
00235     ping.front() = 0xBC;
00236     set_mode.front() = 0xFE;
00237     mission_set_current.front() = 0xFE;

```

```

00238     encapsulated_data.front() = 0xFE;
00239     param_ext_request_list.front() = 0xFE;
00240     REQUIRE_FALSE(packet_v2::packet_complete(
00241         heartbeat));
00241     REQUIRE_FALSE(packet_v2::packet_complete(ping));
00242     REQUIRE_FALSE(packet_v2::packet_complete(set_mode));
00243     REQUIRE_FALSE(packet_v2::packet_complete(mission_set_current));
00244     REQUIRE_FALSE(packet_v2::packet_complete(encapsulated_data));
00245     REQUIRE_FALSE(packet_v2::packet_complete(param_ext_request_list));
00246 }
00247 SECTION("Returns false when the packet is too short.")
00248 {
00249     heartbeat.pop_back();
00250     ping.pop_back();
00251     set_mode.pop_back();
00252     mission_set_current.pop_back();
00253     encapsulated_data.pop_back();
00254     param_ext_request_list.pop_back();
00255     REQUIRE_FALSE(packet_v2::packet_complete(
00256         heartbeat));
00256     REQUIRE_FALSE(packet_v2::packet_complete(ping));
00257     REQUIRE_FALSE(packet_v2::packet_complete(set_mode));
00258     REQUIRE_FALSE(packet_v2::packet_complete(mission_set_current));
00259     REQUIRE_FALSE(packet_v2::packet_complete(encapsulated_data));
00260     REQUIRE_FALSE(packet_v2::packet_complete(param_ext_request_list));
00261 }
00262 SECTION("Returns false when the packet is too long.")
00263 {
00264     heartbeat.push_back(0x00);
00265     ping.push_back(0x00);
00266     set_mode.push_back(0x00);
00267     mission_set_current.push_back(0x00);
00268     encapsulated_data.push_back(0x00);
00269     param_ext_request_list.push_back(0x00);
00270     REQUIRE_FALSE(packet_v2::packet_complete(
00271         heartbeat));
00271     REQUIRE_FALSE(packet_v2::packet_complete(ping));
00272     REQUIRE_FALSE(packet_v2::packet_complete(set_mode));
00273     REQUIRE_FALSE(packet_v2::packet_complete(encapsulated_data));
00274     REQUIRE_FALSE(packet_v2::packet_complete(encapsulated_data));
00275     REQUIRE_FALSE(packet_v2::packet_complete(param_ext_request_list));
00276 }
00277 }
00278
00279
00280 TEST_CASE("'packet_v2::is_signed' determines whether the given bytes "
00281             "represent a signed packet.", "[packet_v2]")
00282 {
00283     auto heartbeat = to_vector_with_sig(HeartbeatV2());
00284     auto ping = to_vector(PingV2());
00285     auto set_mode = to_vector_with_sig(SetModeV2());
00286     auto mission_set_current = to_vector(MissionSetCurrentV2());
00287     auto encapsulated_data = to_vector_with_sig(EncapsulatedDataV2());
00288     auto param_ext_request_list = to_vector(ParamExtRequestListV2());
00289     SECTION("Returns true when the packet is signed.")
00290 {
00291     REQUIRE(packet_v2::is_signed(heartbeat));
00292     REQUIRE(packet_v2::is_signed(set_mode));
00293     REQUIRE(packet_v2::is_signed(encapsulated_data));
00294 }
00295 SECTION("Returns false when the packet is not signed.")
00296 {
00297     REQUIRE_FALSE(packet_v2::is_signed(ping));
00298     REQUIRE_FALSE(packet_v2::is_signed(mission_set_current));
00299     REQUIRE_FALSE(packet_v2::is_signed(param_ext_request_list));
00300 }
00301 SECTION("Throws an error when the header is invalid.")
00302 {
00303     heartbeat.front() = 0xAD;
00304     ping.front() = 0xBC;
00305     set_mode.front() = 0xFE;
00306     mission_set_current.resize(2);
00307     encapsulated_data.resize(1);
00308     param_ext_request_list.resize(0);
00309     // Errors
00310     REQUIRE_THROWS_AS(
00311         packet_v2::is_signed(heartbeat), std::invalid_argument);
00312     REQUIRE_THROWS_AS(
00313         packet_v2::is_signed(ping), std::invalid_argument);
00314     REQUIRE_THROWS_AS(
00315         packet_v2::is_signed(set_mode), std::invalid_argument);

```

```

00316     REQUIRE_THROWS_AS(
00317         packet_v2::is_signed(mission_set_current), std::invalid_argument);
00318     REQUIRE_THROWS_AS(
00319         packet_v2::is_signed(encapsulated_data), std::invalid_argument);
00320     REQUIRE_THROWS_AS(
00321         packet_v2::is_signed(param_ext_request_list),
00322         std::invalid_argument);
00323     // Error messages.
00324     REQUIRE_THROWS_WITH(
00325         packet_v2::is_signed(heartbeat),
00326         "Header is incomplete or invalid.");
00327     REQUIRE_THROWS_WITH(
00328         packet_v2::is_signed(ping),
00329         "Header is incomplete or invalid.");
00330     REQUIRE_THROWS_WITH(
00331         packet_v2::is_signed(set_mode),
00332         "Header is incomplete or invalid.");
00333     REQUIRE_THROWS_WITH(
00334         packet_v2::is_signed(mission_set_current),
00335         "Header is incomplete or invalid.");
00336     REQUIRE_THROWS_WITH(
00337         packet_v2::is_signed(encapsulated_data),
00338         "Header is incomplete or invalid.");
00339     REQUIRE_THROWS_WITH(
00340         packet_v2::is_signed(param_ext_request_list),
00341         "Header is incomplete or invalid.");
00342 }
00343 }
00344
00345
00346 TEST_CASE("packet_v2::Packet's can be constructed.", "[packet_v2::Packet]")
00347 {
00348     HeartbeatV2 heartbeat;
00349     PingV2 ping;
00350     SetModeV2 set_mode;
00351     MissionSetCurrentV2 mission_set_current;
00352     EncapsulatedDataV2 encapsulated_data;
00353     ParamExtRequestListV2 param_ext_request_list;
00354     SECTION("With proper arguments and no signature.")
00355     {
00356         REQUIRE_NO_THROW(packet_v2::Packet(to_vector(heartbeat)));
00357         REQUIRE_NO_THROW(packet_v2::Packet(to_vector(ping)));
00358         REQUIRE_NO_THROW(packet_v2::Packet(to_vector(set_mode)));
00359         REQUIRE_NO_THROW(packet_v2::Packet(to_vector(mission_set_current)));
00360         REQUIRE_NO_THROW(packet_v2::Packet(to_vector(encapsulated_data)));
00361         REQUIRE_NO_THROW(packet_v2::Packet(to_vector(param_ext_request_list)));
00362     }
00363     SECTION("With proper arguments and a signature.")
00364     {
00365         REQUIRE_NO_THROW(packet_v2::Packet(to_vector_with_sig(
00366             heartbeat)));
00366         REQUIRE_NO_THROW(packet_v2::Packet(to_vector_with_sig(
00367             ping)));
00368         REQUIRE_NO_THROW(packet_v2::Packet(to_vector_with_sig(set_mode)));
00369         REQUIRE_NO_THROW(
00370             packet_v2::Packet(to_vector_with_sig(mission_set_current)));
00371         REQUIRE_NO_THROW(
00372             packet_v2::Packet(to_vector_with_sig(encapsulated_data)));
00373         REQUIRE_NO_THROW(
00374             packet_v2::Packet(to_vector_with_sig(param_ext_request_list)));
00375     }
00376     SECTION("And ensures a complete header is given.")
00377     {
00378         // 0 length packet.
00379         REQUIRE_THROWS_AS(packet_v2::Packet({}), std::length_error);
00380         REQUIRE_THROWS_WITH(packet_v2::Packet({}), "Packet is empty.");
00381         // Packet short 1 byte.
00382         REQUIRE_THROWS_AS(
00383             packet_v2::Packet({0xFD, 2, 3, 4, 5, 6, 7, 8, 9}),
00384             std::length_error);
00385         REQUIRE_THROWS_WITH(
00386             packet_v2::Packet({0xFD, 2, 3, 4, 5, 6, 7, 8, 9}),
00387             "Packet (9 bytes) is shorter than a v2.0 header (10 bytes).");
00388     }
00389     SECTION("And ensures packets begin with the magic byte (0xFD).")
00390     {
00391         ping.magic = 0xAD;
00392         REQUIRE_THROWS_AS(
00393             packet_v2::Packet(to_vector(ping)), std::invalid_argument);
00394         REQUIRE_THROWS_WITH(
00395             packet_v2::Packet(to_vector(ping)),

```

```

00395         "Invalid packet starting byte (0xAD), "
00396         "v2.0 packets should start with 0xFD.");
00397     }
00398     SECTION("And ensures the message ID is valid.")
00399     {
00400         ping.msgid = 255; // ID 255 is not currently valid.
00401         REQUIRE_THROWS_AS(
00402             packet_v2::Packet(to_vector(ping)),
00403             InvalidPacketIDError);
00404         REQUIRE_THROWS_WITH(
00405             packet_v2::Packet(to_vector(ping)),
00406             "Packet ID (#255) is not part of the ''"
00407             MAVLINK_DIALECT "' MAVLink dialect.");
00408         ping.msgid = 5000; // ID 5000 is not currently valid.
00409         REQUIRE_THROWS_AS(
00410             packet_v2::Packet(to_vector(ping)),
00411             InvalidPacketIDError);
00412         REQUIRE_THROWS_WITH(
00413             packet_v2::Packet(to_vector(ping)),
00414             "Packet ID (#5000) is not part of the ''"
00415             MAVLINK_DIALECT "' MAVLink dialect.");
00416     }
00417     SECTION("And ensures the packet is the correct length (without signature).")
00418     {
00419         // HEARTBEAT (no target system/component).
00420         auto heartbeat_data = to_vector(heartbeat);
00421         heartbeat_data.push_back(0x00);
00422         REQUIRE_THROWS_AS(packet_v2::Packet(heartbeat_data), std::length_error);
00423         REQUIRE_THROWS_WITH(
00424             packet_v2::Packet(heartbeat_data),
00425             "Packet is 22 bytes, should be 21 bytes.");
00426         // PING (with target system/component).
00427         auto ping_data = to_vector(ping);
00428         ping_data.pop_back();
00429         REQUIRE_THROWS_AS(packet_v2::Packet(ping_data), std::length_error);
00430         REQUIRE_THROWS_WITH(
00431             packet_v2::Packet(ping_data),
00432             "Packet is 25 bytes, should be 26 bytes.");
00433         // SET_MODE (target system only, no target component).
00434         auto set_mode_data = to_vector(set_mode);
00435         set_mode_data.push_back(0x00);
00436         REQUIRE_THROWS_AS(packet_v2::Packet(set_mode_data), std::length_error);
00437         REQUIRE_THROWS_WITH(
00438             packet_v2::Packet(set_mode_data),
00439             "Packet is 19 bytes, should be 18 bytes.");
00440         // PARAM_REQUEST_READ (trimmed out, 0.0, system/component).
00441         auto mission_set_current_data = to_vector(mission_set_current);
00442         mission_set_current_data.pop_back();
00443         REQUIRE_THROWS_AS(
00444             packet_v2::Packet(mission_set_current_data), std::length_error);
00445         REQUIRE_THROWS_WITH(
00446             packet_v2::Packet(mission_set_current_data),
00447             "Packet is 13 bytes, should be 14 bytes.");
00448         // ENCAPSULATED_DATA (longest packet).
00449         auto encapsulated_data_data = to_vector(encapsulated_data);
00450         encapsulated_data_data.push_back(0x00);
00451         REQUIRE_THROWS_AS(
00452             packet_v2::Packet(encapsulated_data_data), std::length_error);
00453         REQUIRE_THROWS_WITH(
00454             packet_v2::Packet(encapsulated_data_data),
00455             "Packet is 268 bytes, should be 267 bytes.");
00456         // PARAM_EXT_REQUEST_LIST (message ID beyond 255).
00457         auto param_ext_request_list_data = to_vector(param_ext_request_list);
00458         param_ext_request_list_data.pop_back();
00459         REQUIRE_THROWS_AS(
00460             packet_v2::Packet(param_ext_request_list_data), std::length_error);
00461         REQUIRE_THROWS_WITH(
00462             packet_v2::Packet(param_ext_request_list_data),
00463             "Packet is 13 bytes, should be 14 bytes.");
00464     }
00465     SECTION("And ensures the packet is the correct length (with signature).")
00466     {
00467         // HEARTBEAT (no target system/component).
00468         auto heartbeat_data = to_vector_with_sig(heartbeat);
00469         heartbeat_data.push_back(0x00);
00470         REQUIRE_THROWS_AS(packet_v2::Packet(heartbeat_data), std::length_error);
00471         REQUIRE_THROWS_WITH(
00472             packet_v2::Packet(heartbeat_data),
00473             "Signed packet is 35 bytes, should be 34 bytes.");
00474         // PING (with target system/component).
00475         auto ping_data = to_vector_with_sig(ping);

```

```

00474     ping_data.pop_back();
00475     REQUIRE_THROWS_AS(packet_v2::Packet(ping_data), std::length_error);
00476     REQUIRE_THROWS_WITH(
00477         packet_v2::Packet(ping_data),
00478         "Signed packet is 38 bytes, should be 39 bytes.");
00479     // SET_MODE (target system only, no target component).
00480     auto set_mode_data = to_vector_with_sig(set_mode);
00481     set_mode_data.push_back(0x00);
00482     REQUIRE_THROWS_AS(packet_v2::Packet(set_mode_data), std::length_error);
00483     REQUIRE_THROWS_WITH(
00484         packet_v2::Packet(set_mode_data),
00485         "Signed packet is 32 bytes, should be 31 bytes.");
00486     // PARAM_REQUEST_READ (trimmed out, 0.0, system/component).
00487     auto mission_set_current_data = to_vector_with_sig(mission_set_current);
00488     mission_set_current_data.pop_back();
00489     REQUIRE_THROWS_AS(
00490         packet_v2::Packet(mission_set_current_data), std::length_error);
00491     REQUIRE_THROWS_WITH(
00492         packet_v2::Packet(mission_set_current_data),
00493         "Signed packet is 26 bytes, should be 27 bytes.");
00494     // ENCAPSULATED_DATA (longest packet).
00495     auto encapsulated_data_data = to_vector_with_sig(encapsulated_data);
00496     encapsulated_data_data.push_back(0x00);
00497     REQUIRE_THROWS_AS(
00498         packet_v2::Packet(encapsulated_data_data), std::length_error);
00499     REQUIRE_THROWS_WITH(
00500         packet_v2::Packet(encapsulated_data_data),
00501         "Signed packet is 281 bytes, should be 280 bytes.");
00502     // PARAM_EXT_REQUEST_LIST (message ID beyond 255).
00503     auto param_ext_request_list_data =
00504         to_vector_with_sig(param_ext_request_list);
00505     param_ext_request_list_data.pop_back();
00506     REQUIRE_THROWS_AS(
00507         packet_v2::Packet(param_ext_request_list_data), std::length_error);
00508     REQUIRE_THROWS_WITH(
00509         packet_v2::Packet(param_ext_request_list_data),
00510         "Signed packet is 26 bytes, should be 27 bytes.");
00511 }
00512 }
00513
00514
00515 TEST_CASE("packet_v2::Packet's are comparable.", "[packet_v2::Packet]")
00516 {
00517     SECTION("with ==")
00518     {
00519         REQUIRE(
00520             packet_v2::Packet(to_vector(PingV2())) ==
00521             packet_v2::Packet(to_vector(PingV2())));
00522         REQUIRE_FALSE(
00523             packet_v2::Packet(to_vector(PingV2())) ==
00524             packet_v2::Packet(to_vector(SetModeV2())));
00525         REQUIRE_FALSE(
00526             packet_v2::Packet(to_vector(PingV2())) ==
00527             packet_v2::Packet(to_vector_with_sig(PingV2())));
00528     }
00529     SECTION("with !=")
00530     {
00531         REQUIRE(
00532             packet_v2::Packet(to_vector(PingV2())) !=
00533             packet_v2::Packet(to_vector(SetModeV2())));
00534         REQUIRE(
00535             packet_v2::Packet(to_vector(PingV2())) !=
00536             packet_v2::Packet(to_vector_with_sig(PingV2())));
00537         REQUIRE_FALSE(
00538             packet_v2::Packet(to_vector(PingV2())) !=
00539             packet_v2::Packet(to_vector(PingV2())));
00540     }
00541 }
00542
00543
00544 TEST_CASE("packet_v2::Packet's are copyable.", "[packet_v2::Packet]")
00545 {
00546     packet_v2::Packet original(to_vector(PingV2()));
00547     packet_v2::Packet copy(original);
00548     REQUIRE(copy == packet_v2::Packet(to_vector(PingV2())));
00549 }
00550
00551
00552 TEST_CASE("packet_v2::Packet's are movable.", "[packet_v2::Packet]")
00553 {
00554     packet_v2::Packet original(to_vector(PingV2()));

```

```

00555     packet_v2::Packet moved(std::move(original));
00556     REQUIRE(moved == packet_v2::Packet(to_vector(PingV2())));
00557 }
00558
00559
00560 TEST_CASE("packet_v2::Packet's are assignable.", "[Packet]")
00561 {
00562     packet_v2::Packet packet_a(to_vector(PingV2()));
00563     packet_v2::Packet packet_b(to_vector(SetModeV2()));
00564     REQUIRE(packet_a == packet_v2::Packet(to_vector(PingV2())));
00565     packet_a = packet_b;
00566     REQUIRE(packet_b == packet_v2::Packet(to_vector(SetModeV2())));
00567 }
00568
00569
00570 TEST_CASE("packet_v2::Packet's are assignable (by move semantics).", "[Packet]")
00571 {
00572     packet_v2::Packet packet_a(to_vector(PingV2()));
00573     packet_v2::Packet packet_b(to_vector(SetModeV2()));
00574     REQUIRE(packet_a == packet_v2::Packet(to_vector(PingV2())));
00575     packet_a = packet_b;
00576     REQUIRE(packet_b == packet_v2::Packet(to_vector(SetModeV2())));
00577 }
00578
00579
00580 TEST_CASE("packet_v2::Packet's contain raw packet data and make it accessible.",
00581         "[packet_v2::Packet]")
00582 {
00583     auto heartbeat = to_vector_with_sig(HeartbeatV2());
00584     auto ping = to_vector(PingV2());
00585     auto set_mode = to_vector_with_sig(SetModeV2());
00586     auto mission_set_current = to_vector(MissionSetCurrentV2());
00587     auto encapsulated_data = to_vector_with_sig(EncapsulatedDataV2());
00588     auto param_ext_request_list = to_vector_with_sig(ParamExtRequestListV2());
00589     REQUIRE(packet_v2::Packet(heartbeat).data() ==
00590             heartbeat);
00591     REQUIRE(packet_v2::Packet(ping).data() == ping);
00592     REQUIRE(packet_v2::Packet(set_mode).data() == set_mode);
00593     REQUIRE(
00594         packet_v2::Packet(mission_set_current).data() == mission_set_current);
00595     REQUIRE(
00596         packet_v2::Packet(encapsulated_data).data() == encapsulated_data);
00597     REQUIRE(
00598         packet_v2::Packet(param_ext_request_list).data() ==
00599             param_ext_request_list);
00600
00601
00602 TEST_CASE("packet_v2::Packet's have a version.", "[packet_v2::Packet]")
00603 {
00604     auto heartbeat = to_vector_with_sig(HeartbeatV2());
00605     auto ping = to_vector(PingV2());
00606     auto set_mode = to_vector_with_sig(SetModeV2());
00607     auto mission_set_current = to_vector(MissionSetCurrentV2());
00608     auto encapsulated_data = to_vector_with_sig(EncapsulatedDataV2());
00609     auto param_ext_request_list = to_vector(ParamExtRequestListV2());
00610     // All should read Packet::V2 (0x0200) for v2.0.
00611     REQUIRE(packet_v2::Packet(heartbeat).version() ==
00612             Packet::V2);
00613     REQUIRE(packet_v2::Packet(ping).version() == Packet::V2);
00614     REQUIRE(packet_v2::Packet(set_mode).version() == Packet::V2);
00615     REQUIRE(packet_v2::Packet(mission_set_current).version() == 0x200);
00616     REQUIRE(packet_v2::Packet(encapsulated_data).version() == 0x200);
00617     REQUIRE(packet_v2::Packet(param_ext_request_list).version() == 0x200);
00618
00619
00620 TEST_CASE("packet_v2::Packet's have an ID.", "[packet_v2::Packet]")
00621 {
00622     auto heartbeat = to_vector_with_sig(HeartbeatV2());
00623     auto ping = to_vector(PingV2());
00624     auto set_mode = to_vector_with_sig(SetModeV2());
00625     auto mission_set_current = to_vector(MissionSetCurrentV2());
00626     auto encapsulated_data = to_vector_with_sig(EncapsulatedDataV2());
00627     auto param_ext_request_list = to_vector(ParamExtRequestListV2());
00628     REQUIRE(packet_v2::Packet(heartbeat).id() == 0);
00629     REQUIRE(packet_v2::Packet(ping).id() == 4);
00630     REQUIRE(packet_v2::Packet(set_mode).id() == 11);
00631     REQUIRE(packet_v2::Packet(mission_set_current).id() == 41);
00632     REQUIRE(packet_v2::Packet(encapsulated_data).id() == 131);
00633     REQUIRE(packet_v2::Packet(param_ext_request_list).id() == 321);

```

```

00634 }
00635
00636
00637 TEST_CASE("packet_v2::Packet's have a name.", "[packet_v2::Packet]")
00638 {
00639     auto heartbeat = to_vector_with_sig(HeartbeatV2());
00640     auto ping = to_vector(PingV2());
00641     auto set_mode = to_vector_with_sig(SetModeV2());
00642     auto mission_set_current = to_vector(MissionSetCurrentV2());
00643     auto encapsulated_data = to_vector_with_sig(EncapsulatedDataV2());
00644     auto param_ext_request_list = to_vector(ParamExtRequestListV2());
00645     REQUIRE(packet_v2::Packet(heartbeat).name() == "HEARTBEAT");
00646     REQUIRE(packet_v2::Packet(ping).name() == "PING");
00647     REQUIRE(packet_v2::Packet(set_mode).name() == "SET_MODE");
00648     REQUIRE(
00649         packet_v2::Packet(mission_set_current).name() == "MISSION_SET_CURRENT");
00650     REQUIRE(packet_v2::Packet(encapsulated_data).name() == "ENCAPSULATED_DATA");
00651     REQUIRE(
00652         packet_v2::Packet(param_ext_request_list).name() ==
00653         "PARAM_EXT_REQUEST_LIST");
00654 }
00655
00656
00657 TEST_CASE("packet_v2::Packet's have a source address.", "[packet_v2::Packet]")
00658 {
00659     auto heartbeat = to_vector_with_sig(HeartbeatV2());
00660     auto ping = to_vector(PingV2());
00661     auto set_mode = to_vector_with_sig(SetModeV2());
00662     auto mission_set_current = to_vector(MissionSetCurrentV2());
00663     auto encapsulated_data = to_vector_with_sig(EncapsulatedDataV2());
00664     auto param_ext_request_list = to_vector(ParamExtRequestListV2());
00665     REQUIRE(packet_v2::Packet(heartbeat).source() ==
00666         MAVAddress("127.1"));
00667     REQUIRE(packet_v2::Packet(ping).source() == MAVAddress("192.168"));
00668     REQUIRE(packet_v2::Packet(set_mode).source() == MAVAddress("172.0"));
00669     REQUIRE(
00670         packet_v2::Packet(mission_set_current).source() ==
00671         MAVAddress("255.0"));
00672     REQUIRE(
00673         packet_v2::Packet(encapsulated_data).source() ==
00674         MAVAddress("224.255"));
00675 }
00676
00677
00678 TEST_CASE("packet_v2::Packet's optionally have a destination address.",
00679     "[packet_v2::Packet]")
00680 {
00681     auto heartbeat = to_vector_with_sig(HeartbeatV2());
00682     auto ping = to_vector(PingV2());
00683     auto set_mode = to_vector_with_sig(SetModeV2());
00684     auto mission_set_current = to_vector(MissionSetCurrentV2());
00685     auto encapsulated_data = to_vector_with_sig(EncapsulatedDataV2());
00686     auto param_ext_request_list = to_vector(ParamExtRequestListV2());
00687     REQUIRE_THROWS_AS(
00688         packet_v2::Packet(heartbeat).dest().value(), std::bad_optional_access);
00689     REQUIRE(packet_v2::Packet(ping).dest().value() ==
00690         MAVAddress("127.1"));
00691     REQUIRE(packet_v2::Packet(set_mode).dest().value() ==
00692         MAVAddress("123.0"));
00693     REQUIRE(
00694         packet_v2::Packet(mission_set_current).dest().value() ==
00695         MAVAddress("0.0"));
00696     REQUIRE_THROWS_AS(
00697         packet_v2::Packet(encapsulated_data).dest().value(),
00698         std::bad_optional_access);
00699     REQUIRE(
00700         packet_v2::Packet(param_ext_request_list).dest().value() ==
00701         MAVAddress("32.64"));
00702
00703 TEST_CASE("packet_v2::Packet's optionally have a source connection.",
00704     "[packet_v2::Packet]")
00705 {
00706     auto heartbeat = packet_v2::Packet(to_vector(HeartbeatV2()));
00707     SECTION("Defaults to nullptr.")
00708     {
00709         REQUIRE(heartbeat.connection() == nullptr);

```

```

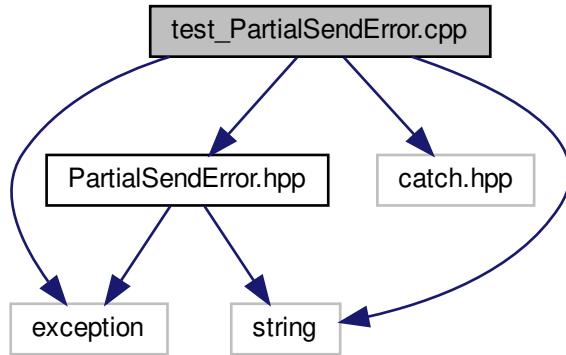
00710     }
00711     SECTION("Can be set with the 'connection' method.")
00712     {
00713         fakeit::Mock<Filter> mock_filter;
00714         auto filter = mock_shared(mock_filter);
00715         auto conn = std::make_shared<Connection>("SOURCE", filter);
00716         heartbeat.connection(conn);
00717         REQUIRE(heartbeat.connection() != nullptr);
00718         REQUIRE(heartbeat.connection() == conn);
00719         REQUIRE(str(*heartbeat.connection()) == "SOURCE");
00720     }
00721 }
00722
00723
00724 TEST_CASE("packet_v2::Packet's are printable.", "[packet_v2::Packet]")
00725 {
00726     auto heartbeat = to_vector_with_sig(HeartbeatV2());
00727     auto ping = to_vector(PingV2());
00728     auto set_mode = to_vector_with_sig(SetModeV2());
00729     auto mission_set_current = to_vector(MissionSetCurrentV2());
00730     auto encapsulated_data = to_vector_with_sig(EncapsulatedDataV2());
00731     auto param_ext_request_list = to_vector(ParamExtRequestListV2());
00732     REQUIRE(
00733         str(packet_v2::Packet(heartbeat)) ==
00734         "HEARTBEAT (#0) from 127.1 (v2.0)");
00735     REQUIRE(
00736         str(packet_v2::Packet(ping)) ==
00737         "PING (#4) from 192.168 to 127.1 (v2.0)");
00738     REQUIRE(
00739         str(packet_v2::Packet(set_mode)) ==
00740         "SET_MODE (#11) from 172.0 to 123.0 (v2.0)");
00741     REQUIRE(
00742         str(packet_v2::Packet(mission_set_current)) ==
00743         "MISSION_SET_CURRENT (#41) from 255.0 to 0.0 (v2.0)");
00744     REQUIRE(
00745         str(packet_v2::Packet(encapsulated_data)) ==
00746         "ENCAPSULATED_DATA (#131) from 224.255 (v2.0)");
00747     REQUIRE(
00748         str(packet_v2::Packet(param_ext_request_list)) ==
00749         "PARAM_EXT_REQUEST_LIST (#321) from 0.255 to 32.64 (v2.0)");
00750 }

```

16.245 test_PartialSendError.cpp File Reference

```
#include <exception>
#include <string>
#include <catch.hpp>
#include "PartialSendError.hpp"
```

Include dependency graph for test_PartialSendError.cpp:



Functions

- `TEST_CASE ("PartialSendError's can be thrown.", "[PartialSendError]")`
- `TEST_CASE ("PartialSendError's have a message.", "[PartialSendError]")`

16.245.1 Function Documentation

16.245.1.1 TEST_CASE() [1/2]

```
TEST_CASE (
    "PartialSendError's can be thrown." ,
    "" [PartialSendError] )
```

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

16.245.1.2 TEST_CASE() [2/2]

```
TEST_CASE (
    "PartialSendError's have a message." ,
    "" [PartialSendError] )
```

Definition at line 50 of file [test_PartialSendError.cpp](#).

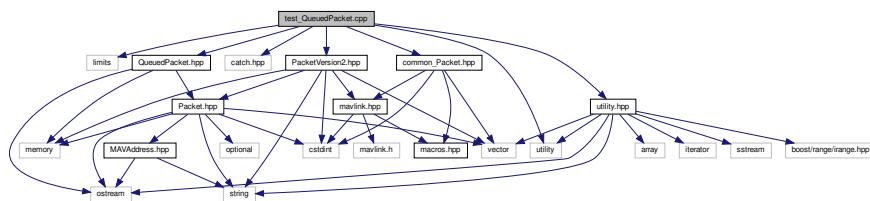
16.246 test_PartialSendError.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <exception>
00019 #include <string>
00020
00021 #include <catch.hpp>
00022
00023 #include "PartialSendError.hpp"
00024
00025
00026 [[noreturn]] static void throw_partial_send_error(
00027     unsigned long bytes_sent, unsigned long total_bytes)
00028 {
00029     throw PartialSendError(bytes_sent, total_bytes);
00030 }
00031
00032
00033 TEST_CASE("PartialSendError's can be thrown.", "[PartialSendError]")
00034 {
00035     SECTION("And can be caught.")
00036     {
00037         REQUIRE_THROWS(throw_partial_send_error(50, 100));
00038     }
00039     SECTION("And can be caught as PartialSendError.")
00040     {
00041         REQUIRE_THROWS_AS(throw_partial_send_error(50, 100), PartialSendError);
00042     }
00043     SECTION("And can be caught as std::exception.")
00044     {
00045         REQUIRE_THROWS_AS(throw_partial_send_error(50, 100), std::exception);
00046     }
00047 }
00048
00049
00050 TEST_CASE("PartialSendError's have a message.", "[PartialSendError]")
00051 {
00052     REQUIRE_THROWS_WITH(
00053         throw_partial_send_error(10, 100), "Could only write 10 of 100 bytes.");
00054     REQUIRE_THROWS_WITH(
00055         throw_partial_send_error(20, 100), "Could only write 20 of 100 bytes.");
00056     REQUIRE_THROWS_WITH(
00057         throw_partial_send_error(30, 100), "Could only write 30 of 100 bytes.");
00058     REQUIRE_THROWS_WITH(
00059         throw_partial_send_error(40, 100), "Could only write 40 of 100 bytes.");
00060     REQUIRE_THROWS_WITH(
00061         throw_partial_send_error(50, 100), "Could only write 50 of 100 bytes.");
00062     REQUIRE_THROWS_WITH(
00063         throw_partial_send_error(60, 100), "Could only write 60 of 100 bytes.");
00064     REQUIRE_THROWS_WITH(
00065         throw_partial_send_error(70, 100), "Could only write 70 of 100 bytes.");
00066     REQUIRE_THROWS_WITH(
00067         throw_partial_send_error(80, 100), "Could only write 80 of 100 bytes.");
00068     REQUIRE_THROWS_WITH(
00069         throw_partial_send_error(90, 100), "Could only write 90 of 100 bytes.");
00070 }
00071
00072
00073 // Required for complete function coverage.
00074 TEST_CASE("Run dynamic destructors (PartialSendError).", "[PartialSendError]")
00075 {
00076     PartialSendError *partial_send = nullptr;
00077     REQUIRE_NO_THROW(partial_send = new PartialSendError(50, 100));
```

```
00078     REQUIRE_NO_THROW(delete partial_send);
00079 }
```

16.247 test_QueuedPacket.cpp File Reference

```
#include <limits>
#include <utility>
#include <catch.hpp>
#include "PacketVersion2.hpp"
#include "QueuedPacket.hpp"
#include "utility.hpp"
#include "common_Packet.hpp"
Include dependency graph for test_QueuedPacket.cpp:
```



Functions

- [TEST_CASE \("QueuedPacket's can be constructed.", "\[QueuedPacket\]"\)](#)
- [TEST_CASE \("QueuedPacket' are comparable.", "\[QueuedPacket\]"\)](#)
- [TEST_CASE \("QueuedPacket's are copyable.", "\[QueuedPacket\]"\)](#)
- [TEST_CASE \("QueuedPacket's are movable.", "\[QueuedPacket\]"\)](#)
- [TEST_CASE \("QueuedPacket's are assignable.", "\[QueuedPacket\]"\)](#)
- [TEST_CASE \("QueuedPacket's are assignable \(by move semantics\).", "\[QueuedPacket\]"\)](#)
- [TEST_CASE \("QueuedPacket's 'packet' method returns the contained MAVLink packet", "\[QueuedPacket\]"\)](#)
- [TEST_CASE \("QueuedPacket's are printable.", "\[QueuedPacket\]"\)](#)

16.247.1 Function Documentation

16.247.1.1 TEST_CASE() [1/8]

```
TEST_CASE (
    "QueuedPacket's can be constructed." ,
    "" [QueuedPacket] )
```

Definition at line 30 of file [test_QueuedPacket.cpp](#).

16.247.1.2 TEST_CASE() [2/8]

```
TEST_CASE (
    "QueuedPacket's are comparable." ,
    "" [QueuedPacket] )
```

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

16.247.1.3 TEST_CASE() [3/8]

```
TEST_CASE (
    "QueuedPacket's are copyable." ,
    "" [QueuedPacket] )
```

Definition at line 167 of file [test_QueuedPacket.cpp](#).

16.247.1.4 TEST_CASE() [4/8]

```
TEST_CASE (
    "QueuedPacket's are movable." ,
    "" [QueuedPacket] )
```

Definition at line 176 of file [test_QueuedPacket.cpp](#).

16.247.1.5 TEST_CASE() [5/8]

```
TEST_CASE (
    "QueuedPacket's are assignable." ,
    "" [QueuedPacket] )
```

Definition at line 185 of file [test_QueuedPacket.cpp](#).

16.247.1.6 TEST_CASE() [6/8]

```
TEST_CASE (
    "QueuedPacket's are assignable (by move semantics)." ,
    "" [QueuedPacket] )
```

Definition at line 196 of file [test_QueuedPacket.cpp](#).

16.247.1.7 TEST_CASE() [7/8]

```
TEST_CASE (
    "QueuedPacket's 'packet' method returns the contained MAVLink packet" ,
    "" [QueuedPacket] )
```

Definition at line 208 of file [test_QueuedPacket.cpp](#).

16.247.1.8 TEST_CASE() [8/8]

```
TEST_CASE (
    "QueuedPacket's are printable." ,
    "" [QueuedPacket] )
```

Definition at line 216 of file [test_QueuedPacket.cpp](#).

16.248 test_QueuedPacket.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <limits>
00019 #include <utility>
00020
00021 #include <catch.hpp>
00022
00023 #include "PacketVersion2.hpp"
00024 #include "QueuedPacket.hpp"
00025 #include "utility.hpp"
00026
00027 #include "common_Packet.hpp"
00028
00029
00030 TEST_CASE("QueuedPacket's can be constructed.", "[QueuedPacket]")
00031 {
00032     auto packet = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00033     REQUIRE_NOTHROW(QueuedPacket(packet, 3, 10));
00034     REQUIRE_THROWS_AS(QueuedPacket(nullptr, 3, 10), std::invalid_argument);
00035     REQUIRE_THROWS_WITH(
00036         QueuedPacket(nullptr, 3, 10), "Given packet pointer is null.");
00037 }
00038
00039
00040 TEST_CASE("QueuedPacket' are comparable.", "[QueuedPacket]")
00041 {
00042     auto packet = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00043     SECTION("with ==")
00044     {
```

```
00045     REQUIRE(QueuedPacket(packet, 3, 10) == QueuedPacket(packet, 3, 10));
00046     REQUIRE_FALSE(
00047         QueuedPacket(packet, 3, 10) == QueuedPacket(packet, 0, 10));
00048     REQUIRE_FALSE(
00049         QueuedPacket(packet, 3, 10) == QueuedPacket(packet, 3, 100));
00050 }
00051 SECTION("with !=")
00052 {
00053     REQUIRE(QueuedPacket(packet, 3, 10) != QueuedPacket(packet, 0, 10));
00054     REQUIRE(QueuedPacket(packet, 3, 10) != QueuedPacket(packet, 3, 100));
00055     REQUIRE_FALSE(
00056         QueuedPacket(packet, 3, 10) != QueuedPacket(packet, 3, 10));
00057 }
00058 SECTION("with <")
00059 {
00060     // Priority takes precedence.
00061     REQUIRE(
00062         QueuedPacket(packet, 0, 10) < QueuedPacket(packet, 3, 100));
00063     REQUIRE_FALSE(
00064         QueuedPacket(packet, 3, 100) < QueuedPacket(packet, 0, 10));
00065     // Ticket number.
00066     REQUIRE(
00067         QueuedPacket(packet, 3, 100) < QueuedPacket(packet, 3, 10));
00068     REQUIRE_FALSE(
00069         QueuedPacket(packet, 3, 10) < QueuedPacket(packet, 3, 100));
00070     // Ticket number, with rollover.
00071     REQUIRE(
00072         QueuedPacket(packet, 3, 0) <
00073             QueuedPacket(
00074                 packet, 3,
00075                     std::numeric_limits<unsigned long long>::max() / 2 + 1));
00076     REQUIRE_FALSE(
00077         QueuedPacket(packet, 3, 0) <
00078             QueuedPacket(
00079                 packet, 3,
00080                     std::numeric_limits<unsigned long long>::max() / 2));
00081     // Not equal.
00082     REQUIRE_FALSE(
00083         QueuedPacket(packet, 3, 10) < QueuedPacket(packet, 3, 10));
00084 }
00085 SECTION("with >")
00086 {
00087     // Priority takes precedence.
00088     REQUIRE(
00089         QueuedPacket(packet, 3, 10) > QueuedPacket(packet, 0, 100));
00090     REQUIRE_FALSE(
00091         QueuedPacket(packet, 0, 100) > QueuedPacket(packet, 3, 10));
00092     // Ticket number.
00093     REQUIRE(
00094         QueuedPacket(packet, 3, 10) > QueuedPacket(packet, 3, 100));
00095     REQUIRE_FALSE(
00096         QueuedPacket(packet, 3, 100) > QueuedPacket(packet, 3, 10));
00097     // Ticket number, with rollover.
00098     REQUIRE(
00099         QueuedPacket(
00100             packet, 3,
00101                 std::numeric_limits<unsigned long long>::max() / 2 + 1) >
00102             QueuedPacket(packet, 3, 0));
00103     REQUIRE_FALSE(
00104         QueuedPacket(
00105             packet, 3,
00106                 std::numeric_limits<unsigned long long>::max() / 2) >
00107             QueuedPacket(packet, 3, 0));
00108     // Not equal.
00109     REQUIRE_FALSE(
00110         QueuedPacket(packet, 3, 10) > QueuedPacket(packet, 3, 10));
00111 }
00112 SECTION("with <=")
00113 {
00114     // Equality.
00115     REQUIRE(QueuedPacket(packet, 3, 10) <= QueuedPacket(packet, 3, 10));
00116     // Priority takes precedence.
00117     REQUIRE(
00118         QueuedPacket(packet, 0, 10) <= QueuedPacket(packet, 3, 100));
00119     REQUIRE_FALSE(
00120         QueuedPacket(packet, 3, 100) <= QueuedPacket(packet, 0, 10));
00121     // Ticket number.
00122     REQUIRE(
00123         QueuedPacket(packet, 3, 100) <= QueuedPacket(packet, 3, 10));
00124     REQUIRE_FALSE(
00125         QueuedPacket(packet, 3, 10) <= QueuedPacket(packet, 3, 100));
```

```

00126     // Ticket number, with rollover.
00127     REQUIRE(
00128         QueuedPacket(packet, 3, 0) <=
00129         QueuedPacket(
00130             packet, 3,
00131             std::numeric_limits<unsigned long long>::max() / 2 + 1));
00132     REQUIRE_FALSE(
00133         QueuedPacket(packet, 3, 0) <=
00134         QueuedPacket(
00135             packet, 3,
00136             std::numeric_limits<unsigned long long>::max() / 2));
00137 }
00138 SECTION("with >=")
00139 {
00140     // Equality.
00141     REQUIRE(QueuedPacket(packet, 3, 10) >= QueuedPacket(packet, 3, 10));
00142     // Priority takes precedence.
00143     REQUIRE(
00144         QueuedPacket(packet, 3, 10) >= QueuedPacket(packet, 0, 100));
00145     REQUIRE_FALSE(
00146         QueuedPacket(packet, 0, 100) >= QueuedPacket(packet, 3, 10));
00147     // Ticket number.
00148     REQUIRE(
00149         QueuedPacket(packet, 3, 10) >= QueuedPacket(packet, 3, 100));
00150     REQUIRE_FALSE(
00151         QueuedPacket(packet, 3, 100) >= QueuedPacket(packet, 3, 10));
00152     // Ticket number, with rollover.
00153     REQUIRE(
00154         QueuedPacket(
00155             packet, 3,
00156             std::numeric_limits<unsigned long long>::max() / 2 + 1) >=
00157         QueuedPacket(packet, 3, 0));
00158     REQUIRE_FALSE(
00159         QueuedPacket(
00160             packet, 3,
00161             std::numeric_limits<unsigned long long>::max() / 2) >=
00162         QueuedPacket(packet, 3, 0));
00163 }
00164 }
00165
00166
00167 TEST_CASE("QueuedPacket's are copyable.", "[QueuedPacket]")
00168 {
00169     auto packet = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00170     auto original = QueuedPacket(packet, 3, 10);
00171     auto copy(original);
00172     REQUIRE(copy == QueuedPacket(packet, 3, 10));
00173 }
00174
00175
00176 TEST_CASE("QueuedPacket's are movable.", "[QueuedPacket]")
00177 {
00178     auto packet = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00179     auto original = QueuedPacket(packet, 3, 10);
00180     auto moved(std::move(original));
00181     REQUIRE(moved == QueuedPacket(packet, 3, 10));
00182 }
00183
00184
00185 TEST_CASE("QueuedPacket's are assignable.", "[QueuedPacket]")
00186 {
00187     auto packet = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00188     auto a = QueuedPacket(packet, 3, 10);
00189     auto b = QueuedPacket(packet, 10, 3);
00190     REQUIRE(a == QueuedPacket(packet, 3, 10));
00191     a = b;
00192     REQUIRE(a == QueuedPacket(packet, 10, 3));
00193 }
00194
00195
00196 TEST_CASE("QueuedPacket's are assignable (by move semantics).",
00197             "[QueuedPacket]")
00198 {
00199     auto packet = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00200     auto a = QueuedPacket(packet, 3, 10);
00201     auto b = QueuedPacket(packet, 10, 3);
00202     REQUIRE(a == QueuedPacket(packet, 3, 10));
00203     a = std::move(b);
00204     REQUIRE(a == QueuedPacket(packet, 10, 3));
00205 }
00206

```

```

00207
00208 TEST_CASE("QueuedPacket's 'packet' method returns the contained MAVLink packet",
00209     "[QueuedPacket]")
00210 {
00211     auto packet = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00212     REQUIRE(*QueuedPacket(packet, 3, 10).packet() == *packet);
00213 }
00214
00215
00216 TEST_CASE("QueuedPacket's are printable.", "[QueuedPacket]")
00217 {
00218     auto packet = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00219     REQUIRE(
00220         str(QueuedPacket(packet, -10, 1)) ==
00221         "PING (#4) from 192.168 to 127.1 (v2.0) with priority -10");
00222     REQUIRE(
00223         str(QueuedPacket(packet, 0, 1)) ==
00224         "PING (#4) from 192.168 to 127.1 (v2.0) with priority 0");
00225     REQUIRE(
00226         str(QueuedPacket(packet, 3, 10)) ==
00227         "PING (#4) from 192.168 to 127.1 (v2.0) with priority 3");
00228     REQUIRE(
00229         str(QueuedPacket(packet, 10, 3)) ==
00230         "PING (#4) from 192.168 to 127.1 (v2.0) with priority 10");
00231 }

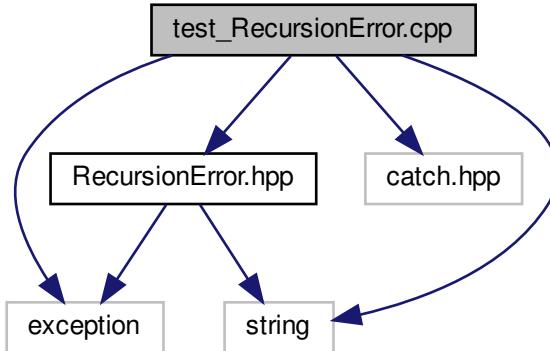
```

16.249 test_RecursionError.cpp File Reference

```

#include <exception>
#include <string>
#include <catch.hpp>
#include "RecursionError.hpp"
Include dependency graph for test_RecursionError.cpp:

```



Functions

- [TEST_CASE \("RecursionError's can be thrown.", "\[RecursionError\]"\)](#)
- [TEST_CASE \("RecursionError's have a message.", "\[RecursionError\]"\)](#)

16.249.1 Function Documentation

16.249.1.1 TEST_CASE() [1/2]

```
TEST_CASE (
    "RecursionError's can be thrown." ,
    "" [RecursionError] )
```

Definition at line 32 of file [test_RecursionError.cpp](#).

16.249.1.2 TEST_CASE() [2/2]

```
TEST_CASE (
    "RecursionError's have a message." ,
    "" [RecursionError] )
```

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

16.250 test_RecursionError.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <exception>
00019 #include <string>
00020
00021 #include <catch.hpp>
00022
00023 #include "RecursionError.hpp"
00024
00025
00026 [[noreturn]] static void throw_recursion_error(std::string message)
00027 {
00028     throw RecursionError(message);
00029 }
00030
00031
00032 TEST_CASE("RecursionError's can be thrown.", "[RecursionError]")
00033 {
00034     SECTION("And can be caught.")
00035     {
```

```

00036     REQUIRE_THROWS(throw_recursion_error("example"));
00037 }
00038 SECTION("And can be caught as DNSLookupError.")
00039 {
00040     REQUIRE_THROWS_AS(throw_recursion_error("example"), RecursionError);
00041 }
00042 SECTION("And can be caught as std::exception.")
00043 {
00044     REQUIRE_THROWS_AS(throw_recursion_error("example"), std::exception);
00045 }
00046 }
00047
00048
00049 TEST_CASE("RecursionError's have a message.", "[RecursionError]")
00050 {
00051     REQUIRE_THROWS_WITH(throw_recursion_error("example"), "example");
00052 }
00053
00054
00055 // Required for complete function coverage.
00056 TEST_CASE("Run dynamic destructors (RecursionError).", "[RecursionError]")
00057 {
00058     RecursionError *recursion = nullptr;
00059     REQUIRE_NOTHROW(recursion = new RecursionError("error"));
00060     REQUIRE_NOTHROW(delete recursion);
00061 }

```

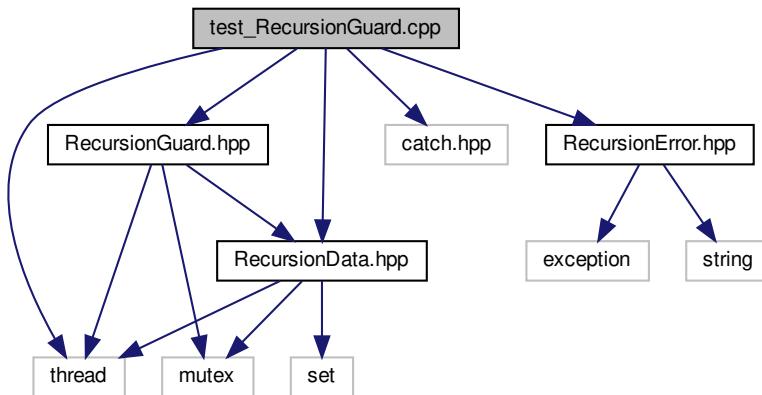
16.251 test_RecursionGuard.cpp File Reference

```

#include <thread>
#include <catch.hpp>
#include "RecursionData.hpp"
#include "RecursionError.hpp"
#include "RecursionGuard.hpp"

```

Include dependency graph for test_RecursionGuard.cpp:



Functions

- [TEST_CASE \("RecursionGuard's are constructable.", "\[RecursionGuard\]"\)](#)
- [TEST_CASE \("RecursionGuard's prevent recursion within a single thread.", "\[RecursionGuard\]"\)](#)

16.251.1 Function Documentation

16.251.1.1 TEST_CASE() [1/2]

```
TEST_CASE (
    "RecursionGuard's are constructable." ,
    "" [RecursionGuard] )
```

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

16.251.1.2 TEST_CASE() [2/2]

```
TEST_CASE (
    "RecursionGuard's prevent recursion within a single thread." ,
    "" [RecursionGuard] )
```

Definition at line 34 of file [test_RecursionGuard.cpp](#).

16.252 test_RecursionGuard.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <thread>
00019
00020 #include <catch.hpp>
00021
00022 #include "RecursionData.hpp"
00023 #include "RecursionError.hpp"
00024 #include "RecursionGuard.hpp"
00025
00026
00027 TEST_CASE("RecursionGuard's are constructable.", "[RecursionGuard]")
00028 {
00029     RecursionData rdata;
00030     REQUIRE_NO_THROW(RecursionGuard(rdata));
00031 }
00032
00033
00034 TEST_CASE("RecursionGuard's prevent recursion within a single thread.",
00035             "[RecursionGuard]")
```

```

00036 {
00037     SECTION("No recursion.")
00038     {
00039         RecursionData rdata;
00040         {
00041             REQUIRE_NOTHROW(RecursionGuard(rdata));
00042         }
00043         {
00044             REQUIRE_NOTHROW(RecursionGuard(rdata));
00045         }
00046         {
00047             REQUIRE_NOTHROW(RecursionGuard(rdata));
00048         }
00049     }
00050     SECTION("Throws an error on recursion.")
00051     {
00052         RecursionData rdata;
00053         RecursionGuard rg(rdata);
00054         REQUIRE_THROWS_AS(RecursionGuard(rdata), RecursionError);
00055         REQUIRE_THROWS_WITH(RecursionGuard(rdata), "Recursion detected.");
00056     }
00057     SECTION("Is thread safe (calls across threads is not recursion).")
00058     {
00059         RecursionData rdata;
00060         RecursionGuard rguard(rdata);
00061         std::thread thread([&]()
00062         {
00063             RecursionGuard rguard_(rdata);
00064         });
00065         thread.join();
00066     }
00067 }

```

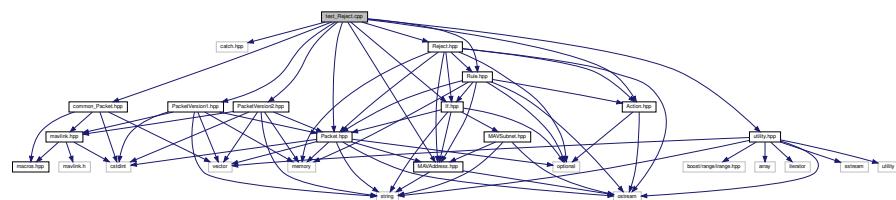
16.253 test_Reject.cpp File Reference

```

#include <catch.hpp>
#include "Action.hpp"
#include "If.hpp"
#include "MAVAddress.hpp"
#include "Packet.hpp"
#include "PacketVersion1.hpp"
#include "PacketVersion2.hpp"
#include "Reject.hpp"
#include "Rule.hpp"
#include "utility.hpp"
#include "common_Packet.hpp"

```

Include dependency graph for test_Reject.cpp:



Functions

- `TEST_CASE` ("Reject's are constructable.", "[`Reject`]"")

- [TEST_CASE \("Reject's are comparable.", "\[Reject\]"\)](#)
- [TEST_CASE \("Reject's 'action' method determines what to do with a " "packet/address combination.", "\[Reject\]"\)](#)
- [Reject reject \(`If\(\)`.type\("PING"\).from\("192.168/8"\).to\("172.16/4"\)\)](#)
- [TEST_CASE \("Reject's 'clone' method returns a polymorphic copy.", "\[Reject\]"\)](#)

Variables

- [Reject reject](#)
- [Rule & rule = reject](#)

16.253.1 Function Documentation

16.253.1.1 `reject()`

```
Reject reject (
    If().type ("PING") .from ("192.168/8") .to ("172.16/4") )
```

16.253.1.2 `TEST_CASE()` [1/4]

```
TEST_CASE (
    "Reject's are constructable." ,
    "" [Reject] )
```

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

16.253.1.3 `TEST_CASE()` [2/4]

```
TEST_CASE (
    "Reject's are comparable." ,
    "" [Reject] )
```

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

16.253.1.4 TEST_CASE() [3/4]

```
TEST_CASE (
    "Reject's 'action' method determines what to do with a \"packet/address combination.\"",
    "" [Reject] )
```

Definition at line 71 of file [test_Reject.cpp](#).

References [ping](#).

16.253.1.5 TEST_CASE() [4/4]

```
TEST_CASE (
    "Reject's 'clone' method returns a polymorphic copy." ,
    "" [Reject] )
```

Definition at line 134 of file [test_Reject.cpp](#).

References [Rule::clone\(\)](#), [reject](#), and [rule](#).

Here is the call graph for this function:



16.253.2 Variable Documentation

16.253.2.1 reject

```
Reject reject(IF().type("PING").from("192.168/8").to("172.16/4"))
```

Initial value:

```
{  
    auto ping = packet_v2::Packet(to_vector(PingV2()))
```

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

16.253.2.2 rule

```
Rule& rule = reject
```

Definition at line 106 of file [test_Reject.cpp](#).

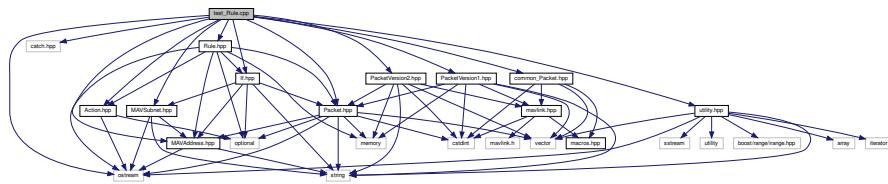
16.254 test_Reject.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
0010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
0011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
0012 // GNU General Public License for more details.
0013 //
0014 // You should have received a copy of the GNU General Public License
0015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
0016
0017
0018 #include <catch.hpp>
0019
0020 #include "Action.hpp"
0021 #include "If.hpp"
0022 #include "MAVAddress.hpp"
0023 #include "Packet.hpp"
0024 #include "PacketVersion1.hpp"
0025 #include "PacketVersion2.hpp"
0026 #include "Reject.hpp"
0027 #include "Rule.hpp"
0028 #include "utility.hpp"
0029
0030 #include "common_Packet.hpp"
0031
0032
0033 TEST_CASE("Reject's are constructable.", "[Reject]")
0034 {
0035     SECTION("Without a condition (match all packet/address combinations).")
0036     {
0037         REQUIRE_NOTHROW(Reject());
0038     }
0039     SECTION("With a condition.")
0040     {
0041         REQUIRE_NOTHROW(Reject(If()));
0042         REQUIRE_NOTHROW(Reject(If().type("PING")));
0043         REQUIRE_NOTHROW(Reject(If().from("192.168")));
0044         REQUIRE_NOTHROW(Reject(If().to("172.16")));
0045     }
0046 }
0047
0048
0049 TEST_CASE("Reject's are comparable.", "[Reject]")
0050 {
0051     SECTION("with ==")
0052     {
0053         REQUIRE(Reject() == Reject());
0054         REQUIRE(Reject(If().type("PING")) == Reject(If().type("PING")));
0055         REQUIRE_FALSE(
0056             Reject(If().type("PING")) == Reject(If().type("SET_MODE")));
0057         REQUIRE_FALSE(Reject(If().type("PING")) == Reject(If()));
0058         REQUIRE_FALSE(Reject(If().type("PING")) == Reject());
0059     }
0060     SECTION("with !=")
0061     {
0062         REQUIRE(Reject(If().type("PING")) != Reject());
0063         REQUIRE(Reject(If().type("PING")) != Reject(If()));
0064         REQUIRE(Reject(If().type("PING")) != Reject(If().type("SET_MODE")));
0065         REQUIRE_FALSE(Reject() != Reject());
0066     }
0067 }
```

```
00066     REQUIRE_FALSE(Reject(If().type("PING")) != Reject(If().type("PING")));
00067 }
00068 }
00069
00070
00071 TEST_CASE("Reject's 'action' method determines what to do with a "
00072             "packet/address combination.", "[Reject]")
00073 {
00074     auto ping = packet_v2::Packet(to_vector(PingV2()));
00075     SECTION("Returns the reject action if there is no conditional.")
00076     {
00077         REQUIRE(
00078             Reject().action(ping, MAVAddress("192.168")) ==
00079             Action::make_reject());
00080     }
00081     SECTION("Returns the reject action if the conditional is a match.")
00082     {
00083         REQUIRE(
00084             Reject(If().type("PING")).action(ping, MAVAddress("192.168")) ==
00085             Action::make_reject());
00086         REQUIRE(
00087             Reject(If().to("192.168")).action(ping, MAVAddress("192.168")) ==
00088             Action::make_reject());
00089     }
00090     SECTION("Returns the continue action if the conditional does not match.")
00091     {
00092         REQUIRE(
00093             Reject(If().type("SET_MODE")).action(ping, MAVAddress("192.168")) ==
00094             Action::make_continue());
00095         REQUIRE(
00096             Reject(If().to("172.16")).action(ping, MAVAddress("192.168")) ==
00097             Action::make_continue());
00098     }
00099 }
00100
00101
00102 TEST_CASE("Reject's are printable (without a condition).", "[Reject]")
00103 {
00104     auto ping = packet_v2::Packet(to_vector(PingV2()));
00105     Reject reject;
00106     Rule &rule = reject;
00107     SECTION("By direct type.")
00108     {
00109         REQUIRE(str(reject) == "reject");
00110     }
00111     SECTION("By polymorphic type.")
00112     {
00113         REQUIRE(str(rule) == "reject");
00114     }
00115 }
00116
00117
00118 TEST_CASE("Reject's are printable (with a condition).", "[Reject]")
00119 {
00120     auto ping = packet_v2::Packet(to_vector(PingV2()));
00121     Reject reject(If().type("PING").from("192.168/8").to("172.16/4"));
00122     Rule &rule = reject;
00123     SECTION("By direct type.")
00124     {
00125         REQUIRE(str(reject) == "reject if PING from 192.168/8 to 172.16/4");
00126     }
00127     SECTION("By polymorphic type.")
00128     {
00129         REQUIRE(str(rule) == "reject if PING from 192.168/8 to 172.16/4");
00130     }
00131 }
00132
00133
00134 TEST_CASE("Reject's 'clone' method returns a polymorphic copy.", "[Reject]")
00135 {
00136     Reject reject(If().type("PING"));
00137     Rule &rule = reject;
00138     std::unique_ptr<Rule> polymorphic_copy = rule.clone();
00139     REQUIRE(reject == *polymorphic_copy);
00140 }
```

16.255 test_Rule.cpp File Reference

```
#include <iostream>
#include <catch.hpp>
#include "Action.hpp"
#include "If.hpp"
#include "MAVAddress.hpp"
#include "MAVSubnet.hpp"
#include "Packet.hpp"
#include "PacketVersion1.hpp"
#include "PacketVersion2.hpp"
#include "Rule.hpp"
#include "utility.hpp"
#include "common_Packet.hpp"
Include dependency graph for test_Rule.cpp:
```



Functions

- [TEST_CASE \("Rule's can be constructed.", "\[Rule\]"\)](#)
- [TEST_CASE \("Rule's are comparable.", "\[Rule\]"\)](#)
- [TEST_CASE \("Rule's 'action' method determines what to do with a packet with " "respect to a destination address.", "\[Rule\]"\)](#)
- [TEST_CASE \("Rule's are printable.", "\[Rule\]"\)](#)
- [TEST_CASE \("Rule's 'clone' method returns a polymorphic copy.", "\[Rule\]"\)](#)

16.255.1 Function Documentation

16.255.1.1 TEST_CASE() [1/5]

```
TEST_CASE (
    "Rule's can be constructed." ,
    "" [Rule] )
```

Definition at line 82 of file [test_Rule.cpp](#).

16.255.1.2 TEST_CASE() [2/5]

```
TEST_CASE (
    "Rule's are comparable." ,
    "" [Rule] )
```

Definition at line 88 of file [test_Rule.cpp](#).

16.255.1.3 TEST_CASE() [3/5]

```
TEST_CASE (
    "Rule's 'action' method determines what to do with a packet with \"respect to a
destination address." ,
    "" [Rule] )
```

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

References [ping](#), and [rule](#).

16.255.1.4 TEST_CASE() [4/5]

```
TEST_CASE (
    "Rule's are printable." ,
    "" [Rule] )
```

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

References [ping](#), and [rule](#).

16.255.1.5 TEST_CASE() [5/5]

```
TEST_CASE (
    "Rule's 'clone' method returns a polymorphic copy." ,
    "" [Rule] )
```

Definition at line 144 of file [test_Rule.cpp](#).

References [Rule::clone\(\)](#), and [rule](#).

Here is the call graph for this function:



16.256 test_Rule.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <iostream>
00019
00020 #include <catch.hpp>
00021
00022 #include "Action.hpp"
00023 #include "If.hpp"
00024 #include "MAVAddress.hpp"
00025 #include "MAVSubnet.hpp"
00026 #include "Packet.hpp"
00027 #include "PacketVersion1.hpp"
00028 #include "PacketVersion2.hpp"
00029 #include "Rule.hpp"
00030 #include "utility.hpp"
00031
00032 #include "common_Packet.hpp"
00033
00034
00035 namespace
00036 {
00037
00038     class RuleTestClass : public Rule
00039     {
00040         protected:
00041             virtual std::ostream &print_(std::ostream &os) const
00042             {
00043                 os << "test_rule";
00044                 return os;
00045             }
00046
00047         public:
00048             virtual std::unique_ptr<Rule> clone() const
00049             {
00050                 return std::make_unique<RuleTestClass>();
00051             }
00052             virtual Action action(
00053                 const Packet &packet, const MAVAddress &address) const
00054             {
00055                 if (packet.name() == "PING")
00056                 {
00057                     if (MAVSubnet("192.0/14").contains(address))
00058                     {
00059                         return Action::make_accept();
00060                     }
00061
00062                     return Action::make_reject();
00063                 }
00064
00065                 return Action::make_continue();
00066             }
00067             virtual bool operator==(const Rule &other) const
00068             {
00069                 (void)other;
00070                 return true;
00071             }
00072             virtual bool operator!=(const Rule &other) const
00073             {
00074                 (void)other;
00075                 return false;
00076             }
00077     };
}
```

```
00078
00079 }
00080
00081
00082 TEST_CASE("Rule's can be constructed.", "[Rule]")
00083 {
00084     REQUIRE_NO_THROW(RuleTestClass());
00085 }
00086
00087
00088 TEST_CASE("Rule's are comparable.", "[Rule]")
00089 {
00090     REQUIRE(RuleTestClass() == RuleTestClass());
00091     REQUIRE_FALSE(RuleTestClass() != RuleTestClass());
00092 }
00093
00094
00095 TEST_CASE("Rule's 'action' method determines what to do with a packet with "
00096             "respect to a destination address.", "[Rule]")
00097 {
00098     auto ping = packet_v1::Packet(to_vector(PingV1()));
00099     auto set_mode = packet_v2::Packet(to_vector(SetModeV2()));
00100     RuleTestClass rule;
00101     REQUIRE(rule.action(ping, MAVAddress("192.0")) ==
00102             Action::make_accept());
00103     REQUIRE(rule.action(ping, MAVAddress("192.1")) ==
00104             Action::make_accept());
00105     REQUIRE(rule.action(ping, MAVAddress("192.2")) ==
00106             Action::make_accept());
00107     REQUIRE(rule.action(ping, MAVAddress("192.3")) ==
00108             Action::make_accept());
00109     REQUIRE(rule.action(ping, MAVAddress("192.4")) ==
00110             Action::make_reject());
00111     REQUIRE(rule.action(ping, MAVAddress("192.5")) ==
00112             Action::make_reject());
00113     REQUIRE(rule.action(ping, MAVAddress("192.6")) ==
00114             Action::make_reject());
00115     REQUIRE(rule.action(ping, MAVAddress("192.7")) ==
00116             Action::make_reject());
00117     REQUIRE(
00118         rule.action(set_mode, MAVAddress("192.0")) ==
00119         Action::make_continue());
00120     REQUIRE(
00121         rule.action(set_mode, MAVAddress("192.1")) ==
00122         Action::make_continue());
00123     REQUIRE(
00124         rule.action(set_mode, MAVAddress("192.2")) ==
00125         Action::make_continue());
00126
00127
00128 TEST_CASE("Rule's are printable.", "[Rule]")
00129 {
00130     auto ping = packet_v2::Packet(to_vector(PingV2()));
00131     RuleTestClass rule;
00132     Rule &polymorphic = rule;
00133     SECTION("By direct type.")
00134     {
00135         REQUIRE(str(rule) == "test_rule");
00136     }
00137     SECTION("By polymorphic type.")
00138     {
00139         REQUIRE(str(polymorphic) == "test_rule");
00140     }
00141 }
00142
```

```

00143
00144 TEST_CASE("Rule's 'clone' method returns a polymorphic copy.", "[Rule]")
00145 {
00146     RuleTestClass rule;
00147     Rule &polymorphic = rule;
00148     std::unique_ptr<Rule> polymorphic_copy = polymorphic.clone();
00149     REQUIRE(rule == *polymorphic_copy);
00150 }

```

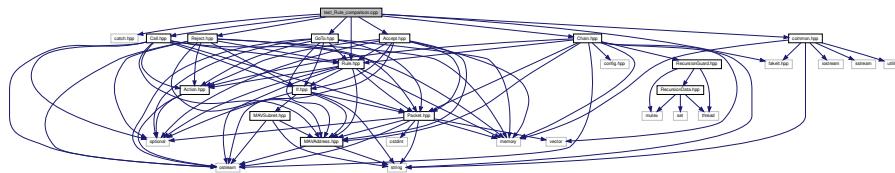
16.257 test_Rule_comparison.cpp File Reference

```

#include <catch.hpp>
#include <fakeit.hpp>
#include "Accept.hpp"
#include "Call.hpp"
#include "Chain.hpp"
#include "GoTo.hpp"
#include "Reject.hpp"
#include "Rule.hpp"
#include "common.hpp"

```

Include dependency graph for test_Rule_comparison.cpp:



Functions

- [TEST_CASE \("Rule's are polymorphically comparable.", "\[Rule\]"\)](#)

16.257.1 Function Documentation

16.257.1.1 TEST_CASE()

```

TEST_CASE (
    "Rule's are polymorphically comparable." ,
    "" [Rule] )

```

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

References [mock_shared\(\)](#).

Here is the call graph for this function:



16.258 test_Rule_comparison.cpp

```

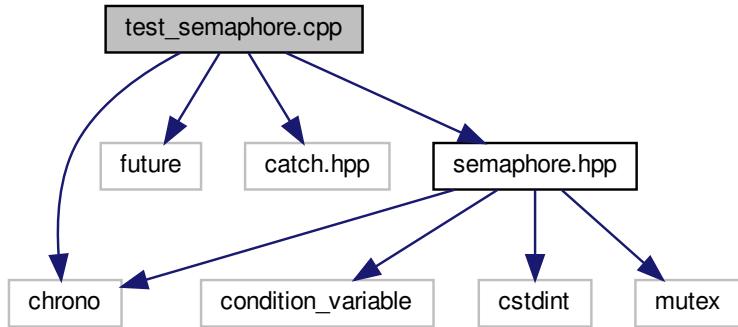
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <catch.hpp>
00019 #include <fakeit.hpp>
00020
00021 #include "Accept.hpp"
00022 #include "Call.hpp"
00023 #include "Chain.hpp"
00024 #include "GoTo.hpp"
00025 #include "Reject.hpp"
00026 #include "Rule.hpp"
00027
00028 #include "common.hpp"
00029
00030
00031 TEST_CASE("Rule's are polymorphically comparable.", "[Rule]")
00032 {
00033     fakeit::Mock<Chain> mock;
00034     std::shared_ptr<Chain> chain = mock_shared(mock);
00035     SECTION("with ==")
00036     {
00037         REQUIRE_FALSE(Accept() == Reject());
00038         REQUIRE_FALSE(Accept() == Call(chain));
00039         REQUIRE_FALSE(Accept() == GoTo(chain));
00040         REQUIRE_FALSE(Reject() == Accept());
00041         REQUIRE_FALSE(Reject() == Call(chain));
00042         REQUIRE_FALSE(Reject() == GoTo(chain));
00043         REQUIRE_FALSE(Call(chain) == Accept());
00044         REQUIRE_FALSE(Call(chain) == Reject());
00045         REQUIRE_FALSE(Call(chain) == GoTo(chain));
00046         REQUIRE_FALSE(GoTo(chain) == Accept());
00047         REQUIRE_FALSE(GoTo(chain) == Reject());
00048         REQUIRE_FALSE(GoTo(chain) == Call(chain));
00049     }
00050     SECTION("with !=")
00051     {
00052         REQUIRE(Accept() != Reject());
00053         REQUIRE(Accept() != Call(chain));
00054         REQUIRE(Accept() != GoTo(chain));
00055         REQUIRE(Reject() != Accept());
00056         REQUIRE(Reject() != Call(chain));
  
```

```

00057     REQUIRE(Reject() != GoTo(chain));
00058     REQUIRE(Call(chain) != Accept());
00059     REQUIRE(Call(chain) != Reject());
00060     REQUIRE(Call(chain) != GoTo(chain));
00061     REQUIRE(GoTo(chain) != Accept());
00062     REQUIRE(GoTo(chain) != Reject());
00063     REQUIRE(GoTo(chain) != Call(chain));
00064 }
00065 }
```

16.259 test_semaphore.cpp File Reference

```
#include <chrono>
#include <future>
#include <catch.hpp>
#include <semaphore.hpp>
Include dependency graph for test_semaphore.cpp:
```



Functions

- [TEST_CASE \("semaphore's can be constructed.", "\[semaphore\]"\)](#)
- [TEST_CASE \("semaphore's 'wait' method waits until the semaphore can be decremented.", "\[semaphore\]"\)](#)
- [TEST_CASE \("semaphore's 'wait_for' method waits until the semaphore can be decremented, or the timeout is reached \(returning false if it timed out\).", "\[semaphore\]"\)](#)
- [TEST_CASE \("semaphore's 'wait_until' method waits until the semaphore can be decremented, or the timeout timepoint is reached \(returning false if it timed out\).", "\[semaphore\]"\)](#)

16.259.1 Function Documentation

16.259.1.1 TEST_CASE() [1/4]

```
TEST_CASE (
    "semaphore's can be constructed." ,
    "" [semaphore] )
```

Definition at line 29 of file [test_semaphore.cpp](#).

16.259.1.2 TEST_CASE() [2/4]

```
TEST_CASE (
    "semaphore's 'wait' method waits until the semaphore can be " "decremented." ,
    "" [semaphore] )
```

Definition at line 36 of file [test_semaphore.cpp](#).

16.259.1.3 TEST_CASE() [3/4]

```
TEST_CASE (
    "semaphore's 'wait_for' method waits until the semaphore can be " " decremented,
    or the timeout is reached(returning false if it " "timed out)." ,
    "" [semaphore] )
```

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

16.259.1.4 TEST_CASE() [4/4]

```
TEST_CASE (
    "semaphore's 'wait_until' method waits until the semaphore can be " " decremented,
    or the timeout timepoint is reached(returning false " "if it timed out)." ,
    "" [semaphore] )
```

Definition at line 188 of file [test_semaphore.cpp](#).

16.260 test_semaphore.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <chrono>
00019 #include <future>
00020
00021 #include <catch.hpp>
00022
00023 #include <semaphore.hpp>
00024
00025
00026 using namespace std::chrono_literals;
00027
00028
00029 TEST_CASE("semaphore's can be constructed.", "[semaphore]")
00030 {
00031     REQUIRE_NO_THROW(semaphore());
00032     REQUIRE_NO_THROW(semaphore(10));
00033 }
00034
00035
00036 TEST_CASE("semaphore's 'wait' method waits until the semaphore can be "
00037             "decremented.", "[semaphore]")
00038 {
00039     SECTION("Single wait.")
00040     {
00041         semaphore sp;
00042         auto future = std::async(std::launch::async, [&]()
00043         {
00044             sp.wait();
00045         });
00046         REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00047         sp.notify();
00048         REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00049         future.wait();
00050     }
00051     SECTION("Multiple wait.")
00052     {
00053         semaphore sp;
00054         auto future = std::async(std::launch::async, [&]()
00055         {
00056             sp.wait();
00057             sp.wait();
00058         });
00059         REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00060         sp.notify();
00061         REQUIRE(future.wait_for(10ms) != std::future_status::ready);
00062         sp.notify();
00063         REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00064         future.wait();
00065     }
00066     SECTION("Single wait, with initial value by notification.")
00067     {
00068         semaphore sp;
00069         sp.notify();
00070         auto future = std::async(std::launch::async, [&]()
00071         {
00072             sp.wait();
00073         });
00074         REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00075         future.wait();
00076     }
00077     SECTION("Multiple wait, with initial value by notification.")

```

```
00078     {
00079         semaphore sp;
00080         sp.notify();
00081         auto future = std::async(std::launch::async, [&]()
00082         {
00083             sp.wait();
00084             sp.wait();
00085         });
00086         REQUIRE(future.wait_for(10ms) != std::future_status::ready);
00087         sp.notify();
00088         REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00089         future.wait();
00090     }
00091     SECTION("Single wait, with initial value in constructor.")
00092     {
00093         semaphore sp(1);
00094         auto future = std::async(std::launch::async, [&]()
00095         {
00096             sp.wait();
00097         });
00098         REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00099         future.wait();
00100    }
00101    SECTION("Multiple wait, with initial value in constructor.")
00102    {
00103        semaphore sp(2);
00104        auto future = std::async(std::launch::async, [&]()
00105        {
00106            sp.wait();
00107            sp.wait();
00108        });
00109        REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00110        future.wait();
00111    }
00112 }
00113
00114
00115 TEST_CASE("semaphore's 'wait_for' method waits until the semaphore can be "
00116             "decremented, or the timeout is reached (returning false if it "
00117             "timed out).", "[semaphore]")
00118 {
00119     SECTION("Single wait (no timeout).")
00120     {
00121         semaphore sp;
00122         auto future = std::async(std::launch::async, [&]()
00123         {
00124             return sp.wait_for(20ms);
00125         });
00126         REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00127         sp.notify();
00128         REQUIRE(future.wait_for(5ms) == std::future_status::ready);
00129         REQUIRE(future.get());
00130     }
00131     SECTION("Single wait (timeout).")
00132     {
00133         semaphore sp;
00134         auto future = std::async(std::launch::async, [&]()
00135         {
00136             return sp.wait_for(5ms);
00137         });
00138         REQUIRE(future.wait_for(1ms) != std::future_status::ready);
00139         REQUIRE(future.wait_for(20ms) == std::future_status::ready);
00140         REQUIRE_FALSE(future.get());
00141         // Ensure subsequent waits don't fail to wait. Tests for the "always
00142         // decrement" bug.
00143         future = std::async(std::launch::async, [&]()
00144         {
00145             return sp.wait_for(5ms);
00146         });
00147         REQUIRE(future.wait_for(1ms) != std::future_status::ready);
00148         REQUIRE(future.wait_for(20ms) == std::future_status::ready);
00149         REQUIRE_FALSE(future.get());
00150     }
00151     SECTION("Multiple wait (no timeout).")
00152     {
00153         semaphore sp;
00154         auto future = std::async(std::launch::async, [&]()
00155         {
00156             return sp.wait_for(20ms) && sp.wait_for(20ms);
00157         });
00158         REQUIRE(future.wait_for(0ms) != std::future_status::ready);
```

```

00159     sp.notify();
00160     REQUIRE(future.wait_for(1ms) != std::future_status::ready);
00161     sp.notify();
00162     REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00163     REQUIRE(future.get());
00164 }
00165 SECTION("Multiple wait (timeout).")
00166 {
00167     semaphore sp;
00168     auto future = std::async(std::launch::async, [&]()
00169     {
00170         return sp.wait_for(2ms) && sp.wait_for(2ms);
00171     });
00172     REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00173     REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00174     REQUIRE_FALSE(future.get());
00175     // Ensure subsequent waits don't fail to wait. Tests for the "always
00176     // decrement" bug.
00177     future = std::async(std::launch::async, [&]()
00178     {
00179         return sp.wait_for(2ms) && sp.wait_for(2ms);
00180     });
00181     REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00182     REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00183     REQUIRE_FALSE(future.get());
00184 }
00185 }
00186
00187
00188 TEST_CASE("semaphore's 'wait_until' method waits until the semaphore can be "
00189             "decremented, or the timeout timepoint is reached (returning false "
00190             "if it timed out).", "[semaphore]")
00191 {
00192     SECTION("Single wait (no timeout).")
00193     {
00194         semaphore sp;
00195         auto future = std::async(std::launch::async, [&]()
00196         {
00197             return sp.wait_until(std::chrono::steady_clock::now() + 20ms);
00198         });
00199         REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00200         sp.notify();
00201         REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00202         REQUIRE(future.get());
00203     }
00204     SECTION("Single wait (timeout).")
00205     {
00206         semaphore sp;
00207         auto future = std::async(std::launch::async, [&]()
00208         {
00209             return sp.wait_until(std::chrono::steady_clock::now() + 2ms);
00210         });
00211         REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00212         REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00213         REQUIRE_FALSE(future.get());
00214         // Ensure subsequent waits don't fail to wait. Tests for the "always
00215         // decrement" bug.
00216         future = std::async(std::launch::async, [&]()
00217         {
00218             return sp.wait_until(std::chrono::steady_clock::now() + 5ms);
00219         });
00220         REQUIRE(future.wait_for(1ms) != std::future_status::ready);
00221         REQUIRE(future.wait_for(20ms) == std::future_status::ready);
00222         REQUIRE_FALSE(future.get());
00223     }
00224     SECTION("Multiple wait (no timeout).")
00225     {
00226         semaphore sp;
00227         auto future = std::async(std::launch::async, [&]()
00228         {
00229             return sp.wait_until(std::chrono::steady_clock::now() + 100ms) &&
00230                 sp.wait_until(std::chrono::steady_clock::now() + 100ms);
00231         });
00232         REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00233         sp.notify();
00234         REQUIRE(future.wait_for(10ms) != std::future_status::ready);
00235         sp.notify();
00236         REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00237         REQUIRE(future.get());
00238     }
00239     SECTION("Multiple wait (timeout).")

```

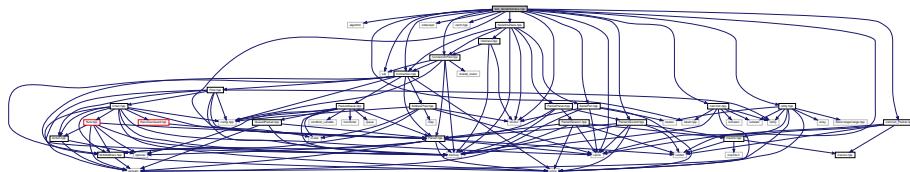
```

00240  {
00241      semaphore sp;
00242      auto future = std::async(std::launch::async, [&]()
00243      {
00244          return sp.wait_until(std::chrono::steady_clock::now() + 1ms) &&
00245          sp.wait_until(std::chrono::steady_clock::now() + 1ms);
00246      });
00247      REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00248      REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00249      REQUIRE_FALSE(future.get());
00250      // Ensure subsequent waits don't fail to wait. Tests for the "always
00251      // decrement" bug.
00252      future = std::async(std::launch::async, [&]()
00253      {
00254          return sp.wait_until(std::chrono::steady_clock::now() + 1ms) &&
00255          sp.wait_until(std::chrono::steady_clock::now() + 1ms);
00256      });
00257      REQUIRE(future.wait_for(0ms) != std::future_status::ready);
00258      REQUIRE(future.wait_for(10ms) == std::future_status::ready);
00259      REQUIRE_FALSE(future.get());
00260  }
00261 }
```

16.261 test_SerialInterface.cpp File Reference

```

#include <algorithm>
#include <chrono>
#include <cstdint>
#include <set>
#include <stdexcept>
#include <vector>
#include <catch.hpp>
#include <fakeit.hpp>
#include "Connection.hpp"
#include "ConnectionPool.hpp"
#include "Packet.hpp"
#include "PacketVersion2.hpp"
#include "SerialInterface.hpp"
#include "SerialPort.hpp"
#include "utility.hpp"
#include "common.hpp"
#include "common_Packet.hpp"
Include dependency graph for test_SerialInterface.cpp:
```



Functions

- [TEST_CASE \("SerialInterface's can be constructed.", "\[SerialInterface\]"\)](#)
- [TEST_CASE \("SerialInterface's 'receive_packet' method.", "\[SerialInterface\]"\)](#)
- [TEST_CASE \("SerialInterface's 'send_packet' method.", "\[SerialInterface\]"\)](#)
- [TEST_CASE \("SerialInterface's are printable.", "\[SerialInterface\]"\)](#)

16.261.1 Function Documentation

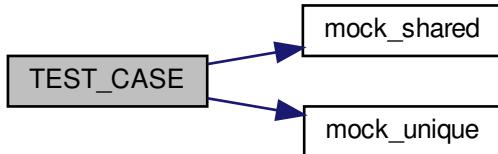
16.261.1.1 TEST_CASE() [1/4]

```
TEST_CASE (
    "SerialInterface's can be constructed." ,
    "" [SerialInterface] )
```

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

References [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



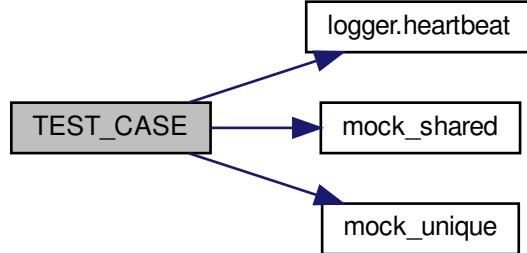
16.261.1.2 TEST_CASE() [2/4]

```
TEST_CASE (
    "SerialInterface's 'receive_packet' method." ,
    "" [SerialInterface] )
```

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

References [logger::heartbeat\(\)](#), [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



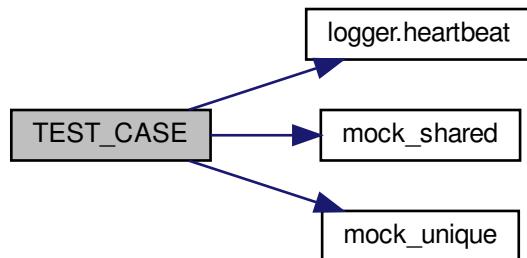
16.261.1.3 TEST_CASE() [3/4]

```
TEST_CASE (
    "SerialInterface's 'send_packet' method." ,
    "" [SerialInterface] )
```

Definition at line 312 of file [test_SerialInterface.cpp](#).

References [logger::heartbeat\(\)](#), [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



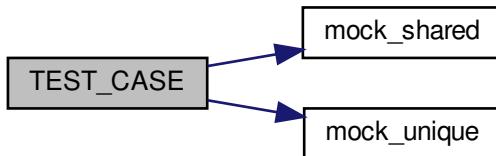
16.261.1.4 TEST_CASE() [4/4]

```
TEST_CASE (
    "SerialInterface's are printable." ,
    "" [SerialInterface] )
```

Definition at line 395 of file [test_SerialInterface.cpp](#).

References [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



16.262 test_SerialInterface.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <algorithm>
00019 #include <chrono>
00020 #include <cstdint>
00021 #include <set>
00022 #include <stdexcept>
00023 #include <vector>
00024
00025 #include <catch.hpp>
00026 #include <fakeit.hpp>
00027
00028 #include "Connection.hpp"
00029 #include "ConnectionPool.hpp"
00030 #include "Packet.hpp"
00031 #include "PacketVersion2.hpp"
00032 #include "SerialInterface.hpp"
00033 #include "SerialPort.hpp"
00034 #include "utility.hpp"
00035
```

```

00036 #include "common.hpp"
00037 #include "common_Packet.hpp"
00038
00039
00040 using namespace std::chrono_literals;
00041
00042
00043 TEST_CASE("SerialInterface's can be constructed.", "[SerialInterface]")
00044 {
00045     fakeit::Mock<SerialPort> mock_port;
00046     fakeit::Mock<ConnectionPool> mock_pool;
00047     fakeit::Mock<Connection> mock_connection;
00048     fakeit::Fake(Method(mock_pool, add));
00049     auto port = mock_unique(mock_port);
00050     auto pool = mock_shared(mock_pool);
00051     auto connection = mock_unique(mock_connection);
00052     SECTION("When all inputs are valid, registers connection with pool.")
00053     {
00054         Connection *conn = nullptr;
00055         fakeit::When(Method(mock_pool, add)).AlwaysDo([&](auto a)
00056         {
00057             conn = a.lock().get();
00058         });
00059         REQUIRE_NO_THROW(
00060             SerialInterface(std::move(port), pool, std::move(connection)));
00061         fakeit::Verify(Method(mock_pool, add)).Once();
00062         REQUIRE(conn == &mock_connection.get());
00063     }
00064     SECTION("Ensures the serial port pointer is not null.")
00065     {
00066         REQUIRE_THROWS_AS(
00067             SerialInterface(nullptr, pool, std::move(connection)),
00068             std::invalid_argument);
00069         connection = mock_unique(mock_connection);
00070         REQUIRE_THROWS_WITH(
00071             SerialInterface(nullptr, pool, std::move(connection)),
00072             "Given serial port pointer is null.");
00073     }
00074     SECTION("Ensures the connection pool pointer is not null.")
00075     {
00076         REQUIRE_THROWS_AS(
00077             SerialInterface(std::move(port), nullptr, std::move(connection)),
00078             std::invalid_argument);
00079         port = mock_unique(mock_port);
00080         connection = mock_unique(mock_connection);
00081         REQUIRE_THROWS_WITH(
00082             SerialInterface(std::move(port), nullptr, std::move(connection)),
00083             "Given connection pool pointer is null.");
00084     }
00085     SECTION("Ensures the connection pointer is not null.")
00086     {
00087         REQUIRE_THROWS_AS(
00088             SerialInterface(std::move(port), pool, nullptr),
00089             std::invalid_argument);
00090         port = mock_unique(mock_port);
00091         REQUIRE_THROWS_WITH(
00092             SerialInterface(std::move(port), pool, nullptr),
00093             "Given connection pointer is null.");
00094     }
00095 }
00096
00097
00098 TEST_CASE("SerialInterface's 'receive_packet' method.", "[SerialInterface]")
00099 {
00100     // MAVLink packets.
00101     auto heartbeat =
00102         std::make_shared<packet_v2::Packet>(to_vector(HeartbeatV2()));
00103     auto encapsulated_data =
00104         std::make_shared<packet_v2::Packet>(to_vector(EncapsulatedDataV2()));
00105     // Serial port
00106     SerialPort serial_port;
00107     fakeit::Mock<SerialPort> mock_port(serial_port);
00108     auto port = mock_unique(mock_port);
00109     using read_type =
00110         void(std::back_insert_iterator<std::vector<uint8_t>>,
00111             const std::chrono::nanoseconds &);
00112     // Pool
00113     fakeit::Mock<ConnectionPool> mock_pool;
00114     auto pool = mock_shared(mock_pool);
00115     fakeit::Fake(Method(mock_pool, add));
00116     std::multiset<packet_v2::Packet,

```

```

0017      bool(*)(const packet_v2::Packet &, const
0018      packet_v2::Packet &)
0019  {
0020      return a.data() < b.data();
0021  };
0022  fakeit::When(Method(mock_pool, send)) .AlwaysDo([&] (auto & a)
0023  {
0024      const Packet *packet = a.get();
0025      send_packets.insert(
0026          packet_v2::Packet(
0027              *dynamic_cast<const packet_v2::Packet *>(packet)));
0028  });
0029  // Connection
0030  fakeit::Mock<Connection> mock_connection;
0031  auto connection = mock_unique(mock_connection);
0032  fakeit::Fake(Method(mock_connection, add_address));
0033  // Interface
0034  SerialInterface serial(std::move(port), pool, std::move(connection));
0035  std::chrono::nanoseconds timeout = 250ms;
0036  SECTION("No packet received.")
0037  {
0038      // Mocks
0039      fakeit::When(OverloadedMethod(mock_port, read, read_type)
0040                  ).AlwaysDo([](auto a, auto b)
0041  {
0042      (void)a;
0043      (void)b;
0044  });
0045      // Test
0046      serial.receive_packet(timeout);
0047      // Verification
0048      fakeit::Verify(OverloadedMethod(mock_port, read, read_type).Matching(
0049                      [&](auto a, auto b)
0050  {
0051      (void)a;
0052      return b == 250ms;
0053  })).Once();
0054      fakeit::Verify(Method(mock_connection, add_address)).Exactly(0);
0055      fakeit::Verify(Method(mock_pool, send)).Exactly(0);
0056  }
0057  SECTION("Partial packet received.")
0058  {
0059      // Mocks
0060      fakeit::When(OverloadedMethod(mock_port, read, read_type)
0061                  ).AlwaysDo([](auto a, auto b)
0062  {
0063      (void)b;
0064      auto vec = to_vector(HeartbeatV2());
0065      std::copy(vec.begin(), vec.end() - 1, a);
0066  });
0067      // Test
0068      serial.receive_packet(timeout);
0069      // Verification
0070      fakeit::Verify(OverloadedMethod(mock_port, read, read_type).Matching(
0071                      [&](auto a, auto b)
0072  {
0073      (void)a;
0074      return b == 250ms;
0075  })).Once();
0076      fakeit::Verify(Method(mock_connection, add_address)).Exactly(0);
0077      fakeit::Verify(Method(mock_pool, send)).Exactly(0);
0078  }
0079  SECTION("Full packet received.")
0080  {
0081      // Mocks
0082      fakeit::When(OverloadedMethod(mock_port, read, read_type)
0083                  ).AlwaysDo([](auto a, auto b)
0084  {
0085      (void)b;
0086      auto vec = to_vector(HeartbeatV2());
0087      std::copy(vec.begin(), vec.end(), a);
0088  });
0089      // Test
0090      serial.receive_packet(timeout);
0091      // Verification
0092      fakeit::Verify(OverloadedMethod(mock_port, read, read_type).Matching(
0093                      [&](auto a, auto b)
0094  {
0095      (void)a;
0096      return b == 250ms;
0097  });

```

```

00197     }).Once();
00198     fakeit::Verify(Method(mock_connection, add_address
00199         ).Using(MAVAddress("127.1"))).Exactly(1);
00200     fakeit::Verify(Method(mock_pool, send)).Exactly(1);
00201     REQUIRE(send_packets.count(*heartbeat) == 1);
00202     auto it = send_packets.find(*heartbeat);
00203     REQUIRE(it != send_packets.end());
00204     REQUIRE(it->connection() != nullptr);
00205 }
00206 SECTION("Multiple packets received (same MAVLink address).")
00207 {
00208     // Mocks
00209     fakeit::When(OverloadedMethod(mock_port, read, read_type)
00210         ).AlwaysDo([](auto a, auto b)
00211     {
00212         (void)b;
00213         auto vec = to_vector(HeartbeatV2());
00214         std::copy(vec.begin(), vec.end(), a);
00215     });
00216     // Test
00217     serial.receive_packet(timeout);
00218     serial.receive_packet(timeout);
00219     // Verification
00220     fakeit::Verify(OverloadedMethod(mock_port, read, read_type).Matching(
00221         [&](auto a, auto b)
00222     {
00223         (void)a;
00224         return b == 250ms;
00225     })).Exactly(2);
00226     fakeit::Verify(Method(mock_connection, add_address
00227         ).Using(MAVAddress("127.1"))).Exactly(2);
00228     fakeit::Verify(Method(mock_pool, send)).Exactly(2);
00229     REQUIRE(send_packets.count(*heartbeat) == 2);
00230     auto it = send_packets.find(*heartbeat);
00231     REQUIRE(it != send_packets.end());
00232     REQUIRE(it->connection() != nullptr);
00233     it++;
00234     REQUIRE(it != send_packets.end());
00235     REQUIRE(it->connection() != nullptr);
00236 }
00237 SECTION("Multiple packets received (different MAVLink address).")
00238 {
00239     // Mocks
00240     fakeit::When(OverloadedMethod(mock_port, read, read_type)
00241         ).Do([](auto a, auto b)
00242     {
00243         (void)b;
00244         auto vec = to_vector(HeartbeatV2());
00245         std::copy(vec.begin(), vec.end(), a);
00246     }).Do([](auto a, auto b)
00247     {
00248         (void)b;
00249         auto vec = to_vector(EncapsulatedDataV2());
00250         std::copy(vec.begin(), vec.end(), a);
00251     });
00252     // Test
00253     serial.receive_packet(timeout);
00254     serial.receive_packet(timeout);
00255     // Verification
00256     fakeit::Verify(OverloadedMethod(mock_port, read, read_type).Matching(
00257         [&](auto a, auto b)
00258     {
00259         (void)a;
00260         return b == 250ms;
00261     })).Exactly(2);
00262     fakeit::Verify(Method(mock_connection, add_address
00263         ).Using(MAVAddress("127.1"))).Exactly(1);
00264     fakeit::Verify(Method(mock_connection, add_address
00265         ).Using(MAVAddress("224.255"))).Exactly(1);
00266     fakeit::Verify(Method(mock_pool, send)).Exactly(2);
00267     REQUIRE(send_packets.count(*heartbeat) == 1);
00268     REQUIRE(send_packets.count(*encapsulated_data) == 1);
00269     auto it = send_packets.find(*heartbeat);
00270     REQUIRE(it != send_packets.end());
00271     REQUIRE(it->connection() != nullptr);
00272     it = send_packets.find(*encapsulated_data);
00273     REQUIRE(it != send_packets.end());
00274     REQUIRE(it->connection() != nullptr);
00275 }
00276 SECTION("Partial packets should be combined and parsed.")
00277 {

```

```

00278     // Mocks
00279     fakeit::When(OverloadedMethod(mock_port, read, read_type)
00280                 ).Do([](auto a, auto b)
00281     {
00282         (void)b;
00283         auto vec = to_vector(EncapsulatedDataV2());
00284         std::copy(vec.begin(), vec.end() - 10, a);
00285     }).Do([](auto a, auto b)
00286     {
00287         (void)b;
00288         auto vec = to_vector(EncapsulatedDataV2());
00289         std::copy(vec.end() - 10, vec.end(), a);
00290     });
00291     // Test
00292     serial.receive_packet(timeout);
00293     serial.receive_packet(timeout);
00294     // Verification
00295     fakeit::Verify(OverloadedMethod(mock_port, read, read_type).Matching(
00296                     [&](auto a, auto b)
00297     {
00298         (void)a;
00299         return b == 250ms;
00300     })).Exactly(2);
00301     fakeit::Verify(Method(mock_connection, add_address
00302                         ).Using(MAVAddress("224.255"))).Exactly(1);
00303     fakeit::Verify(Method(mock_pool, send)).Exactly(1);
00304     REQUIRE(send_packets.count(*encapsulated_data) == 1);
00305     auto it = send_packets.find(*encapsulated_data);
00306     REQUIRE(it != send_packets.end());
00307     REQUIRE(it->connection() != nullptr);
00308 }
00309 }
00310
00311
00312 TEST_CASE("SerialInterface's 'send_packet' method.", "[SerialInterface]")
00313 {
00314     // MAVLink packets.
00315     auto heartbeat =
00316         std::make_shared<packet_v2::Packet>(to_vector(HeartbeatV2()));
00317     auto encapsulated_data =
00318         std::make_shared<packet_v2::Packet>(to_vector(EncapsulatedDataV2()));
00319     // Serial port
00320     SerialPort serial_port;
00321     fakeit::Mock<SerialPort> mock_port(serial_port);
00322     auto port = mock_unique(mock_port);
00323     using write_type =
00324         void(std::vector<uint8_t>::const_iterator first,
00325               std::vector<uint8_t>::const_iterator last);
00326     std::multiset<std::vector<uint8_t>> write_bytes;
00327     fakeit::When(OverloadedMethod(mock_port, write, write_type)).AlwaysDo(
00328         [&](auto a, auto b)
00329     {
00330         std::vector<uint8_t> vec;
00331         std::copy(a, b, std::back_inserter(vec));
00332         write_bytes.insert(vec);
00333     });
00334     // Pool
00335     fakeit::Mock<ConnectionPool> mock_pool;
00336     auto pool = mock_shared(mock_pool);
00337     fakeit::Fake(Method(mock_pool, add));
00338     // Connection
00339     fakeit::Mock<Connection> mock_connection;
00340     auto connection = mock_unique(mock_connection);
00341     // Interface
00342     SerialInterface serial(std::move(port), pool, std::move(connection));
00343     std::chrono::nanoseconds timeout = 250ms;
00344     SECTION("No packets, timeout.")
00345     {
00346         // Mocks
00347         fakeit::When(Method(mock_connection, next_packet)).Return(nullptr);
00348         // Test
00349         serial.send_packet(timeout);
00350         // Verification
00351         fakeit::Verify(
00352             Method(mock_connection, next_packet).Using(250ms)).Once();
00353         fakeit::Verify(
00354             OverloadedMethod(mock_port, write, write_type)).Exactly(0);
00355     }
00356     SECTION("Single packet.")
00357     {
00358         // Mocks

```

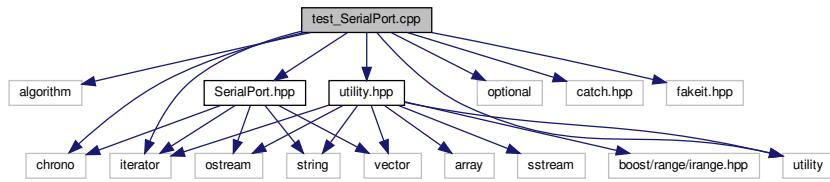
```

00359     fakeit::When(Method(mock_connection, next_packet)).Return(heartbeat);
00360     // Test
00361     serial.send_packet(timeout);
00362     // Verification
00363     fakeit::Verify(
00364         Method(mock_connection, next_packet).Using(250ms)).Once();
00365     fakeit::Verify(
00366         OverloadedMethod(mock_port, write, write_type)).Exactly(1);
00367     REQUIRE(write_bytes.count(heartbeat->data()) == 1);
00368 }
00369 SECTION("Multiple packets.")
00370 {
00371     // Mocks
00372     fakeit::When(Method(mock_connection, next_packet)
00373         ).Return(heartbeat).Return(encapsulated_data);
00374     // Test
00375     serial.send_packet(timeout);
00376     // Verification
00377     fakeit::Verify(
00378         Method(mock_connection, next_packet).Using(250ms)).Once();
00379     fakeit::Verify(
00380         OverloadedMethod(mock_port, write, write_type)).Exactly(1);
00381     REQUIRE(write_bytes.count(heartbeat->data()) == 1);
00382     // Test
00383     serial.send_packet(timeout);
00384     // Verification
00385     fakeit::Verify(
00386         Method(mock_connection, next_packet).Using(250ms)).Exactly(2);
00387     fakeit::Verify(
00388         OverloadedMethod(mock_port, write, write_type)).Exactly(2);
00389     REQUIRE(write_bytes.count(heartbeat->data()) == 1);
00390     REQUIRE(write_bytes.count(encapsulated_data->data()) == 1);
00391 }
00392 }
00393
00394
00395 TEST_CASE("SerialInterface's are printable.", "[SerialInterface]")
00396 {
00397     fakeit::Mock<ConnectionPool> mock_pool;
00398     fakeit::Mock<Connection> mock_connection;
00399     fakeit::Fake(Method(mock_pool, add));
00400     auto pool = mock_shared(mock_pool);
00401     auto connection = mock_unique(mock_connection);
00402     auto port = std::make_unique<SerialPort>();
00403     SerialInterface serial(std::move(port), pool, std::move(connection));
00404     REQUIRE(str(serial) == "unknown serial port");
00405 }
```

16.263 test_SerialPort.cpp File Reference

```
#include <algorithm>
#include <chrono>
#include <iterator>
#include <optional>
#include <utility>
#include <catch.hpp>
#include <fakeit.hpp>
#include "SerialPort.hpp"
#include "utility.hpp"
```

Include dependency graph for test_SerialPort.cpp:



Functions

- [TEST_CASE](#) ("SerialPort's 'read' method takes a timeout and returns a vector " "of bytes.", "[[SerialPort](#)]"")
- [TEST_CASE](#) ("SerialPort's 'read' method takes a back inserter and a timeout " "writes the received bytes to the back inserter.", "[[SerialPort](#)]"")
- [TEST_CASE](#) ("SerialPort's 'write' method accepts a vector of bytes.", "[[SerialPort](#)]"")
- [TEST_CASE](#) ("SerialPort's 'write' method accepts two vector iterators", "[[UDPSocket](#)]"")
- [TEST_CASE](#) ("SerialPort's are printable.", "[SerialPort](#)")

16.263.1 Function Documentation

16.263.1.1 TEST_CASE() [1/5]

```
TEST_CASE (
    "SerialPort's 'read' method takes a timeout and returns a vector " "of bytes." ,
    "" [SerialPort] )
```

Definition at line 34 of file [test_SerialPort.cpp](#).

References [SerialPort::read\(\)](#).

Here is the call graph for this function:



16.263.1.2 TEST_CASE() [2/5]

```
TEST_CASE (
    "SerialPort's 'read' method takes a back inserter and a timeout " "writes the received
bytes to the back inserter." ,
    "" [SerialPort] )
```

Definition at line 70 of file [test_SerialPort.cpp](#).

References [SerialPort::read\(\)](#).

Here is the call graph for this function:

**16.263.1.3 TEST_CASE() [3/5]**

```
TEST_CASE (
    "SerialPort's 'write' method accepts a vector of bytes." ,
    "" [SerialPort] )
```

Definition at line 100 of file [test_SerialPort.cpp](#).

References [SerialPort::write\(\)](#).

Here is the call graph for this function:



16.263.1.4 TEST_CASE() [4/5]

```
TEST_CASE (
    "SerialPort's 'write' method accepts two vector iterators" ,
    "" [UDPSocket] )
```

Definition at line 126 of file [test_SerialPort.cpp](#).

References [SerialPort::write\(\)](#).

Here is the call graph for this function:



16.263.1.5 TEST_CASE() [5/5]

```
TEST_CASE (
    "SerialPort's are printable." ,
    "SerialPort" )
```

Definition at line 152 of file [test_SerialPort.cpp](#).

16.264 test_SerialPort.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <algorithm>
00019 #include <chrono>
00020 #include <iterator>
00021 #include <optional>
```

```
00022 #include <utility>
00023
00024 #include <catch.hpp>
00025 #include <fakeit.hpp>
00026
00027 #include "SerialPort.hpp"
00028 #include "utility.hpp"
00029
00030
00031 using namespace std::chrono_literals;
00032
00033
00034 TEST_CASE("SerialPort's 'read' method takes a timeout and returns a vector "
00035             "of bytes.", "[SerialPort]")
00036 {
00037     // This test ensures that one receive method calls the other.
00038     SerialPort serial;
00039     fakeit::Mock<SerialPort> mock_port(serial);
00040     fakeit::When(
00041         OverloadedMethod(
00042             mock_port, read,
00043             void(std::back_insert_iterator<std::vector<uint8_t>>,
00044                  const std::chrono::nanoseconds &)).AlwaysDo(
00045                 [](auto a, auto b)
00046             {
00047                 (void)b;
00048                 std::vector<uint8_t> vec = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
00049                 std::copy(vec.begin(), vec.end(), a);
00050             });
00051     SerialPort &port = mock_port.get();
00052     std::vector<uint8_t> vec_compare = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
00053     std::chrono::nanoseconds timeout = 1ms;
00054     auto data = port.read(timeout);
00055     REQUIRE(data == vec_compare);
00056     fakeit::Verify(
00057         OverloadedMethod(
00058             mock_port, read,
00059             void(std::back_insert_iterator<std::vector<uint8_t>>,
00060                  const std::chrono::nanoseconds &)).Matching(
00061                 [](auto a, auto b)
00062             {
00063                 (void)a;
00064                 return b == 1ms;
00065             }).Once());
00066 }
00067
00068
00069
00070 TEST_CASE("SerialPort's 'read' method takes a back inserter and a timeout "
00071             "writes the received bytes to the back inserter.", "[SerialPort]")
00072 {
00073     // This test ensures that one read method calls the other.
00074     using read_type = std::vector<uint8_t>(const std::chrono::nanoseconds &);
00075     SerialPort serial;
00076     fakeit::Mock<SerialPort> mock_port(serial);
00077     fakeit::When(
00078         OverloadedMethod(
00079             mock_port, read, read_type)).AlwaysDo([](auto a)
00080     {
00081         (void)a;
00082         std::vector<uint8_t> vec = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
00083         return vec;
00084     });
00085     SerialPort &port = mock_port.get();
00086     std::vector<uint8_t> vec_compare = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
00087     std::chrono::nanoseconds timeout = 1ms;
00088     std::vector<uint8_t> data;
00089     port.read(std::back_inserter(data), timeout);
00090     REQUIRE(data == vec_compare);
00091     fakeit::Verify(
00092         OverloadedMethod(
00093             mock_port, read, read_type).Matching([](auto a)
00094         {
00095             return a == 1ms;
00096         }).Once());
00097 }
00098
00099
00100 TEST_CASE("SerialPort's 'write' method accepts a vector of bytes.",
00101             "[SerialPort]")
00102 }
```

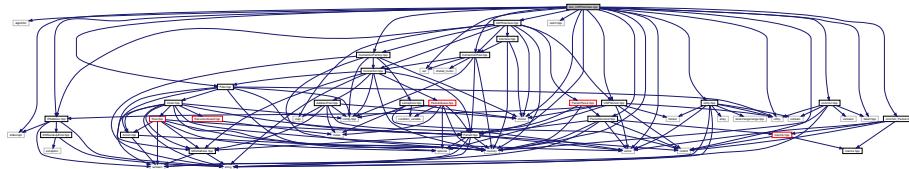
```

00103 // This test ensures that one write method calls the other.
00104 SerialPort serial;
00105 fakeit::Mock<SerialPort> mock_port(serial);
00106 fakeit::Fake(
00107     OverloadedMethod(
00108         mock_port, write,
00109         void(std::vector<uint8_t>::const_iterator,
00110             std::vector<uint8_t>::const_iterator)));
00111 SerialPort &port = mock_port.get();
00112 std::vector<uint8_t> vec = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
00113 port.write(vec);
00114 fakeit::Verify(
00115     OverloadedMethod(
00116         mock_port, write,
00117         void(std::vector<uint8_t>::const_iterator,
00118             std::vector<uint8_t>::const_iterator)).Matching(
00119     [&](auto a, auto b)
00120 {
00121     return a == vec.begin() && b == vec.end();
00122 }).Once();
00123 }
00124
00125
00126 TEST_CASE("SerialPort's 'write' method accepts two vector iterators",
00127     "[UDPSocket]")
00128 {
00129 // This test ensures that one write method calls the other.
00130 SerialPort serial;
00131 fakeit::Mock<SerialPort> mock_port(serial);
00132 std::vector<uint8_t> write_vec;
00133 fakeit::When(
00134     OverloadedMethod(
00135         mock_port, write,
00136         void(const std::vector<uint8_t> &))).AlwaysDo(
00137     [&](auto a)
00138 {
00139     write_vec = a;
00140 });
00141 SerialPort &port = mock_port.get();
00142 std::vector<uint8_t> vec = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
00143 port.write(vec.begin(), vec.end());
00144 fakeit::Verify(
00145     OverloadedMethod(
00146         mock_port, write,
00147         void(const std::vector<uint8_t> &))).Once();
00148 REQUIRE(vec == write_vec);
00149 }
00150
00151
00152 TEST_CASE("SerialPort's are printable.", "SerialPort")
00153 {
00154     REQUIRE(str(SerialPort{}) == "unknown serial port");
00155 }
```

16.265 test_UDPInterface.cpp File Reference

```
#include <algorithm>
#include <chrono>
#include <cstdint>
#include <memory>
#include <set>
#include <stdexcept>
#include <utility>
#include <vector>
#include <catch.hpp>
#include <fakeit.hpp>
#include "ConnectionFactory.hpp"
#include "ConnectionPool.hpp"
#include "Filter.hpp"
```

```
#include "IPAddress.hpp"
#include "Packet.hpp"
#include "PacketVersion2.hpp"
#include "UDPInterface.hpp"
#include "UDPSocket.hpp"
#include "utility.hpp"
#include "common.hpp"
#include "common_Packet.hpp"
Include dependency graph for test_UDPInterface.cpp:
```



Functions

- [TEST_CASE \("UDPInterface's can be constructed.", "\[UPDInterface\]"\)](#)
- [TEST_CASE \("UDPInterface's 'receive_packet' method.", "\[UPDInterface\]"\)](#)
- [TEST_CASE \("UDPInterface's 'send_packet' method.", "\[UPDInterface\]"\)](#)
- [TEST_CASE \("UDPInterface's are printable.", "\[UPDInterface\]"\)](#)
- [TEST_CASE \("OLD TEST: UDPInterface's tests.", "\[UPDInterface\]"\)](#)
- [TEST_CASE \("OLD TEST: UDPInterface's 'receive_packet' method receives one or more MAVLink packets.", "\[UPDInterface\]"\)](#)
- [TEST_CASE \("OLD TEST: UDPInterface's 'send_packet' method sends one or more MAVLink packets.", "\[UPDInterface\]"\)](#)

16.265.1 Function Documentation

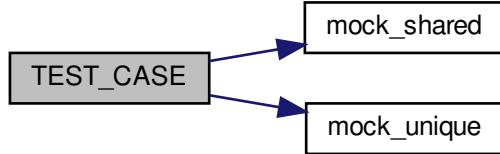
16.265.1.1 TEST_CASE() [1/7]

```
TEST_CASE (
    "UDPInterface's can be constructed." ,
    "" [UPDInterface] )
```

Definition at line 47 of file [test_UDPInterface.cpp](#).

References [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



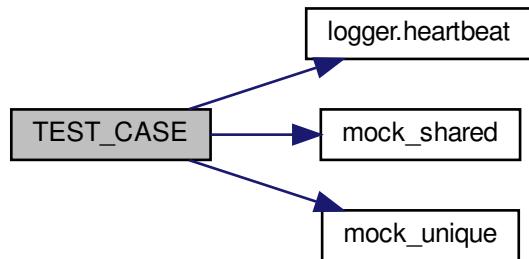
16.265.1.2 TEST_CASE() [2/7]

```
TEST_CASE (
    "UDPIterace's 'receive_packet' method." ,
    "" [UDPIterface] )
```

Definition at line 94 of file [test_UDPIterface.cpp](#).

References [logger::heartbeat\(\)](#), [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



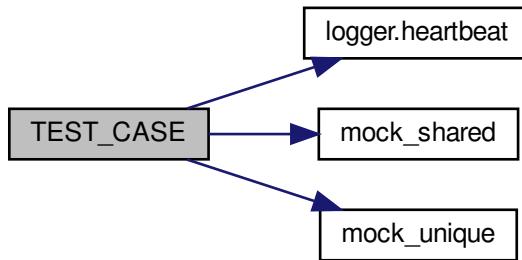
16.265.1.3 TEST_CASE() [3/7]

```
TEST_CASE (
    "UDPInterface's 'send_packet' method." ,
    "" [UDPInterface] )
```

Definition at line 454 of file [test_UDPInterface.cpp](#).

References [logger::heartbeat\(\)](#), [mock_shared\(\)](#), [mock_unique\(\)](#), and [ping](#).

Here is the call graph for this function:



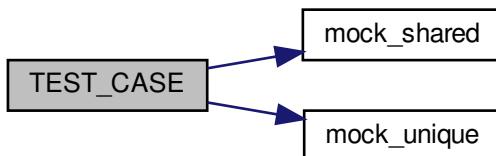
16.265.1.4 TEST_CASE() [4/7]

```
TEST_CASE (
    "UDPInterface's are printable." ,
    "" [UDPInterface] )
```

Definition at line 763 of file [test_UDPInterface.cpp](#).

References [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



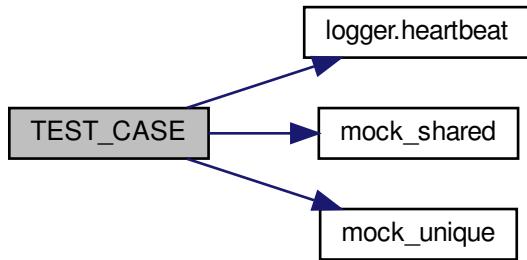
16.265.1.5 TEST_CASE() [5/7]

```
TEST_CASE (
    "OLD TEST: UDPInterface's tests." ,
    "" [UPDIInterface] )
```

Definition at line 781 of file [test_UDPInterface.cpp](#).

References [logger::heartbeat\(\)](#), [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



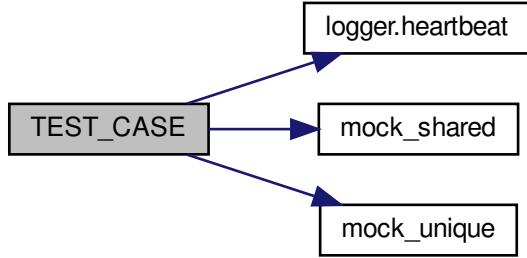
16.265.1.6 TEST_CASE() [6/7]

```
TEST_CASE (
    "OLD TEST: UDPInterface's 'receive_packet' method receives one or more MAVLink
packets." ,
    "" [UPDIInterface] )
```

Definition at line 815 of file [test_UDPInterface.cpp](#).

References [logger::heartbeat\(\)](#), [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



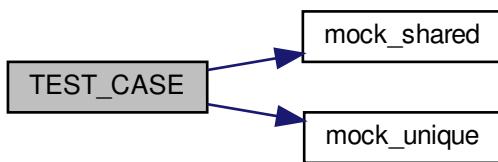
16.265.1.7 TEST_CASE() [7/7]

```
TEST_CASE (
    "OLD TEST: UDPInterface's 'send_packet' method sends one or more \" \" MAVLink packets."
    ""
    "[UPDInterface] )
```

Definition at line 867 of file [test_UDPInterface.cpp](#).

References [mock_shared\(\)](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



16.266 test_UDPInterface.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <algorithm>
00019 #include <chrono>
00020 #include <cstdint>
00021 #include <memory>
00022 #include <set>
00023 #include <stdexcept>
00024 #include <utility>
00025 #include <vector>
00026
00027 #include <catch.hpp>
00028 #include <fakeit.hpp>
00029
00030 #include "ConnectionFactory.hpp"
00031 #include "ConnectionPool.hpp"
00032 #include "Filter.hpp"
00033 #include "IPAddress.hpp"
00034 #include "Packet.hpp"
00035 #include "PacketVersion2.hpp"
00036 #include "UDPInterface.hpp"
00037 #include "UDPSocket.hpp"
00038 #include "utility.hpp"
00039
00040 #include "common.hpp"
00041 #include "common_Packet.hpp"
00042
00043
00044 using namespace std::chrono_literals;
00045
00046
00047 TEST_CASE("UDPInterface's can be constructed.", "[UDPInterface]")
00048 {
00049     fakeit::Mock<UDPSocket> mock_socket;
00050     fakeit::Mock<ConnectionPool> mock_pool;
00051     fakeit::Mock<ConnectionFactory>> mock_factory;
00052     auto socket = mock_unique(mock_socket);
00053     auto pool = mock_shared(mock_pool);
00054     auto factory = mock_unique(mock_factory);
00055     SECTION("When all inputs are valid.")
00056     {
00057         REQUIRE_NO_THROW(
00058             UDPInterface(std::move(socket), pool, std::move(factory)));
00059     }
00060     SECTION("Ensures the socket pointer is not null.")
00061     {
00062         REQUIRE_THROWS_AS(
00063             UDPInterface(nullptr, pool, std::move(factory)),
00064             std::invalid_argument);
00065         factory = mock_unique(mock_factory);
00066         REQUIRE_THROWS_WITH(
00067             UDPInterface(nullptr, pool, std::move(factory)),
00068             "Given socket pointer is null.");
00069     }
00070     SECTION("Ensures the connection pool pointer is not null.")
00071     {
00072         REQUIRE_THROWS_AS(
00073             UDPInterface(std::move(socket), nullptr, std::move(factory)),
00074             std::invalid_argument);
00075         socket = mock_unique(mock_socket);
00076         factory = mock_unique(mock_factory);
00077         REQUIRE_THROWS_WITH(

```

```

00078         UDPInterface(std::move(socket), nullptr, std::move(factory)),
00079         "Given connection pool pointer is null.");
00080     }
00081     SECTION("Ensures the connection factory pointer is not null.")
00082     {
00083         REQUIRE_THROWS_AS(
00084             UDPInterface(std::move(socket), pool, nullptr),
00085             std::invalid_argument);
00086         socket = mock_unique(mock_socket);
00087         REQUIRE_THROWS_WITH(
00088             UDPInterface(std::move(socket), pool, nullptr),
00089             "Given connection factory pointer is null.");
00090     }
00091 }
00092
00093
00094 TEST_CASE("UDPInterface's 'receive_packet' method.", "[UDPInterface]")
00095 {
00096     // MAVLink packets.
00097     auto heartbeat =
00098         std::make_shared<packet_v2::Packet>(to_vector(HeartbeatV2()));
00099     auto mission_set_current =
00100         std::make_shared<packet_v2::Packet>(to_vector(MissionSetCurrentV2()));
00101     auto encapsulated_data =
00102         std::make_shared<packet_v2::Packet>(to_vector(EncapsulatedDataV2()));
00103     // Connection pool
00104     ConnectionPool pool_obj;
00105     fakeit::Mock<ConnectionPool> spy_pool(pool_obj);
00106     fakeit::Spy(Method(spy_pool, add));
00107     auto pool = mock_shared(spy_pool);
00108     // Filter
00109     fakeit::Mock<Filter> mock_filter;
00110     std::multiset<packet_v2::Packet,
00111         bool(*)(const packet_v2::Packet &, const
00112         packet_v2::Packet &)>
00113         will_accept_packets([](const auto & a, const auto & b)
00114     {
00115         return a.data() < b.data();
00116     });
00117     std::multiset<MAVAddress> will_accept_addresses;
00118     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00119         [&](auto & a, auto & b)
00120     {
00121         will_accept_packets.insert(
00122             dynamic_cast<const packet_v2::Packet &>(a));
00123         will_accept_addresses.insert(b);
00124         return std::pair<bool, int>(true, 0);
00125     });
00126     auto filter = mock_shared(mock_filter);
00127     // Socket
00128     using receive_type =
00129         IPAddress(std::back_insert_iterator<std::vector<uint8_t>>,
00130                     const std::chrono::nanoseconds &);
00131     UDPSocket udp_socket;
00132     fakeit::Mock<UDPSocket> mock_socket(udp_socket);
00133     auto socket = mock_unique(mock_socket);
00134     // Interface
00135     UDPInterface udp(
00136         std::move(socket), pool,
00137         std::make_unique<ConnectionFactory>(filter));
00138     SECTION("No packet received.")
00139     {
00140         // Mocks
00141         fakeit::When(
00142             OverloadedMethod(
00143                 mock_socket, receive, receive_type)).AlwaysReturn(IPAddress(0));
00144         // Test
00145         udp.receive_packet(timeout);
00146         // Verification
00147         fakeit::Verify(
00148             OverloadedMethod(mock_socket, receive, receive_type).Matching(
00149                 [&](auto a, auto b)
00150             {
00151                 (void)a;
00152                 return b == 250ms;
00153             }).Once());
00154         fakeit::Verify(Method(mock_filter, will_accept)).Exactly(0);
00155         fakeit::Verify(Method(spy_pool, add)).Exactly(0);
00156     }
00157     SECTION("Partial packet received.")

```

```

00158    {
00159        // Mocks
00160        fakeit::When(OverloadedMethod(mock_socket, receive, receive_type)
00161            ).Do([](auto a, auto b)
00162        {
00163            // Load 127.1 mavlink address into connection.
00164            (void)b;
00165            auto vec = to_vector(HeartbeatV2());
00166            std::copy(vec.begin(), vec.end(), a);
00167            return IPAddress("127.0.0.1:4000");
00168        }).Do([](auto a, auto b)
00169        {
00170            (void)b;
00171            auto vec = to_vector(EncapsulatedDataV2());
00172            std::copy(vec.begin(), vec.end() - 1, a);
00173            return IPAddress("127.0.0.1:4000");
00174        });
00175        // Test
00176        udp.receive_packet(timeout);
00177        udp.receive_packet(timeout);
00178        // Verification
00179        fakeit::Verify(
00180            OverloadedMethod(mock_socket, receive, receive_type).Matching(
00181                [&](auto a, auto b)
00182            {
00183                (void)a;
00184                return b == 250ms;
00185            }).Exactly(2);
00186        fakeit::Verify(Method(mock_filter, will_accept)).Exactly(0);
00187        fakeit::Verify(Method(spy_pool, add)).Exactly(1);
00188    }
00189 SECTION("Full packet received.")
00190 {
00191     // Mocks
00192     fakeit::When(OverloadedMethod(mock_socket, receive, receive_type)
00193         ).Do([](auto a, auto b)
00194     {
00195         // Load 127.1 mavlink address into connection.
00196         (void)b;
00197         auto vec = to_vector(HeartbeatV2());
00198         std::copy(vec.begin(), vec.end(), a);
00199         return IPAddress("127.0.0.1:4000");
00200     }).Do([](auto a, auto b)
00201     {
00202         (void)b;
00203         auto vec = to_vector(EncapsulatedDataV2());
00204         std::copy(vec.begin(), vec.end(), a);
00205         return IPAddress("127.0.0.1:4001");
00206     });
00207     // Test
00208     udp.receive_packet(timeout);
00209     udp.receive_packet(timeout);
00210     // Verification
00211     fakeit::Verify(
00212         OverloadedMethod(mock_socket, receive, receive_type).Matching(
00213             [&](auto a, auto b)
00214         {
00215             (void)a;
00216             return b == 250ms;
00217         }).Exactly(2);
00218     fakeit::Verify(Method(mock_filter, will_accept)).Once();
00219     REQUIRE(will_accept_packets.count(*encapsulated_data) == 1);
00220     REQUIRE(will_accept_addresses.count(MAVAddress("127.1")) == 1);
00221     fakeit::Verify(Method(spy_pool, add)).Exactly(2);
00222     auto it = will_accept_packets.find(*encapsulated_data);
00223     REQUIRE(it != will_accept_packets.end());
00224     REQUIRE(it->connection() != nullptr);
00225 }
00226 SECTION("Multiple packets received (same IP and MAVLink addresses).")
00227 {
00228     // Mocks
00229     fakeit::When(OverloadedMethod(mock_socket, receive, receive_type)
00230         ).Do([](auto a, auto b)
00231     {
00232         // Load 127.1 mavlink address into connection.
00233         (void)b;
00234         auto vec = to_vector(HeartbeatV2());
00235         std::copy(vec.begin(), vec.end(), a);
00236         return IPAddress("127.0.0.1:4000");
00237     }).AlwaysDo([](auto a, auto b)
00238     {

```

```
00239         (void)b;
00240         auto vec = to_vector(EncapsulatedDataV2());
00241         std::copy(vec.begin(), vec.end(), a);
00242         return IPAddress("127.0.0.1:4001");
00243     });
00244     // Test
00245     udp.receive_packet(timeout);
00246     udp.receive_packet(timeout);
00247     udp.receive_packet(timeout);
00248     // Verification
00249     fakeit::Verify(
00250         OverloadedMethod(mock_socket, receive, receive_type).Matching(
00251             [&](auto a, auto b)
00252         {
00253             (void)a;
00254             return b == 250ms;
00255         }).Exactly(3);
00256     fakeit::Verify(Method(mock_filter, will_accept)).Exactly(2);
00257     REQUIRE(will_accept_packets.count(*encapsulated_data) == 2);
00258     REQUIRE(will_accept_addresses.count(MAVAddress("127.1")) == 2);
00259     fakeit::Verify(Method(spy_pool, add)).Exactly(2);
00260     auto it = will_accept_packets.find(*encapsulated_data);
00261     REQUIRE(it != will_accept_packets.end());
00262     REQUIRE(it->connection() != nullptr);
00263     it++;
00264     REQUIRE(it != will_accept_packets.end());
00265     REQUIRE(it->connection() != nullptr);
00266 }
00267 SECTION("Multiple packets received (same IP, different MAVLink addresses).")
00268 {
00269     // Mocks
00270     fakeit::When(OverloadedMethod(mock_socket, receive, receive_type)
00271         .Do([](auto a, auto b)
00272     {
00273         // Load 127.1 mavlink address into connection.
00274         (void)b;
00275         auto vec = to_vector(HeartbeatV2());
00276         std::copy(vec.begin(), vec.end(), a);
00277         return IPAddress("127.0.0.1:4000");
00278     }).Do([](auto a, auto b)
00279     {
00280         (void)b;
00281         auto vec = to_vector(EncapsulatedDataV2());
00282         std::copy(vec.begin(), vec.end(), a);
00283         return IPAddress("127.0.0.1:4001");
00284     }).Do([](auto a, auto b)
00285     {
00286         (void)b;
00287         auto vec = to_vector(MissionSetCurrentV2());
00288         std::copy(vec.begin(), vec.end(), a);
00289         return IPAddress("127.0.0.1:4001");
00290     });
00291     // Test
00292     udp.receive_packet(timeout);
00293     udp.receive_packet(timeout);
00294     udp.receive_packet(timeout);
00295     // Verification
00296     fakeit::Verify(
00297         OverloadedMethod(mock_socket, receive, receive_type).Matching(
00298             [&](auto a, auto b)
00299         {
00300             (void)a;
00301             return b == 250ms;
00302         }).Exactly(3);
00303     fakeit::Verify(Method(mock_filter, will_accept)).Exactly(2);
00304     REQUIRE(will_accept_packets.count(*encapsulated_data) == 1);
00305     REQUIRE(will_accept_packets.count(*mission_set_current) == 1);
00306     REQUIRE(will_accept_addresses.count(MAVAddress("127.1")) == 2);
00307     fakeit::Verify(Method(spy_pool, add)).Exactly(2);
00308     auto it = will_accept_packets.find(*encapsulated_data);
00309     REQUIRE(it != will_accept_packets.end());
00310     REQUIRE(it->connection() != nullptr);
00311     it = will_accept_packets.find(*mission_set_current);
00312     REQUIRE(it != will_accept_packets.end());
00313     REQUIRE(it->connection() != nullptr);
00314 }
00315 SECTION("Multiple packets received (different IP and MAVLink addresses).")
00316 {
00317     // Mocks
00318     fakeit::When(OverloadedMethod(mock_socket, receive, receive_type)
00319         .Do([](auto a, auto b)
```

```

00320      {
00321          // Load 127.1 mavlink address into connection.
00322          (void)b;
00323          auto vec = to_vector(HeartbeatV2());
00324          std::copy(vec.begin(), vec.end(), a);
00325          return IPAddress("127.0.0.1:4000");
00326      }).Do([](auto a, auto b)
00327      {
00328          (void)b;
00329          auto vec = to_vector(EncapsulatedDataV2());
00330          std::copy(vec.begin(), vec.end(), a);
00331          return IPAddress("127.0.0.1:4001");
00332      }).Do([](auto a, auto b)
00333      {
00334          (void)b;
00335          auto vec = to_vector(MissionSetCurrentV2());
00336          std::copy(vec.begin(), vec.end(), a);
00337          return IPAddress("127.0.0.1:4002");
00338      });
00339      // Test
00340      udp.receive_packet(timeout);
00341      udp.receive_packet(timeout);
00342      udp.receive_packet(timeout);
00343      // Verification
00344      fakeit::Verify(
00345          OverloadedMethod(mock_socket, receive, receive_type).Matching(
00346              [&](auto a, auto b)
00347          {
00348              (void)a;
00349              return b == 250ms;
00350          }).Exactly(3);
00351      fakeit::Verify(Method(mock_filter, will_accept)).Exactly(3);
00352      REQUIRE(will_accept_packets.count(*encapsulated_data) == 1);
00353      REQUIRE(will_accept_packets.count(*mission_set_current) == 2);
00354      REQUIRE(will_accept_addresses.count(MAVAddress("127.1")) == 2);
00355      REQUIRE(will_accept_addresses.count(MAVAddress("224.255")) == 1);
00356      fakeit::Verify(Method(spy_pool, add)).Exactly(3);
00357      auto it = will_accept_packets.find(*encapsulated_data);
00358      REQUIRE(it != will_accept_packets.end());
00359      REQUIRE(it->connection() != nullptr);
00360      it = will_accept_packets.find(*mission_set_current);
00361      REQUIRE(it != will_accept_packets.end());
00362      REQUIRE(it->connection() != nullptr);
00363      it++;
00364      REQUIRE(it != will_accept_packets.end());
00365      REQUIRE(it->connection() != nullptr);
00366  }
00367 SECTION("Partial packets with same IP address should be combined and "
00368         "parsed.")
00369 {
00370     // Mocks
00371     fakeit::When(OverloadedMethod(mock_socket, receive, receive_type)
00372                 .Do([](auto a, auto b)
00373                 {
00374                     // Load 127.1 mavlink address into connection.
00375                     (void)b;
00376                     auto vec = to_vector(HeartbeatV2());
00377                     std::copy(vec.begin(), vec.end(), a);
00378                     return IPAddress("127.0.0.1:4000");
00379                 }).Do([](auto a, auto b)
00380                 {
00381                     (void)b;
00382                     auto vec = to_vector(EncapsulatedDataV2());
00383                     std::copy(vec.begin(), vec.end() - 10, a);
00384                     return IPAddress("127.0.0.1:4001");
00385                 }).Do([](auto a, auto b)
00386                 {
00387                     (void)b;
00388                     auto vec = to_vector(EncapsulatedDataV2());
00389                     std::copy(vec.end() - 10, vec.end(), a);
00390                     return IPAddress("127.0.0.1:4001");
00391                 });
00392     // Test
00393     udp.receive_packet(timeout);
00394     udp.receive_packet(timeout);
00395     udp.receive_packet(timeout);
00396     // Verification
00397     fakeit::Verify(
00398         OverloadedMethod(mock_socket, receive, receive_type).Matching(
00399             [&](auto a, auto b)
00400         {

```

```

00401         (void)a;
00402         return b == 250ms;
00403     }).Exactly(3);
00404     fakeit::Verify(Method(mock_filter, will_accept)).Once();
00405     REQUIRE(will_accept_packets.count(*encapsulated_data) == 1);
00406     REQUIRE(will_accept_addresses.count(MAVAddress("127.1")) == 1);
00407     fakeit::Verify(Method(spy_pool, add)).Exactly(2);
00408     auto it = will_accept_packets.find(*encapsulated_data);
00409     REQUIRE(it != will_accept_packets.end());
00410     REQUIRE(it->connection() != nullptr);
00411 }
00412 SECTION("Partial packets with different IP addresses should be dropped.")
00413 {
00414     // Mocks
00415     fakeit::When(OverloadedMethod(mock_socket, receive, receive_type)
00416                 .Do([](auto a, auto b)
00417                 {
00418                     // Load 127.1 mavlink address into connection.
00419                     (void)b;
00420                     auto vec = to_vector(HeartbeatV2());
00421                     std::copy(vec.begin(), vec.end(), a);
00422                     return IPAddress("127.0.0.1:4000");
00423                 }).Do([](auto a, auto b)
00424                 {
00425                     (void)b;
00426                     auto vec = to_vector(EncapsulatedDataV2());
00427                     std::copy(vec.begin(), vec.end() - 10, a);
00428                     return IPAddress("127.0.0.1:4001");
00429                 }).Do([](auto a, auto b)
00430                 {
00431                     (void)b;
00432                     auto vec = to_vector(EncapsulatedDataV2());
00433                     std::copy(vec.end() - 10, vec.end(), a);
00434                     return IPAddress("127.0.0.1:4002");
00435                 });
00436     // Test
00437     udp.receive_packet(timeout);
00438     udp.receive_packet(timeout);
00439     udp.receive_packet(timeout);
00440     // Verification
00441     fakeit::Verify(
00442         OverloadedMethod(mock_socket, receive, receive_type).Matching(
00443             [&](auto a, auto b)
00444             {
00445                 (void)a;
00446                 return b == 250ms;
00447             }).Exactly(3);
00448     fakeit::Verify(Method(mock_filter, will_accept)).Exactly(0);
00449     fakeit::Verify(Method(spy_pool, add)).Once();
00450 }
00451 }
00452
00453
00454 TEST_CASE("UDPIterace's 'send_packet' method.", "[UDPIterace]")
00455 {
00456     // MAVLink packets.
00457     auto ping = std::make_shared<packet_v2::Packet>(to_vector(PingV2()));
00458     auto heartbeat =
00459         std::make_shared<packet_v2::Packet>(to_vector(HeartbeatV2()));
00460     auto encapsulated_data =
00461         std::make_shared<packet_v2::Packet>(to_vector(EncapsulatedDataV2()));
00462     // Filter
00463     fakeit::Mock<Filter> mock_filter;
00464     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00465         [&](auto & a, auto & b)
00466         {
00467             (void)a;
00468             (void)b;
00469             return std::pair<bool, int>(true, 0);
00470         });
00471     auto filter = mock_shared(mock_filter);
00472     // Socket
00473     std::multiset<std::vector<uint8_t>> send_bytes;
00474     std::multiset<IPAddress> send_addresses;
00475     using receive_type =
00476         IPAddress(std::back_inserter<std::vector<uint8_t>>,
00477                    const std::chrono::nanoseconds &);
00478     using send_type =
00479         void(std::vector<uint8_t>::const_iterator,
00480              std::vector<uint8_t>::const_iterator,
00481              const IPAddress &);

```

```

00482     UDPSocket udp_socket;
00483     fakeit::Mock<UDPSocket> mock_socket(udp_socket);
00484     fakeit::When(OverloadedMethod(mock_socket, send, send_type)
00485                 .AlwaysDo([&](auto a, auto b, auto c)
00486     {
00487         std::vector<uint8_t> vec;
00488         std::copy(a, b, std::back_inserter(vec));
00489         send_bytes.insert(vec);
00490         send_addresses.insert(c);
00491     });
00492     auto socket = mock_unique(mock_socket);
00493     // Connection Factory
00494     ConnectionFactory<> factory_obj(filter);
00495     fakeit::Mock<ConnectionFactory>> spy_factory(factory_obj);
00496     fakeit::Spy(Method(spy_factory, wait_for_packet));
00497     // Interface
00498     UDPInterface udp(
00499         std::move(socket),
00500         std::make_shared<ConnectionPool>(),
00501         mock_unique(spy_factory));
00502     std::chrono::nanoseconds timeout = 1ms;
00503     SECTION("No packets, timeout.")
00504     {
00505         // Test
00506         udp.send_packet(timeout);
00507         // Verification
00508         fakeit::Verify(Method(spy_factory, wait_for_packet).Using(1ms)).Once();
00509         fakeit::Verify(
00510             OverloadedMethod(mock_socket, send, send_type)).Exactly(0);
00511     }
00512     SECTION("Single connection, single packet.")
00513     {
00514         // Mocks
00515         fakeit::When(OverloadedMethod(mock_socket, receive, receive_type)
00516                     .Do([](auto a, auto b)
00517         {
00518             // Load 127.1 mavlink address into connection.
00519             (void)b;
00520             auto vec = to_vector(HeartbeatV2());
00521             std::copy(vec.begin(), vec.end(), a);
00522             return IPAddress("127.0.0.1:4000");
00523         }).Do([](auto a, auto b)
00524         {
00525             (void)b;
00526             auto vec = to_vector(EncapsulatedDataV2());
00527             std::copy(vec.begin(), vec.end(), a);
00528             return IPAddress("127.0.0.1:4001");
00529         });
00530         // Test
00531         udp.receive_packet(timeout);
00532         udp.receive_packet(timeout);
00533         udp.send_packet(timeout);
00534         // Verification
00535         fakeit::Verify(Method(spy_factory, wait_for_packet).Using(1ms)).Once();
00536         fakeit::Verify(OverloadedMethod(mock_socket, send, send_type)).Once();
00537         REQUIRE(send_bytes.size() == 1);
00538         REQUIRE(send_bytes.count(to_vector(EncapsulatedDataV2())) == 1);
00539         REQUIRE(send_addresses.count(IPAddress("127.0.0.1:4000")) == 1);
00540     }
00541     SECTION("Single connection, multiple packets.")
00542     {
00543         // Mocks
00544         fakeit::When(OverloadedMethod(mock_socket, receive, receive_type)
00545                     .Do([](auto a, auto b)
00546         {
00547             // Load 127.1 mavlink address into connection.
00548             (void)b;
00549             auto vec = to_vector(HeartbeatV2());
00550             std::copy(vec.begin(), vec.end(), a);
00551             return IPAddress("127.0.0.1:4000");
00552         }).Do([](auto a, auto b)
00553         {
00554             (void)b;
00555             auto vec = to_vector(EncapsulatedDataV2());
00556             std::copy(vec.begin(), vec.end(), a);
00557             return IPAddress("127.0.0.1:4001");
00558         }).Do([](auto a, auto b)
00559         {
00560             (void)b;
00561             auto vec = to_vector(MissionSetCurrentV2());
00562             std::copy(vec.begin(), vec.end(), a);

```

```
00563         return IPAddress("127.0.0.1:4001");
00564     });
00565     // Test
00566     udp.receive_packet(timeout);
00567     udp.receive_packet(timeout);
00568     udp.receive_packet(timeout);
00569     udp.send_packet(timeout);
00570     // Verification
00571     fakeit::Verify(Method(spy_factory, wait_for_packet).Using(1ms)).Once();
00572     fakeit::Verify(OverloadedMethod(mock_socket, send, send_type)).Once();
00573     REQUIRE(send_bytes.size() == 1);
00574     REQUIRE(send_bytes.count(to_vector(EncapsulatedDataV2())) == 1);
00575     REQUIRE(send_addresses.count(IPAddress("127.0.0.1:4000")) == 1);
00576     // Test
00577     udp.send_packet(timeout);
00578     // Verification
00579     fakeit::Verify(
00580         Method(spy_factory, wait_for_packet).Using(1ms).Exactly(2));
00581     fakeit::Verify(
00582         OverloadedMethod(mock_socket, send, send_type)).Exactly(2);
00583     REQUIRE(send_bytes.size() == 2);
00584     REQUIRE(send_bytes.count(to_vector(EncapsulatedDataV2())) == 1);
00585     REQUIRE(send_bytes.count(to_vector(MissionSetCurrentV2())) == 1);
00586     REQUIRE(send_addresses.count(IPAddress("127.0.0.1:4000")) == 2);
00587 }
00588 SECTION("Multiple connections, multiple packets.")
00589 {
00590     // Mocks
00591     fakeit::When(OverloadedMethod(mock_socket, receive, receive_type))
00592         .Do([](auto a, auto b)
00593     {
00594         // Load 127.1 mavlink address into connection.
00595         (void)b;
00596         auto vec = to_vector(HeartbeatV2());
00597         std::copy(vec.begin(), vec.end(), a);
00598         return IPAddress("127.0.0.1:4000");
00599     }).Do([](auto a, auto b)
00600     {
00601         (void)b;
00602         auto vec = to_vector(EncapsulatedDataV2());
00603         std::copy(vec.begin(), vec.end(), a);
00604         return IPAddress("127.0.0.1:4001");
00605     }).Do([](auto a, auto b)
00606     {
00607         (void)b;
00608         auto vec = to_vector(MissionSetCurrentV2());
00609         std::copy(vec.begin(), vec.end(), a);
00610         return IPAddress("127.0.0.1:4002");
00611     });
00612     // Test
00613     udp.receive_packet(timeout);
00614     udp.receive_packet(timeout);
00615     udp.receive_packet(timeout);
00616     udp.send_packet(timeout);
00617     // Verification
00618     fakeit::Verify(Method(spy_factory, wait_for_packet).Using(1ms)).Once();
00619     fakeit::Verify(Method(spy_factory, wait_for_packet)).Exactly(2);
00620     fakeit::Verify(
00621         OverloadedMethod(mock_socket, send, send_type)).Exactly(2);
00622     REQUIRE(send_bytes.size() == 2);
00623     REQUIRE(send_bytes.count(to_vector(EncapsulatedDataV2())) == 1);
00624     REQUIRE(send_bytes.count(to_vector(MissionSetCurrentV2())) == 1);
00625     REQUIRE(send_addresses.count(IPAddress("127.0.0.1:4000")) == 1);
00626     REQUIRE(send_addresses.count(IPAddress("127.0.0.1:4001")) == 1);
00627     // Test
00628     udp.send_packet(timeout);
00629     // Verification
00630     fakeit::Verify(
00631         Method(spy_factory, wait_for_packet).Using(1ms).Exactly(2));
00632     fakeit::Verify(Method(spy_factory, wait_for_packet)).Exactly(3);
00633     fakeit::Verify(
00634         OverloadedMethod(mock_socket, send, send_type)).Exactly(3);
00635     REQUIRE(send_bytes.size() == 3);
00636     REQUIRE(send_bytes.count(to_vector(EncapsulatedDataV2())) == 1);
00637     REQUIRE(send_bytes.count(to_vector(MissionSetCurrentV2())) == 2);
00638     REQUIRE(send_addresses.count(IPAddress("127.0.0.1:4000")) == 2);
00639     REQUIRE(send_addresses.count(IPAddress("127.0.0.1:4001")) == 1);
00640 }
00641 SECTION("Multiple connections with broadcast packet.")
00642 {
00643     // Mocks
```

```

00644     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00645         [&](auto & a, auto & b)
00646     {
00647         (void)b;
00648
00649         if (a.name() == "MISSION_SET_CURRENT")
00650         {
00651             return std::pair<bool, int>(true, 0);
00652         }
00653
00654         return std::pair<bool, int>(false, 0);
00655     });
00656     fakeit::When(OverloadedMethod(mock_socket, receive, receive_type))
00657         .Do([](auto a, auto b)
00658     {
00659         // Load 127.1 mavlink address into first connection.
00660         (void)b;
00661         auto vec = to_vector(HeartbeatV2());
00662         std::copy(vec.begin(), vec.end(), a);
00663         return IPAddress("127.0.0.1:4000");
00664     }).Do([](auto a, auto b)
00665     {
00666         // Load 224.255 mavlink address into second connection.
00667         (void)b;
00668         auto vec = to_vector(EncapsulatedDataV2());
00669         std::copy(vec.begin(), vec.end(), a);
00670         return IPAddress("127.0.0.1:4001");
00671     }).Do([](auto a, auto b)
00672     {
00673         (void)b;
00674         auto vec = to_vector(MissionSetCurrentV2());
00675         std::copy(vec.begin(), vec.end(), a);
00676         return IPAddress("127.0.0.1:4002");
00677     });
00678     // Test
00679     udp.receive_packet(timeout);
00680     udp.receive_packet(timeout);
00681     udp.receive_packet(timeout);
00682     udp.send_packet(timeout);
00683     // Verification
00684     fakeit::Verify(Method(spy_factory, wait_for_packet).Using(lms)).Once();
00685     fakeit::Verify(Method(spy_factory, wait_for_packet)).Exactly(2);
00686     fakeit::Verify(
00687         OverloadedMethod(mock_socket, send, send_type)).Exactly(2);
00688     REQUIRE(send_bytes.size() == 2);
00689     REQUIRE(send_bytes.count(to_vector(MissionSetCurrentV2())) == 2);
00690     REQUIRE(send_addresses.count(IPAddress("127.0.0.1:4000")) == 1);
00691     REQUIRE(send_addresses.count(IPAddress("127.0.0.1:4001")) == 1);
00692     // Test
00693     udp.send_packet(timeout);
00694     // Verification (no futher operations)
00695     fakeit::Verify(
00696         Method(spy_factory, wait_for_packet).Using(lms)).Exactly(2);
00697     fakeit::Verify(Method(spy_factory, wait_for_packet)).Exactly(3);
00698     fakeit::Verify(
00699         OverloadedMethod(mock_socket, send, send_type)).Exactly(2);
00700 }
00701 SECTION("Multiple connections with targeted packet.")
00702 {
00703     // Mocks
00704     fakeit::When(Method(mock_filter, will_accept)).AlwaysDo(
00705         [&](auto & a, auto & b)
00706     {
00707         (void)b;
00708
00709         if (a.name() == "PING")
00710         {
00711             return std::pair<bool, int>(true, 0);
00712         }
00713
00714         return std::pair<bool, int>(false, 0);
00715     });
00716     fakeit::When(OverloadedMethod(mock_socket, receive, receive_type))
00717         .Do([](auto a, auto b)
00718     {
00719         // Load 127.1 mavlink address into first connection.
00720         (void)b;
00721         auto vec = to_vector(HeartbeatV2());
00722         std::copy(vec.begin(), vec.end(), a);
00723         return IPAddress("127.0.0.1:4000");
00724     }).Do([](auto a, auto b)

```

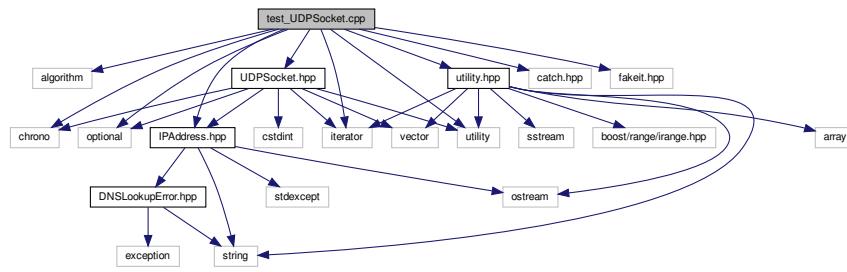
```
00725     {
00726         // Load 224.255 mavlink address into second connection.
00727         (void)b;
00728         auto vec = to_vector(EncapsulatedDataV2());
00729         std::copy(vec.begin(), vec.end(), a);
00730         return IPAddress("127.0.0.1:4001");
00731     }).Do([](auto a, auto b)
00732     {
00733         (void)b;
00734         auto vec = to_vector(PingV2());
00735         std::copy(vec.begin(), vec.end(), a);
00736         return IPAddress("127.0.0.1:4002");
00737     });
00738     // Test
00739     udp.receive_packet(timeout);
00740     udp.receive_packet(timeout);
00741     udp.receive_packet(timeout);
00742     udp.send_packet(timeout);
00743     // Verification
00744     fakeit::Verify(Method(spy_factory, wait_for_packet).Using(lms)).Once();
00745     fakeit::Verify(Method(spy_factory, wait_for_packet)).Once();
00746     fakeit::Verify(
00747         OverloadedMethod(mock_socket, send, send_type)).Once();
00748     REQUIRE(send_bytes.size() == 1);
00749     REQUIRE(send_bytes.count(to_vector(PingV2())) == 1);
00750     REQUIRE(send_addresses.count(IPAddress("127.0.0.1:4000")) == 1);
00751     // Test
00752     udp.send_packet(timeout);
00753     // Verification (no futher operations)
00754     fakeit::Verify(
00755         Method(spy_factory, wait_for_packet).Using(lms)).Exactly(2);
00756     fakeit::Verify(Method(spy_factory, wait_for_packet)).Exactly(2);
00757     fakeit::Verify(
00758         OverloadedMethod(mock_socket, send, send_type)).Once();
00759 }
00760 }
00761
00762
00763 TEST_CASE("UDPInterface's are printable.", "[UDPInterface]")
00764 {
00765     fakeit::Mock<ConnectionPool> mock_pool;
00766     fakeit::Mock<ConnectionFactory>> mock_factory;
00767     auto pool = mock_shared(mock_pool);
00768     auto factory = mock_unique(mock_factory);
00769     auto socket = std::make_unique<UDPSocket>();
00770     UDPInterface udp(std::move(socket), pool, std::move(factory));
00771     REQUIRE(str(udp) == "unknown UDP socket");
00772 }
00773
00774
00775 // The tests below are from attempts to test the UDPInterface class in different
00776 // ways, all failing to provide a method to test the entire class. They have
00777 // been kept because they could theoretically find problems not covered by the
00778 // tests above and do not cause much overhead.
00779 /////////////////////////////////
00780
00781 TEST_CASE("OLD TEST: UDPInterface's tests.", "[UDPInterface]")
00782 {
00783     using receive_type = IPAddress(
00784         std::back_insert_iterator<std::vector<uint8_t>>,
00785         const std::chrono::nanoseconds &);
00786     auto heartbeat =
00787         std::make_shared<packet_v2::Packet>(to_vector(HeartbeatV2()));
00788     fakeit::Mock<Filter> mock_filter;
00789     UDPSocket udp_socket;
00790     fakeit::Mock<UDPSocket> mock_socket(udp_socket);
00791     auto filter = mock_shared(mock_filter);
00792     auto socket = mock_unique(mock_socket);
00793     auto pool = std::make_shared<ConnectionPool>();
00794     SECTION("Timeout")
00795     {
00796         fakeit::When(OverloadedMethod(
00797             mock_socket, receive, receive_type)).AlwaysReturn(
00798             IPAddress(0));
00799         UDPInterface udp(
00800             std::move(socket), pool,
00801             std::make_unique<ConnectionFactory>>(filter));
00802         std::chrono::nanoseconds timeout = 250ms;
00803         udp.receive_packet(timeout);
00804         fakeit::Verify(
00805             OverloadedMethod(mock_socket, receive, receive_type).Matching(
```

```

00806             [&] (auto a, auto b)
00807         {
00808             (void)a;
00809             return b == 250ms;
00810         }).Once();
00811     }
00812 }
00813
00814
00815 TEST_CASE("OLD TEST: UDPInterface's 'receive_packet' method receives one or "
00816             "more MAVLink packets.", "[UPDInterface]")
00817 {
00818     using receive_type = IPAddress(
00819             std::back_insert_iterator<std::vector<uint8_t>>,
00820             const std::chrono::nanoseconds &);
00821     auto heartbeat =
00822         std::make_shared<packet_v2::Packet>(to_vector(HeartbeatV2()));
00823     fakeit::Mock<Filter> mock_filter;
00824     UDPSocket udp_socket;
00825     fakeit::Mock<UDPSocket> mock_socket(udp_socket);
00826     fakeit::Mock<ConnectionPool> mock_pool;
00827     fakeit::Mock<ConnectionFactory>> mock_factory;
00828     auto filter = mock_shared(mock_filter);
00829     auto socket = mock_unique(mock_socket);
00830     auto pool = mock_shared(mock_pool);
00831     auto factory = mock_unique(mock_factory);
00832     SECTION("When no packets available for sending.")
00833     {
00834         fakeit::When(OverloadedMethod(mock_socket, receive, receive_type)
00835                     .AlwaysDo([](auto a, auto b)
00836                     {
00837                         (void)b;
00838                         auto vec = to_vector(HeartbeatV2());
00839                         std::copy(vec.begin(), vec.end(), a);
00840                         return IPAddress("192.168.0.0");
00841                     }));
00842         fakeit::Fake(Method(mock_pool, send));
00843         fakeit::Fake(Method(mock_pool, add));
00844         fakeit::When(Method(mock_factory, get)).AlwaysDo([&](auto a)
00845         {
00846             return std::make_unique<Connection>(a, filter);
00847         });
00848         UDPInterface udp(std::move(socket), pool, std::move(factory));
00849         std::chrono::nanoseconds timeout = 250ms;
00850         udp.receive_packet(timeout);
00851         fakeit::Verify(
00852             OverloadedMethod(mock_socket, receive, receive_type).Matching(
00853                 [](auto a, auto c)
00854             {
00855                 (void)a;
00856                 return c == 250ms;
00857             }).Once());
00858         fakeit::Verify(Method(mock_pool, send).Matching(
00859             [&](auto & a)
00860             {
00861                 return a != nullptr && *a == *heartbeat;
00862             }).Once());
00863     }
00864 }
00865
00866
00867 TEST_CASE("OLD TEST: UDPInterface's 'send_packet' method sends one or more "
00868             "MAVLINK packets.", "[UPDInterface]")
00869 {
00870     UDPSocket udp_socket;
00871     fakeit::Mock<UDPSocket> mock_socket(udp_socket);
00872     fakeit::Mock<ConnectionPool> mock_pool;
00873     fakeit::Mock<ConnectionFactory>> mock_factory;
00874     auto socket = mock_unique(mock_socket);
00875     auto pool = mock_shared(mock_pool);
00876     auto factory = mock_unique(mock_factory);
00877     SECTION("When no packets available for sending.")
00878     {
00879         fakeit::When(Method(mock_factory, wait_for_packet)).AlwaysReturn(false);
00880         UDPInterface udp(std::move(socket), pool, std::move(factory));
00881         std::chrono::nanoseconds timeout = 250ms;
00882         udp.send_packet(timeout);
00883         fakeit::Verify(
00884             Method(mock_factory, wait_for_packet).Using(250ms)).Once();
00885     }
00886 }
```

16.267 test_UDPSocket.cpp File Reference

```
#include <algorithm>
#include <chrono>
#include <iterator>
#include <optional>
#include <utility>
#include <catch.hpp>
#include <fakeit.hpp>
#include "IPAddress.hpp"
#include "UDPSocket.hpp"
#include <utility.hpp>
Include dependency graph for test_UDPSocket.cpp:
```



Functions

- [TEST_CASE \("UDPSocket's 'send' method accepts a vector of bytes and an address.", "\[UDPSocket\]"\)](#)
- [TEST_CASE \("UDPSocket's 'send' method accepts two vector iterators and an " "address.", "\[UDPSocket\]"\)](#)
- [TEST_CASE \("UDPSocket's 'receive' method takes a timeout and returns a vector " "of bytes and the IP address that sent them.", "\[UDPSocket\]"\)](#)
- [TEST_CASE \("UDPSocket's 'receive' method takes a back inserter and a timeout " "and returns the IP address that sent the bytes written to the " "back inserter.", "\[UDPSocket\]"\)](#)
- [TEST_CASE \("UDPSocket's are printable.", "\[UDPSocket\]"\)](#)

16.267.1 Function Documentation

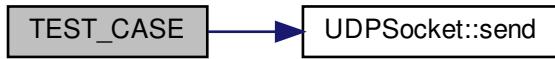
16.267.1.1 TEST_CASE() [1/5]

```
TEST_CASE (
    "UDPSocket's 'send' method accepts a vector of bytes and an address." ,
    "" [UDPSocket] )
```

Definition at line 35 of file [test_UDPSocket.cpp](#).

References [UDPSocket::send\(\)](#).

Here is the call graph for this function:



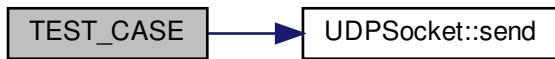
16.267.1.2 TEST_CASE() [2/5]

```
TEST_CASE (
    "UDPSocket's 'send' method accepts two vector iterators and an \"address.\" ,
    \" [UDPSocket] )
```

Definition at line 64 of file [test_UDPSocket.cpp](#).

References [UDPSocket::send\(\)](#).

Here is the call graph for this function:



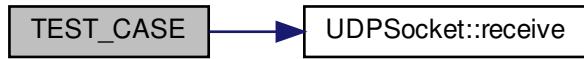
16.267.1.3 TEST_CASE() [3/5]

```
TEST_CASE (
    "UDPSocket's 'receive' method takes a timeout and returns a vector \"of bytes and
    the IP address that sent them.\" ,
    \" [UDPSocket] )
```

Definition at line 92 of file [test_UDPSocket.cpp](#).

References [UDPSocket::receive\(\)](#).

Here is the call graph for this function:



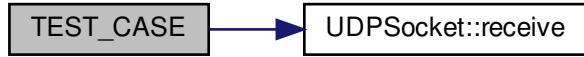
16.267.1.4 TEST_CASE() [4/5]

```
TEST_CASE (
    "UDPSocket's 'receive' method takes a back inserter and a timeout " "and returns the
IP address that sent the bytes written to the " "back inserter." ,
    "" [UDPSocket] )
```

Definition at line 129 of file [test_UDPSocket.cpp](#).

References [UDPSocket::receive\(\)](#).

Here is the call graph for this function:



16.267.1.5 TEST_CASE() [5/5]

```
TEST_CASE (
    "UDPSocket's are printable." ,
    "UDPSocket" )
```

Definition at line 163 of file [test_UDPSocket.cpp](#).

16.268 test_UDPSocket.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <algorithm>
00019 #include <chrono>
00020 #include <iterator>
00021 #include <optional>
00022 #include <utility>
00023
00024 #include <catch.hpp>
00025 #include <fakeit.hpp>
00026
00027 #include "IPAddress.hpp"
00028 #include "UDPSocket.hpp"
00029 #include <utility.hpp>
00030
00031
00032 using namespace std::chrono_literals;
00033
00034
00035 TEST_CASE("UDPSocket's 'send' method accepts a vector of bytes and an address.",
00036     "[UDPSocket]")
00037 {
00038     // This test ensures that one send method calls the other.
00039     UDPSocket udp;
00040     fakeit::Mock<UDPSocket> mock_socket(udp);
00041     fakeit::Fake(
00042         OverloadedMethod(
00043             mock_socket, send,
00044                 void(std::vector<uint8_t>::const_iterator,
00045                     std::vector<uint8_t>::const_iterator,
00046                     const IPAddress & address)));
00047     UDPSocket &socket = mock_socket.get();
00048     std::vector<uint8_t> vec = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
00049     IPAddress addr("192.168.0.0");
00050     socket.send(vec, addr);
00051     fakeit::Verify(
00052         OverloadedMethod(
00053             mock_socket, send,
00054                 void(std::vector<uint8_t>::const_iterator,
00055                     std::vector<uint8_t>::const_iterator,
00056                     const IPAddress & address)).Matching(
00057             [&](auto a, auto b, auto c)
00058         {
00059             return a == vec.begin() && b == vec.end() && c == addr;
00060         }).Once();
00061 }
00062
00063
00064 TEST_CASE("UDPSocket's 'send' method accepts two vector iterators and an "
00065     "address.", "[UDPSocket]")
00066 {
00067     // This test ensures that one send method calls the other.
00068     UDPSocket udp;
00069     fakeit::Mock<UDPSocket> mock_socket(udp);
00070     std::vector<uint8_t> send_vec;
00071     fakeit::When(
00072         OverloadedMethod(
00073             mock_socket, send,
00074                 void(const std::vector<uint8_t> &, const IPAddress &))).AlwaysDo(
00075             [&](auto a, auto b)
00076         {
00077             (void)b;

```

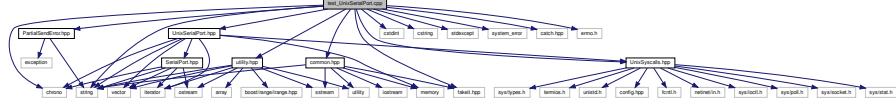
```
00078         send_vec = a;
00079     });
00080     UDPSocket &socket = mock_socket.get();
00081     std::vector<uint8_t> vec = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
00082     IPAddress addr("192.168.0.0");
00083     socket.send(vec.begin(), vec.end(), addr);
00084     fakeit::Verify(
00085         OverloadedMethod(
00086             mock_socket, send,
00087             void(const std::vector<uint8_t> &, const IPAddress &))).Once();
00088     REQUIRE(vec == send_vec);
00089 }
00090
00091
00092 TEST_CASE("UDPSocket's 'receive' method takes a timeout and returns a vector "
00093             "of bytes and the IP address that sent them.", "[UDPSocket]")
00094 {
00095     // This test ensures that one receive method calls the other.
00096     UDPSocket udp;
00097     fakeit::Mock<UDPSocket> mock_socket(udp);
00098     fakeit::When(
00099         OverloadedMethod(
00100             mock_socket, receive,
00101             IPAddress(std::back_insert_iterator<std::vector<uint8_t>>(),
00102                         const std::chrono::nanoseconds &))).AlwaysDo(
00103                 [](auto a, auto b)
00104     {
00105         (void)b;
00106         std::vector<uint8_t> vec = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
00107         std::copy(vec.begin(), vec.end(), a);
00108         return IPAddress("192.168.0.0");
00109     });
00110     UDPSocket &socket = mock_socket.get();
00111     std::vector<uint8_t> vec_compare = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
00112     std::chrono::nanoseconds timeout = 1ms;
00113     auto [data, addr] = socket.receive(timeout);
00114     REQUIRE(data == vec_compare);
00115     REQUIRE(addr == IPAddress("192.168.0.0"));
00116     fakeit::Verify(
00117         OverloadedMethod(
00118             mock_socket, receive,
00119             IPAddress(std::back_insert_iterator<std::vector<uint8_t>>(),
00120                         const std::chrono::nanoseconds &)).Matching(
00121                 [](auto a, auto b)
00122     {
00123         (void)a;
00124         return b == 1ms;
00125     })).Once();
00126 }
00127
00128
00129 TEST_CASE("UDPSocket's 'receive' method takes a back inserter and a timeout "
00130             "and returns the IP address that sent the bytes written to the "
00131             "back inserter.", "[UDPSocket]")
00132 {
00133     // This test ensures that one receive method calls the other.
00134     using receive_type = std::pair<std::vector<uint8_t>, IPAddress>(
00135         const std::chrono::nanoseconds &);
00136     UDPSocket udp;
00137     fakeit::Mock<UDPSocket> mock_socket(udp);
00138     fakeit::When(
00139         OverloadedMethod(
00140             mock_socket, receive, receive_type)).AlwaysDo([](auto a)
00141     {
00142         (void)a;
00143         std::vector<uint8_t> vec = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
00144         return std::pair<std::vector<uint8_t>, IPAddress>(
00145             vec, IPAddress("192.168.0.0"));
00146     });
00147     UDPSocket &socket = mock_socket.get();
00148     std::vector<uint8_t> vec_compare = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
00149     std::chrono::nanoseconds timeout = 1ms;
00150     std::vector<uint8_t> data;
00151     auto addr = socket.receive(std::back_inserter(data), timeout);
00152     REQUIRE(data == vec_compare);
00153     REQUIRE(addr == IPAddress("192.168.0.0"));
00154     fakeit::Verify(
00155         OverloadedMethod(
00156             mock_socket, receive, receive_type).Matching([](auto a)
00157         {
00158             return a == 1ms;
```

```
00159     })).Once();
00160 }
00161
00162
00163 TEST_CASE("UDPSocket's are printable.", "UDPSocket")
00164 {
00165     REQUIRE(str(UDPSocket()) == "unknown UDP socket");
00166 }
```

16.269 test_UnixSerialPort.cpp File Reference

```
#include <chrono>
#include <cstdint>
#include <cstring>
#include <stdexcept>
#include <string>
#include <system_error>
#include <catch.hpp>
#include <errno.h>
#include <fakeit.hpp>
#include "PartialSendError.hpp"
#include "UnixSerialPort.hpp"
#include "UnixSyscalls.hpp"
#include "utility.hpp"
#include "common.hpp"
Include dependency graph for test UnixSerial
```

Include dependency graph for test_UnixSerialPort.cpp:



Functions

- `TEST_CASE` ("UnixSerialPort's open and configure a serial port on construction" "and closes the port on destruction.", "[`UnixSerialPort`]"")
 - `TEST_CASE` ("UnixSerialPort's open method configures the baud rate.", "[`UnixSerialPort`]"")
 - `TEST_CASE` ("UnixSerialPort's 'read' method receives data on the socket.", "[`UnixSerialPort`]"")
 - `TEST_CASE` ("UnixSerialPort's 'write' method sends data over the serial port.", "[`UnixSerialPort`]"")
 - `TEST_CASE` ("UnixSerialPort's are printable.", "[`UnixSerialPort`]"")

16.269.1 Function Documentation

16.269.1.1 TEST_CASE() [1/5]

```
TEST_CASE (
    "UnixSerialPort's open and configure a serial port on construction" "and closes the
port on destruction." ,
    "" [UnixSerialPort] )
```

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

16.269.1.2 TEST_CASE() [2/5]

```
TEST_CASE (
    "UnixSerialPort's open method configures the baud rate." ,
    "" [UnixSerialPort] )
```

Definition at line 303 of file [test_UinxSerialPort.cpp](#).

16.269.1.3 TEST_CASE() [3/5]

```
TEST_CASE (
    "UnixSerialPort's 'read' method receives data on the socket." ,
    "" [UnixSerialPort] )
```

Definition at line 524 of file [test_UinxSerialPort.cpp](#).

References [SerialPort::DEFAULT](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



16.269.1.4 TEST_CASE() [4/5]

```
TEST_CASE (
    "UnixSerialPort's 'write' method sends data over the serial port." ,
    "" [UnixSerialPort] )
```

Definition at line 742 of file [test_UinxSerialPort.cpp](#).

References [SerialPort::DEFAULT](#), and [mock_unique\(\)](#).

Here is the call graph for this function:



16.269.1.5 TEST_CASE() [5/5]

```
TEST_CASE (
    "UnixSerialPort's are printable." ,
    "" [UnixSerialPort] )
```

Definition at line 819 of file [test_UinxSerialPort.cpp](#).

16.270 test_UinxSerialPort.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <chrono>
00019 #include <cstdint>
00020 #include <cstring>
00021 #include <stdexcept>
```

```
00022 #include <string>
00023 #include <system_error>
00024
00025 #include <catch.hpp>
00026 #include <errno.h>
00027 #include <fakeit.hpp>
00028
00029 #include "PartialSendError.hpp"
00030 #include "UnixSerialPort.hpp"
00031 #include "UnixSyscalls.hpp"
00032 #include "utility.hpp"
00033
00034 #include "common.hpp"
00035
00036
00037 using namespace std::chrono_literals;
00038
00039
00040 TEST_CASE("UnixSerialPort's open and configure a serial port on construction"
00041             "and closes the port on destruction.", "[UnixSerialPort]")
00042 {
00043     SECTION("Without hardware flow control (no errors).")
00044     {
00045         // Mock system calls.
00046         fakeit::Mock<UnixSyscalls> mock_sys;
00047         // Mock 'open'.
00048         fakeit::When(Method(mock_sys, open)).AlwaysReturn(3);
00049         // Mock 'tcgetattr'.
00050         fakeit::When(Method(mock_sys, tcgetattr)).AlwaysDo(
00051             [&](auto fd, auto termios_p)
00052         {
00053             (void)fd;
00054             std::memset(termios_p, '\0', sizeof(struct termios));
00055             return 0;
00056         });
00057         // Mock 'tcsetattr'.
00058         struct termios tty;
00059         fakeit::When(Method(mock_sys, tcsetattr)).AlwaysDo(
00060             [&](auto fd, auto action, auto termios_p)
00061         {
00062             (void)fd;
00063             (void)action;
00064             std::memcpy(&tty, termios_p, sizeof(struct termios));
00065             return 0;
00066         });
00067         // Mock 'close'.
00068         fakeit::When(Method(mock_sys, close)).AlwaysReturn(0);
00069     {
00070         // Construct port.
00071         UnixSerialPort port(
00072             "/dev/ttyUSBO", 9600,
00073             SerialPort::DEFAULT,
00074             mock_unique(mock_sys));
00075         // Verify 'open' system call.
00076         fakeit::Verify(Method(mock_sys, open).Matching(
00077             [](auto path, auto flags)
00078         {
00079             return std::string(path) == "/dev/ttyUSBO" &&
00080                 flags == (O_RDWR | O_NOCTTY | O_SYNC);
00081         }).Once());
00082         // Verify 'tcgetattr'.
00083         fakeit::Verify(Method(mock_sys, tcgetattr).Matching(
00084             [&](auto fd, auto termios_p)
00085         {
00086             (void)termios_p;
00087             return fd == 3;
00088         }).Once());
00089         // Verify 'tcsetattr'.
00090         fakeit::Verify(Method(mock_sys, tcsetattr).Matching(
00091             [](auto fd, auto action, auto termios_p)
00092         {
00093             (void)termios_p;
00094             return fd == 3 && action == TCSANOW;
00095         }).Once());
00096         REQUIRE(cfgetispeed(&tty) == B9600);
00097         REQUIRE(cfgetospeed(&tty) == B9600);
00098         REQUIRE((tty.c_cflag & CLOCAL) != 0);
00099         REQUIRE((tty.c_cflag & CREAD) != 0);
00100         REQUIRE((tty.c_cflag & PARENDB) == 0);
00101         REQUIRE((tty.c_cflag & CSTOPB) == 0);
00102         REQUIRE((tty.c_cflag & CS8) != 0);
```

```

00103     REQUIRE((tty.c_cflag & CRTSCTS) == 0);
00104     REQUIRE((tty.c_lflag & ICANON) == 0);
00105     REQUIRE((tty.c_lflag & ECHO) == 0);
00106     REQUIRE((tty.c_lflag & ECHOE) == 0);
00107     REQUIRE((tty.c_lflag & ISIG) == 0);
00108     REQUIRE((tty.c_iflag & IGNBRK) == 0);
00109     REQUIRE((tty.c_iflag & BRKINT) == 0);
00110     REQUIRE((tty.c_iflag & PARMRK) == 0);
00111     REQUIRE((tty.c_iflag & ISTRIP) == 0);
00112     REQUIRE((tty.c_iflag & INLCR) == 0);
00113     REQUIRE((tty.c_iflag & IGNCR) == 0);
00114     REQUIRE((tty.c_iflag & ICRNL) == 0);
00115     REQUIRE((tty.c_iflag & IXON) == 0);
00116     REQUIRE((tty.c_iflag & IXOFF) == 0);
00117     REQUIRE((tty.c_iflag & IXANY) == 0);
00118     REQUIRE((tty.c_oflag & OPOST) == 0);
00119     REQUIRE(tty.c_cc[VMIN] == 0);
00120     REQUIRE(tty.c_cc[VTIME] == 0);
00121     // Verify 'close'.
00122     fakeit::Verify(Method(mock_sys, close).Using(3)).Exactly(0);
00123 }
00124 // Verify 'close'.
00125 fakeit::Verify(Method(mock_sys, close).Using(3)).Once();
00126 }
00127 SECTION("With hardware flow control (no errors).")
00128 {
00129     // Mock system calls.
00130     fakeit::Mock<UnixSyscalls> mock_sys;
00131     // Mock 'open'.
00132     fakeit::When(Method(mock_sys, open)).Return(3);
00133     // Mock 'tcgetattr'.
00134     fakeit::When(Method(mock_sys, tcgetattr)).Do(
00135         [&](auto fd, auto termios_p)
00136     {
00137         (void)fd;
00138         std::memset(termios_p, '\0', sizeof(struct termios));
00139         return 0;
00140     });
00141     // Mock 'tcsetattr'.
00142     struct termios tty;
00143     fakeit::When(Method(mock_sys, tcsetattr)).Do(
00144         [&](auto fd, auto action, auto termios_p)
00145     {
00146         (void)fd;
00147         (void)action;
00148         std::memcpy(&tty, termios_p, sizeof(struct termios));
00149         return 0;
00150     });
00151     // Mock 'close'.
00152     fakeit::When(Method(mock_sys, close)).Return(0);
00153 {
00154     // Construct port.
00155     UnixSerialPort port(
00156         "/dev/ttyUSB0", 9600,
00157         SerialPort::FLOW_CONTROL,
00158         mock_unique(mock_sys));
00159     // Verify 'open' system call.
00160     fakeit::Verify(Method(mock_sys, open).Matching(
00161         [] (auto path, auto flags)
00162     {
00163         return std::string(path) == "/dev/ttyUSB0" &&
00164             flags == (O_RDWR | O_NOCTTY | O_SYNC);
00165     })).Once();
00166     // Verify 'tcgetattr'.
00167     fakeit::Verify(Method(mock_sys, tcgetattr).Matching(
00168         [&](auto fd, auto termios_p)
00169     {
00170         (void)termios_p;
00171         return fd == 3;
00172     })).Once();
00173     // Verify 'tcsetattr'.
00174     fakeit::Verify(Method(mock_sys, tcsetattr).Matching(
00175         [] (auto fd, auto action, auto termios_p)
00176     {
00177         (void)termios_p;
00178         return fd == 3 && action == TCSANOW;
00179     })).Once();
00180     REQUIRE(cfgetispeed(&tty) == B9600);
00181     REQUIRE(cfgetospeed(&tty) == B9600);
00182     REQUIRE((tty.c_cflag & CLOCAL) != 0);
00183     REQUIRE((tty.c_cflag & CREAD) != 0);

```

```
00184     REQUIRE((tty.c_cflag & PARENB) == 0);
00185     REQUIRE((tty.c_cflag & CSTOPB) == 0);
00186     REQUIRE((tty.c_cflag & CS8) != 0);
00187     REQUIRE((tty.c_cflag & CRTSCTS) != 0);
00188     REQUIRE((tty.c_lflag & ICANON) == 0);
00189     REQUIRE((tty.c_lflag & ECHO) == 0);
00190     REQUIRE((tty.c_lflag & ECHOE) == 0);
00191     REQUIRE((tty.c_lflag & ISIG) == 0);
00192     REQUIRE((tty.c_iflag & IGNBRK) == 0);
00193     REQUIRE((tty.c_iflag & BRKINT) == 0);
00194     REQUIRE((tty.c_iflag & PARMRK) == 0);
00195     REQUIRE((tty.c_iflag & ISTRIP) == 0);
00196     REQUIRE((tty.c_iflag & INLCR) == 0);
00197     REQUIRE((tty.c_iflag & IGNCR) == 0);
00198     REQUIRE((tty.c_iflag & ICRNL) == 0);
00199     REQUIRE((tty.c_iflag & IXON) == 0);
00200     REQUIRE((tty.c_iflag & IXOFF) == 0);
00201     REQUIRE((tty.c_iflag & IXANY) == 0);
00202     REQUIRE((tty.c_oflag & OPOST) == 0);
00203     REQUIRE(tty.c_cc[VMIN] == 0);
00204     REQUIRE(tty.c_cc[VTIME] == 0);
00205     // Verify 'close'.
00206     fakeit::Verify(Method(mock_sys, close).Using(3)).Exactly(0);
00207 }
00208 // Verify 'close'.
00209 fakeit::Verify(Method(mock_sys, close).Using(3)).Once();
00210 }
00211 SECTION("Emmits errors from 'open' system call.")
00212 {
00213     fakeit::Mock<UnixSyscalls> mock_sys;
00214     fakeit::When(Method(mock_sys, open)).AlwaysReturn(-1);
00215     std::array<int, 26> errors{{
00216         EACCES,
00217         EDQUOT,
00218         EEXIST,
00219         EFAULT,
00220         EFBIG,
00221         EINTR,
00222         EINVAL,
00223         EISDIR,
00224         ELOOP,
00225         ENFILE,
00226         ENAMETOOLONG,
00227         ENFILE,
00228         ENODEV,
00229         ENOENT,
00230         ENOMEM,
00231         ENOSPC,
00232         ENOTDIR,
00233         ENXIO,
00234         EOPNOTSUPP,
00235         EVERFLOW,
00236         EPERM,
00237         EROFS,
00238         ETXTBSY,
00239         EWOULDBLOCK,
00240         EBADF,
00241         ENOTDIR
00242     }};
00243     for (auto error : errors)
00244     {
00245         errno = error;
00246         REQUIRE_THROWS_AS(UnixSerialPort(
00247             "/dev/ttyUSBO", 9600, SerialPort::DEFAULT,
00248             mock_unique(mock_sys)), std::system_error);
00249     }
00250 }
00251 fakeit::Verify(Method(mock_sys, close).Using(3)).Exactly(0);
00252 }
00253 SECTION("Emmits errors from 'tcgetattr' system call, and closes the port.")
00254 {
00255     fakeit::Mock<UnixSyscalls> mock_sys;
00256     fakeit::When(Method(mock_sys, open)).AlwaysReturn(3);
00257     fakeit::When(Method(mock_sys, close)).AlwaysReturn(0);
00258     fakeit::When(Method(mock_sys, tcgetattr)).AlwaysReturn(-1);
00259     std::array<int, 2> errors{{
00260         EBADF,
00261         ENOTTY
00262     }};
00263 }
```

```

00265     for (auto error : errors)
00266     {
00267         errno = error;
00268         REQUIRE_THROWS_AS(UnixSerialPort(
00269             "/dev/ttyUSB0", 9600, SerialPort::DEFAULT,
00270             mock_unique(mock_sys)), std::system_error);
00271     }
00272
00273     fakeit::Verify(Method(mock_sys, close).Using(3)).Exactly(2);
00274 }
00275 SECTION("Emmits errors from 'tcsetattr' system call, and closes the port.")
00276 {
00277     fakeit::Mock<UnixSyscalls> mock_sys;
00278     fakeit::When(Method(mock_sys, open)).AlwaysReturn(3);
00279     fakeit::When(Method(mock_sys, close)).AlwaysReturn(0);
00280     fakeit::When(Method(mock_sys, tcgetattr)).AlwaysReturn(0);
00281     fakeit::When(Method(mock_sys, tcsetattr)).AlwaysReturn(-1);
00282     std::array<int, 5> errors {{
00283         EBADF,
00284         EINTR,
00285         EINVAL,
00286         ENOTTY,
00287         EIO
00288     }};
00289
00290     for (auto error : errors)
00291     {
00292         errno = error;
00293         REQUIRE_THROWS_AS(UnixSerialPort(
00294             "/dev/ttyUSB0", 9600, SerialPort::DEFAULT,
00295             mock_unique(mock_sys)), std::system_error);
00296     }
00297
00298     fakeit::Verify(Method(mock_sys, close).Using(3)).Exactly(5);
00299 }
00300 }
00301
00302
00303 TEST_CASE("UnixSerialPort's open method configures the baud rate.",
00304     "[UnixSerialPort]")
00305 {
00306     // Mock system calls.
00307     fakeit::Mock<UnixSyscalls> mock_sys;
00308     // Mock 'open'.
00309     fakeit::When(Method(mock_sys, open)).AlwaysReturn(3);
00310     // Mock 'tcgetattr'.
00311     fakeit::When(Method(mock_sys, tcgetattr)).AlwaysDo(
00312         [&](auto fd, auto termios_p)
00313     {
00314         (void)fd;
00315         std::memset(termios_p, '\0', sizeof(struct termios));
00316         return 0;
00317     });
00318     // Mock 'tcsetattr'.
00319     struct termios tty;
00320     fakeit::When(Method(mock_sys, tcsetattr)).AlwaysDo(
00321         [&](auto fd, auto action, auto termios_p)
00322     {
00323         (void)fd;
00324         (void)action;
00325         std::memcpy(&tty, termios_p, sizeof(struct termios));
00326         return 0;
00327     });
00328     // Mock 'close'.
00329     fakeit::When(Method(mock_sys, close)).AlwaysReturn(0);
00330 SECTION("0 bps")
00331 {
00332     UnixSerialPort port(
00333         "/dev/ttyUSB0", 0,
00334         SerialPort::DEFAULT,
00335         mock_unique(mock_sys));
00336     REQUIRE(cfgetispeed(&tty) == B0);
00337     REQUIRE(cfgetospeed(&tty) == B0);
00338 }
00339 SECTION("50 bps")
00340 {
00341     UnixSerialPort port(
00342         "/dev/ttyUSB0", 50,
00343         SerialPort::DEFAULT,
00344         mock_unique(mock_sys));
00345     REQUIRE(cfgetispeed(&tty) == B50);

```

```
00346     REQUIRE(cfgetospeed(&tty) == B50);
00347 }
00348 SECTION("75 bps")
00349 {
00350     UnixSerialPort port(
00351         "/dev/ttyUSB0", 75,
00352         SerialPort::DEFAULT,
00353         mock_unique(mock_sys));
00354     REQUIRE(cfgetispeed(&tty) == B75);
00355     REQUIRE(cfgetospeed(&tty) == B75);
00356 }
00357 SECTION("110 bps")
00358 {
00359     UnixSerialPort port(
00360         "/dev/ttyUSB0", 110,
00361         SerialPort::DEFAULT,
00362         mock_unique(mock_sys));
00363     REQUIRE(cfgetispeed(&tty) == B110);
00364     REQUIRE(cfgetospeed(&tty) == B110);
00365 }
00366 SECTION("134 bps")
00367 {
00368     UnixSerialPort port(
00369         "/dev/ttyUSB0", 134,
00370         SerialPort::DEFAULT,
00371         mock_unique(mock_sys));
00372     REQUIRE(cfgetispeed(&tty) == B134);
00373     REQUIRE(cfgetospeed(&tty) == B134);
00374 }
00375 SECTION("135 bps")
00376 {
00377     UnixSerialPort port(
00378         "/dev/ttyUSB0", 135,
00379         SerialPort::DEFAULT,
00380         mock_unique(mock_sys));
00381     REQUIRE(cfgetispeed(&tty) == B134);
00382     REQUIRE(cfgetospeed(&tty) == B134);
00383 }
00384 SECTION("150 bps")
00385 {
00386     UnixSerialPort port(
00387         "/dev/ttyUSB0", 150,
00388         SerialPort::DEFAULT,
00389         mock_unique(mock_sys));
00390     REQUIRE(cfgetispeed(&tty) == B150);
00391     REQUIRE(cfgetospeed(&tty) == B150);
00392 }
00393 SECTION("200 bps")
00394 {
00395     UnixSerialPort port(
00396         "/dev/ttyUSB0", 200,
00397         SerialPort::DEFAULT,
00398         mock_unique(mock_sys));
00399     REQUIRE(cfgetispeed(&tty) == B200);
00400     REQUIRE(cfgetospeed(&tty) == B200);
00401 }
00402 SECTION("300 bps")
00403 {
00404     UnixSerialPort port(
00405         "/dev/ttyUSB0", 300,
00406         SerialPort::DEFAULT,
00407         mock_unique(mock_sys));
00408     REQUIRE(cfgetispeed(&tty) == B300);
00409     REQUIRE(cfgetospeed(&tty) == B300);
00410 }
00411 SECTION("600 bps")
00412 {
00413     UnixSerialPort port(
00414         "/dev/ttyUSB0", 600,
00415         SerialPort::DEFAULT,
00416         mock_unique(mock_sys));
00417     REQUIRE(cfgetispeed(&tty) == B600);
00418     REQUIRE(cfgetospeed(&tty) == B600);
00419 }
00420 SECTION("1200 bps")
00421 {
00422     UnixSerialPort port(
00423         "/dev/ttyUSB0", 1200,
00424         SerialPort::DEFAULT,
00425         mock_unique(mock_sys));
00426     REQUIRE(cfgetispeed(&tty) == B1200);
```

```
00427     REQUIRE(cfgetospeed(&tty) == B1200);
00428 }
00429 SECTION("1800 bps")
00430 {
00431     UnixSerialPort port(
00432         "/dev/ttyUSB0", 1800,
00433         SerialPort::DEFAULT,
00434         mock_unique(mock_sys));
00435     REQUIRE(cfgetispeed(&tty) == B1800);
00436     REQUIRE(cfgetospeed(&tty) == B1800);
00437 }
00438 SECTION("2400 bps")
00439 {
00440     UnixSerialPort port(
00441         "/dev/ttyUSB0", 2400,
00442         SerialPort::DEFAULT,
00443         mock_unique(mock_sys));
00444     REQUIRE(cfgetispeed(&tty) == B2400);
00445     REQUIRE(cfgetospeed(&tty) == B2400);
00446 }
00447 SECTION("4800 bps")
00448 {
00449     UnixSerialPort port(
00450         "/dev/ttyUSB0", 4800,
00451         SerialPort::DEFAULT,
00452         mock_unique(mock_sys));
00453     REQUIRE(cfgetispeed(&tty) == B4800);
00454     REQUIRE(cfgetospeed(&tty) == B4800);
00455 }
00456 SECTION("9600 bps")
00457 {
00458     UnixSerialPort port(
00459         "/dev/ttyUSB0", 9600,
00460         SerialPort::DEFAULT,
00461         mock_unique(mock_sys));
00462     REQUIRE(cfgetispeed(&tty) == B9600);
00463     REQUIRE(cfgetospeed(&tty) == B9600);
00464 }
00465 SECTION("19200 bps")
00466 {
00467     UnixSerialPort port(
00468         "/dev/ttyUSB0", 19200,
00469         SerialPort::DEFAULT,
00470         mock_unique(mock_sys));
00471     REQUIRE(cfgetispeed(&tty) == B19200);
00472     REQUIRE(cfgetospeed(&tty) == B19200);
00473 }
00474 SECTION("38400 bps")
00475 {
00476     UnixSerialPort port(
00477         "/dev/ttyUSB0", 38400,
00478         SerialPort::DEFAULT,
00479         mock_unique(mock_sys));
00480     REQUIRE(cfgetispeed(&tty) == B38400);
00481     REQUIRE(cfgetospeed(&tty) == B38400);
00482 }
00483 SECTION("57600 bps")
00484 {
00485     UnixSerialPort port(
00486         "/dev/ttyUSB0", 57600,
00487         SerialPort::DEFAULT,
00488         mock_unique(mock_sys));
00489     REQUIRE(cfgetispeed(&tty) == B57600);
00490     REQUIRE(cfgetospeed(&tty) == B57600);
00491 }
00492 SECTION("115200 bps")
00493 {
00494     UnixSerialPort port(
00495         "/dev/ttyUSB0", 115200,
00496         SerialPort::DEFAULT,
00497         mock_unique(mock_sys));
00498     REQUIRE(cfgetispeed(&tty) == B115200);
00499     REQUIRE(cfgetospeed(&tty) == B115200);
00500 }
00501 SECTION("230400 bps")
00502 {
00503     UnixSerialPort port(
00504         "/dev/ttyUSB0", 230400,
00505         SerialPort::DEFAULT,
00506         mock_unique(mock_sys));
00507     REQUIRE(cfgetispeed(&tty) == B230400);
```

```
00508     REQUIRE(cfgetospeed(&tty) == B230400);
00509 }
00510 SECTION("Throws error when given unsupported baud rate.")
00511 {
00512     REQUIRE_THROWS_AS(
00513         UnixSerialPort(
00514             "/dev/ttyUSB0", 9601, SerialPort::DEFAULT,
00515             mock_unique(mock_sys)), std::invalid_argument);
00516     REQUIRE_THROWS_WITH(
00517         UnixSerialPort(
00518             "/dev/ttyUSB0", 9601, SerialPort::DEFAULT,
00519             mock_unique(mock_sys)), "9601 bps is not a valid baud rate.");
00520 }
00521 }
00522
00523
00524 TEST_CASE("UnixSerialPort's 'read' method receives data on the socket.",
00525     "[UnixSerialPort]")
00526 {
00527     // Mock system calls.
00528     fakeit::Mock<UnixSyscalls> mock_sys;
00529     // Mock 'open'.
00530     fakeit::When(Method(mock_sys, open)).AlwaysReturn(3);
00531     // Mock 'tcgetattr'.
00532     fakeit::When(Method(mock_sys, tcgetattr)).AlwaysReturn(0);
00533     // Mock 'tcsetattr'.
00534     fakeit::When(Method(mock_sys, tcsetattr)).AlwaysReturn(0);
00535     // Mock 'close'.
00536     fakeit::When(Method(mock_sys, close)).AlwaysReturn(0);
00537     // Construct port.
00538     UnixSerialPort port(
00539         "/dev/ttyUSB0", 9600,
00540         SerialPort::DEFAULT,
00541         mock_unique(mock_sys));
00542 SECTION("Timeout, no data (no errors).")
00543 {
00544     // Mock 'poll'.
00545     struct pollfd fds;
00546     fakeit::When(Method(mock_sys, poll)).Do(
00547         [&](auto fds_, auto nfds, auto timeout)
00548     {
00549         (void)timeout;
00550         std::memcpy(&fds, fds_, nfds * sizeof(fds));
00551         return 0;
00552     });
00553     // Test.
00554     REQUIRE(port.read(250ms) == std::vector<uint8_t>());
00555     // Verify 'poll'.
00556     fakeit::Verify(Method(mock_sys, poll).Matching(
00557         [] (auto fds_, auto nfds, auto timeout)
00558     {
00559         (void)fds_;
00560         return nfds == 1 && timeout == 250;
00561     }).Once());
00562     REQUIRE(fds.fd == 3);
00563     REQUIRE(fds.events == POLLIN);
00564     REQUIRE(fds.revents == 0);
00565 }
00566 SECTION("Poll error, close and reopen the serial port (no other errors).")
00567 {
00568     // Mock 'tcgetattr'.
00569     fakeit::When(Method(mock_sys, tcgetattr)).AlwaysDo(
00570         [&] (auto fd, auto termios_p)
00571     {
00572         (void)fd;
00573         std::memset(termios_p, '\0', sizeof(struct termios));
00574         return 0;
00575     });
00576     // Mock 'tcsetattr'.
00577     struct termios tty;
00578     fakeit::When(Method(mock_sys, tcsetattr)).AlwaysDo(
00579         [&] (auto fd, auto action, auto termios_p)
00580     {
00581         (void)fd;
00582         (void)action;
00583         std::memcpy(&tty, termios_p, sizeof(struct termios));
00584         return 0;
00585     });
00586     // Mock 'poll'.
00587     struct pollfd fds;
00588     fakeit::When(Method(mock_sys, poll)).AlwaysDo(
```

```

00589      [&](auto fds_, auto nfds, auto timeout)
00590      {
00591          (void)timeout;
00592          std::memcpy(&fds, fds_, nfds * sizeof(fds));
00593          fds_->revents = POLLERR;
00594          return 1;
00595      });
00596      // Test
00597      REQUIRE(port.read(250ms) == std::vector<uint8_t>());
00598      // Verify 'open' system call.
00599      fakeit::Verify(Method(mock_sys, open).Matching(
00600          [](auto path, auto flags)
00601      {
00602          return std::string(path) == "/dev/ttyUSB0" &&
00603              flags == (O_RDWR | O_NOCTTY | O_SYNC);
00604      }).Exactly(2);
00605      // Verify 'tcgetattr'.
00606      fakeit::Verify(Method(mock_sys, tcgetattr).Matching(
00607          [&](auto fd, auto termios_p)
00608      {
00609          (void)termios_p;
00610          return fd == 3;
00611      }).Exactly(2);
00612      // Verify 'tcsetattr'.
00613      fakeit::Verify(Method(mock_sys, tcsetattr).Matching(
00614          [](auto fd, auto action, auto termios_p)
00615      {
00616          (void)termios_p;
00617          return fd == 3 && action == TCSANOW;
00618      }).Exactly(2);
00619      REQUIRE(cfgetispeed(&tty) == B9600);
00620      REQUIRE(cfgetospeed(&tty) == B9600);
00621      REQUIRE((tty.c_cflag & CLOCAL) != 0);
00622      REQUIRE((tty.c_cflag & CREAD) != 0);
00623      REQUIRE((tty.c_cflag & PARENB) == 0);
00624      REQUIRE((tty.c_cflag & CSTOPB) == 0);
00625      REQUIRE((tty.c_cflag & CS8) != 0);
00626      REQUIRE((tty.c_cflag & CRTSCTS) == 0);
00627      REQUIRE((tty.c_lflag & ICANON) == 0);
00628      REQUIRE((tty.c_lflag & ECHO) == 0);
00629      REQUIRE((tty.c_lflag & ECHOE) == 0);
00630      REQUIRE((tty.c_lflag & ISIG) == 0);
00631      REQUIRE((tty.c_iflag & IGNBRK) == 0);
00632      REQUIRE((tty.c_iflag & BRKINT) == 0);
00633      REQUIRE((tty.c_iflag & PARMRK) == 0);
00634      REQUIRE((tty.c_iflag & ISTRIP) == 0);
00635      REQUIRE((tty.c_iflag & INLCR) == 0);
00636      REQUIRE((tty.c_iflag & IGNCR) == 0);
00637      REQUIRE((tty.c_iflag & ICRNL) == 0);
00638      REQUIRE((tty.c_iflag & IXON) == 0);
00639      REQUIRE((tty.c_iflag & IXOFF) == 0);
00640      REQUIRE((tty.c_iflag & IXANY) == 0);
00641      REQUIRE((tty.c_oflag & OPOST) == 0);
00642      REQUIRE(tty.c_cc[VMIN] == 0);
00643      REQUIRE(tty.c_cc[VTIME] == 0);
00644      // Verify 'close'.
00645      fakeit::Verify(Method(mock_sys, close).Using(3)).Exactly(1);
00646  }
00647 SECTION("Data available (no errors).")
00648  {
00649      // Mock 'poll'.
00650      struct pollfd fds;
00651      fakeit::When(Method(mock_sys, poll)).Do(
00652          [&](auto fds_, auto nfds, auto timeout)
00653      {
00654          (void)timeout;
00655          std::memcpy(&fds, fds_, nfds * sizeof(fds));
00656          fds_->revents = POLLIN;
00657          return 1;
00658      });
00659      // Mock 'read'.
00660      fakeit::When(Method(mock_sys, read)).Do(
00661          [](auto fd, auto buf, auto count)
00662      {
00663          (void)fd;
00664          // Write to buffer.
00665          std::vector<uint8_t> vec = {1, 3, 3, 7};
00666          std::memcpy(buf, vec.data(), std::min(vec.size(), count));
00667          // Return number of received bytes.
00668          return std::min(vec.size(), count);
00669      });

```

```
00670     // Test.
00671     REQUIRE(port.read(250ms) == std::vector<uint8_t>({1, 3, 3, 7}));
00672     // Verify 'poll'.
00673     fakeit::Verify(Method(mock_sys, poll).Matching(
00674         [](auto fds_, auto nfds, auto timeout)
00675     {
00676         (void)fds_;
00677         return nfds == 1 && timeout == 250;
00678     }).Once());
00679     REQUIRE(fds_.fd == 3);
00680     REQUIRE(fds_.events == POLLIN);
00681     REQUIRE(fds_.revents == 0);
00682     // Verify 'read'.
00683     fakeit::Verify(Method(mock_sys, read).Matching(
00684         [](auto fd, auto buf, auto count)
00685     {
00686         (void)buf;
00687         return fd == 3 && count >= 1024;
00688     }));
00689 }
00690 SECTION("Emmits errors from 'read' system call.")
00691 {
00692     // Mock 'poll'.
00693     struct pollfd fds;
00694     fakeit::When(Method(mock_sys, poll)).AlwaysDo(
00695         [&](auto fds_, auto nfds, auto timeout)
00696     {
00697         (void)timeout;
00698         std::memcpy(&fds, fds_, nfds * sizeof(fds));
00699         fds_->revents = POLLIN;
00700         return 1;
00701     });
00702     // Mock 'read'.
00703     fakeit::When(Method(mock_sys, read)).AlwaysReturn(-1);
00704     // Test
00705     std::array<int, 7> errors{{
00706         EAGAIN,
00707         EBADF,
00708         EFAULT,
00709         EINTR,
00710         EINVAL,
00711         EIO,
00712         EISDIR
00713     }};
00714
00715     for (auto error : errors)
00716     {
00717         errno = error;
00718         REQUIRE_THROWS_AS(port.read(250ms), std::system_error);
00719     }
00720 }
00721 SECTION("Emmits errors from 'poll' system call.")
00722 {
00723     // Mock 'poll'.
00724     fakeit::When(Method(mock_sys, poll)).AlwaysReturn(-1);
00725     // Test
00726     std::array<int, 4> errors{{
00727         EFAULT,
00728         EINTR,
00729         EINVAL,
00730         ENOMEM
00731     }};
00732
00733     for (auto error : errors)
00734     {
00735         errno = error;
00736         REQUIRE_THROWS_AS(port.read(250ms), std::system_error);
00737     }
00738 }
00739 }
00740
00741
00742 TEST_CASE("UnixSerialPort's 'write' method sends data over the serial port.",
00743             "[UnixSerialPort]")
00744 {
00745     // Mock system calls.
00746     fakeit::Mock<UnixSyscalls> mock_sys;
00747     // Mock 'open'.
00748     fakeit::When(Method(mock_sys, open)).AlwaysReturn(3);
00749     // Mock 'tcgetattr'.
00750     fakeit::When(Method(mock_sys, tcgetattr)).AlwaysReturn(0);
```

```

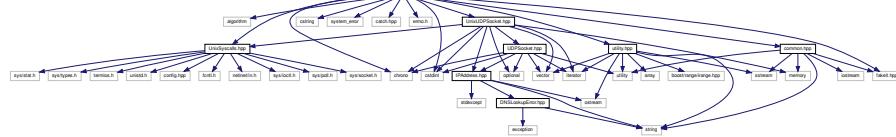
00751 // Mock 'tcsetattr'.
00752 fakeit::When(Method(mock_sys, tcsetattr)).AlwaysReturn(0);
00753 // Mock 'close'.
00754 fakeit::When(Method(mock_sys, close)).Return(0);
00755 // Construct port.
00756 UnixSerialPort port(
00757     "/dev/ttyUSB0", 9600,
00758     SerialPort::DEFAULT,
00759     mock_unique(mock_sys));
00760 SECTION("Without error.")
00761 {
00762     // Mock 'write'.
00763     std::vector<uint8_t> written;
00764     fakeit::When(Method(mock_sys, write)).Do(
00765         [&](auto fd, auto buf, auto count)
00766     {
00767         (void)fd;
00768         written.resize(count);
00769         std::memcpy(written.data(), buf, count);
00770         return count;
00771     });
00772     // Test
00773     std::vector<uint8_t> vec = {1, 3, 3, 7};
00774     port.write(vec);
00775     // Verify 'write'.
00776     fakeit::Verify(Method(mock_sys, write).Matching(
00777         [] (auto fd, auto buf, auto count)
00778     {
00779         (void)buf;
00780         return fd == 3 && count == 4;
00781     })).Once();
00782     REQUIRE(written == vec);
00783 }
00784 SECTION("Could not write all data.")
00785 {
00786     // Mock 'write'.
00787     fakeit::When(Method(mock_sys, write)).AlwaysReturn(3);
00788     // Test
00789     std::vector<uint8_t> vec = {1, 3, 3, 7};
00790     REQUIRE_THROWS_AS(port.write(vec), PartialSendError);
00791     REQUIRE_THROWS_WITH(port.write(vec), "Could only write 3 of 4 bytes.");
00792 }
00793 SECTION("Emits errors from 'write' system call.")
00794 {
00795     fakeit::When(Method(mock_sys, write)).AlwaysReturn(-1);
00796     std::array<int, 11> errors{{
00797         EAGAIN,
00798         EBADF,
00799         EDESTADDRREQ,
00800         EDQUOT,
00801        EFAULT,
00802         EFBIG,
00803         EINTR,
00804         EINVAL,
00805         EIO,
00806         ENOSPC,
00807         EPIPE
00808     }};
00809
00810     for (auto error : errors)
00811     {
00812         errno = error;
00813         REQUIRE_THROWS_AS(port.write({1, 3, 3, 7}), std::system_error);
00814     }
00815 }
00816 }
00817
00818
00819 TEST_CASE("UnixSerialPort's are printable.", "[UnixSerialPort]")
00820 {
00821     // Mock system calls.
00822     fakeit::Mock<UnixSyscalls> mock_sys;
00823     // Mock 'open'.
00824     fakeit::When(Method(mock_sys, open)).AlwaysReturn(3);
00825     // Mock 'tcgetattr'.
00826     fakeit::When(Method(mock_sys, tcgetattr)).AlwaysDo(
00827         [&](auto fd, auto termios_p)
00828     {
00829         (void)fd;
00830         std::memset(termios_p, '\0', sizeof(struct termios));
00831         return 0;
00832     });
00833 }
```

```
00832 });
00833 // Mock 'tcsetattr'.
00834 struct termios tty;
00835 fakeit::When(Method(mock_sys, tcsetattr)).AlwaysDo(
00836 [&](auto fd, auto action, auto termios_p)
00837 {
00838     (void)fd;
00839     (void)action;
00840     std::memcpy(&tty, termios_p, sizeof(struct termios));
00841     return 0;
00842 });
00843 // Mock 'close'.
00844 fakeit::When(Method(mock_sys, close)).AlwaysReturn(0);
00845 SECTION("Without flow control.")
00846 {
00847     UnixSerialPort port(
00848         "/dev/ttyUSB0", 9600,
00849         SerialPort::DEFAULT,
00850         mock_unique(mock_sys));
00851 REQUIRE(
00852     str(port) ==
00853     "serial {\n"
00854     "    device /dev/ttyUSB0;\n"
00855     "    baudrate 9600;\n"
00856     "    flow_control no;\n"
00857     "}");
00858 }
00859 SECTION("With flow control.")
00860 {
00861     UnixSerialPort port(
00862         "/dev/ttyUSB0", 9600,
00863         SerialPort::FLOW_CONTROL,
00864         mock_unique(mock_sys));
00865 REQUIRE(
00866     str(port) ==
00867     "serial {\n"
00868     "    device /dev/ttyUSB0;\n"
00869     "    baudrate 9600;\n"
00870     "    flow_control yes;\n"
00871     "}");
00872 }
00873 }
```

16.271 test_UdpSocket.cpp File Reference

```
#include <algorithm>
#include <chrono>
#include <cstdint>
#include <cstring>
#include <system_error>
#include <catch.hpp>
#include <errno.h>
#include <fakeit.hpp>
#include "UnixSyscalls.hpp"
#include "UnixUDPSocket.hpp"
#include "utility.hpp"
#include "common.hpp"
```

include dependency graph for test_GoogleTest_GoogleMock.



Functions

- [TEST_CASE \("UnixUDPSocket's create and bind a UDP socket on construction and " "closes the socket on destruction.", "\[UnixUDPSocket\]"\)](#)
- [TEST_CASE \("UnixUDPSocket's 'send' method sends data on the socket.", "\[UnixUDPSocket\]"\)](#)
- [TEST_CASE \("UnixUDPSocket's 'receive' method receives data on the socket.", "\[UnixUDPSocket\]"\)](#)
- [TEST_CASE \("UnixUDPSocket's are printable.", "\[UnixUDPSocket\]"\)](#)

16.271.1 Function Documentation

16.271.1.1 TEST_CASE() [1/4]

```
TEST_CASE (
    "UnixUDPSocket's create and bind a UDP socket on construction and " "closes the socket
on destruction." ,
    "" [UnixUDPSocket] )
```

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

16.271.1.2 TEST_CASE() [2/4]

```
TEST_CASE (
    "UnixUDPSocket's 'send' method sends data on the socket." ,
    "" [UnixUDPSocket] )
```

Definition at line 179 of file [test_UnderSocket.cpp](#).

16.271.1.3 TEST_CASE() [3/4]

```
TEST_CASE (
    "UnixUDPSocket's 'receive' method receives data on the socket." ,
    "" [UnixUDPSocket] )
```

Definition at line 284 of file [test_UnderSocket.cpp](#).

References [mock_unique\(\)](#).

Here is the call graph for this function:



16.271.1.4 TEST_CASE() [4/4]

```
TEST_CASE (
    "UnixUDPSocket's are printable." ,
    "" [UnixUDPSocket] )
```

Definition at line 721 of file [test_UinxUDPSocket.cpp](#).

16.272 test_UinxUDPSocket.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <algorithm>
00019 #include <chrono>
00020 #include <cstdint>
00021 #include <cstring>
00022 #include <system_error>
00023
00024 #include <catch.hpp>
00025 #include <errno.h>
00026 #include <fakeit.hpp>
00027
00028 #include "UnixSyscalls.hpp"
00029 #include "UnixUDPSocket.hpp"
00030 #include "utility.hpp"
00031
00032 #include "common.hpp"
00033
00034
00035 using namespace std::chrono_literals;
00036
00037
00038 TEST_CASE("UnixUDPSocket's create and bind a UDP socket on construction and "
00039             "closes the socket on destruction.", "[UnixUDPSocket]")
00040 {
00041     SECTION("Without a specific IP address (no errors).")
00042     {
00043         // Mock system calls.
00044         fakeit::Mock<UnixSyscalls> mock_sys;
00045         // Mock 'socket'.
00046         fakeit::When(Method(mock_sys, socket)).Return(3);
00047         // Mock 'bind'.
00048         struct sockaddr_in address;
00049         fakeit::When(Method(mock_sys, bind)).Do(
00050             [&](auto fd, auto addr, auto addrlen)
00051         {
00052             (void)fd;
00053             std::memcpy(&address, addr, addrlen);
00054             return 0;
00055         });
00056         // Mock 'close'.
00057         fakeit::When(Method(mock_sys, close)).Return(0);
00058     {
00059         // Construct socket.
00060         UnixUDPSocket socket(14050, {}, 0, mock_unique(mock_sys));
00061         fakeit::Verify(Method(mock_sys, socket).Matching(
00062                         [&](auto family, auto type, auto protocol)
```

```

00063     {
00064         return family == AF_INET && type == SOCK_DGRAM && protocol == 0;
00065     }).Once();
00066     fakeit::Verify(Method(mock_sys, bind).Matching(
00067         [&](auto fd, auto addr, auto addrlen)
00068     {
00069         (void)addr;
00070         return fd == 3 && addrlen == sizeof(address);
00071     }).Once();
00072     REQUIRE(address.sin_family == AF_INET);
00073     REQUIRE(ntohs(address.sin_port) == 14050);
00074     REQUIRE(ntohl(address.sin_addr.s_addr) == INADDR_ANY);
00075
00076     for (size_t i = 0; i < sizeof(address.sin_zero); ++i)
00077     {
00078         REQUIRE(address.sin_zero[i] == '\0');
00079     }
00080
00081     fakeit::Verify(Method(mock_sys, close).Using(3)).Exactly(0);
00082 }
00083     fakeit::Verify(Method(mock_sys, close).Using(3)).Once();
00084 }
00085 SECTION("With a specific IP address (no errors).")
00086 {
00087     fakeit::Mock<UnixSyscalls> mock_sys;
00088     fakeit::When(Method(mock_sys, socket)).Return(3);
00089     struct sockaddr_in address;
00090     fakeit::When(Method(mock_sys, bind)).Do(
00091         [&](auto fd, auto addr, auto addrlen)
00092     {
00093         (void)fd;
00094         std::memcpy(&address, addr, addrlen);
00095         return 0;
00096     });
00097     fakeit::When(Method(mock_sys, close)).Return(0);
00098 {
00099     UnixUDPSocket socket(
00100         14050, IPAddress(1234567890), 0, mock_unique(mock_sys));
00101     fakeit::Verify(Method(mock_sys, socket).Matching(
00102         [&](auto family, auto type, auto protocol)
00103     {
00104         return family == AF_INET && type == SOCK_DGRAM && protocol == 0;
00105     }).Once());
00106     fakeit::Verify(Method(mock_sys, bind).Matching(
00107         [] (auto fd, auto addr, auto addrlen)
00108     {
00109         (void)addr;
00110         return fd == 3 && addrlen == sizeof(address);
00111     }).Once());
00112     REQUIRE(address.sin_family == AF_INET);
00113     REQUIRE(ntohs(address.sin_port) == 14050);
00114     REQUIRE(ntohl(address.sin_addr.s_addr) == 1234567890);
00115
00116     for (size_t i = 0; i < sizeof(address.sin_zero); ++i)
00117     {
00118         REQUIRE(address.sin_zero[i] == '\0');
00119     }
00120
00121     fakeit::Verify(Method(mock_sys, close).Using(3)).Exactly(0);
00122 }
00123     fakeit::Verify(Method(mock_sys, close).Using(3)).Once();
00124 }
00125 SECTION("Emmits errors from 'socket' system call.")
00126 {
00127     fakeit::Mock<UnixSyscalls> mock_sys;
00128     fakeit::When(Method(mock_sys, socket)).AlwaysReturn(-1);
00129     std::array<int, 7> errors{{
00130         EACCES,
00131         EAFNOSUPPORT,
00132         EINVAL,
00133         EMFILE,
00134         ENOBUFS,
00135         ENOMEM,
00136         EPROTONOSUPPORT
00137     }};
00138
00139     for (auto error : errors)
00140     {
00141         errno = error;
00142         REQUIRE_THROWS_AS(
00143             UnixUDPSocket(14050, {}, 0, mock_unique(mock_sys)),

```

```

00144             std::system_error);
00145     }
00146 }
00147 SECTION("Emmits errors from 'bind' system call.")
00148 {
00149     fakeit::Mock<UnixSyscalls> mock_sys;
00150     fakeit::When(Method(mock_sys, socket)).AlwaysReturn(4);
00151     fakeit::When(Method(mock_sys, bind)).AlwaysReturn(-1);
00152     std::array<int, 13> errors{{
00153         EACCES,
00154         EADDRINUSE,
00155         EBADF,
00156         EINVAL,
00157         ENOTSOCK,
00158         EADDRNOTAVAIL,
00159        EFAULT,
00160         ELOOP,
00161         ENAMETOOLONG,
00162         ENOENT,
00163         ENOMEM,
00164         ENOTDIR,
00165         EROFS
00166     }};
00167
00168     for (auto error : errors)
00169     {
00170         errno = error;
00171         REQUIRE_THROWS_AS(
00172             UnixUDPSocket(14050, {}, 0, mock_unique(mock_sys)),
00173             std::system_error);
00174     }
00175 }
00176 }
00177
00178
00179 TEST_CASE("UnixUDPSocket's 'send' method sends data on the socket.",
00180             "[UnixUDPSocket]")
00181 {
00182     // Mock system calls.
00183     fakeit::Mock<UnixSyscalls> mock_sys;
00184     // Mock 'socket'.
00185     fakeit::When(Method(mock_sys, socket)).Return(3);
00186     // Mock 'bind'.
00187     fakeit::When(Method(mock_sys, bind)).Return(0);
00188     // Mock 'close'.
00189     fakeit::When(Method(mock_sys, close)).Return(0);
00190     SECTION("Without error.")
00191 {
00192     UnixUDPSocket socket(14050, {}, 0, mock_unique(mock_sys));
00193     // Mock 'sendto'
00194     std::vector<uint8_t> sent;
00195     struct sockaddr_in address;
00196     fakeit::When(Method(mock_sys, sendto)).Do(
00197         [&](auto fd, auto buf, auto len, auto flags,
00198             auto addr, auto addrlen)
00199     {
00200         (void)fd;
00201         (void)flags;
00202         sent.resize(len);
00203         std::memcpy(sent.data(), buf, len);
00204         std::memcpy(&address, addr, addrlen);
00205         return len;
00206     });
00207     // Test
00208     std::vector<uint8_t> vec = {1, 3, 3, 7};
00209     socket.send(vec, IPAddress(1234567890, 14050));
00210     // Verify 'sendto'.
00211     fakeit::Verify(Method(mock_sys, sendto).Matching(
00212         [] (auto fd, auto buf, auto len, auto flags,
00213             auto addr, auto addrlen)
00214     {
00215         (void)buf;
00216         (void)addr;
00217         return fd == 3 && len == 4 && flags == 0 &&
00218             addrlen == sizeof(address);
00219     }).Once());
00220     REQUIRE(sent == vec);
00221     REQUIRE(address.sin_family == AF_INET);
00222     REQUIRE(ntohs(address.sin_port) == 14050);
00223     REQUIRE(ntohl(address.sin_addr.s_addr) == 1234567890);
00224 }
```

```

00225     for (size_t i = 0; i < sizeof(address.sin_zero); ++i)
00226     {
00227         REQUIRE(address.sin_zero[i] == '\0');
00228     }
00229 }
00230 SECTION("Bitrate limit prevents writing packets too fast.")
00231 {
00232     UnixUDPSocket socket(14050, {}, 128, mock_unique(mock_sys));
00233     fakeit::Fake(Method(mock_sys, sendto));
00234     std::vector<uint8_t> vec = {1, 3, 3, 7}; // 4*8/128 = 0.25 seconds
00235     socket.send(vec, IPAddress(1234567890, 14050));
00236     fakeit::Verify(Method(mock_sys, sendto)).Once();
00237     auto tic = std::chrono::steady_clock::now();
00238     socket.send(vec, IPAddress(1234567890, 14050));
00239     auto toc = std::chrono::steady_clock::now();
00240     REQUIRE(
00241         std::chrono::duration_cast<std::chrono::milliseconds>(
00242             toc - tic).count() >= 150);
00243     REQUIRE(
00244         std::chrono::duration_cast<std::chrono::milliseconds>(
00245             toc - tic).count() <= 350);
00246     fakeit::Verify(Method(mock_sys, sendto)).Exactly(2);
00247 }
00248 SECTION("Emmits errors from 'sendto' system call.")
00249 {
00250     UnixUDPSocket socket(14050, {}, 0, mock_unique(mock_sys));
00251     fakeit::When(Method(mock_sys, sendto)).AlwaysReturn(-1);
00252     std::array<int, 18> errors{{
00253         EACCES,
00254         EAGAIN,
00255         EWOULDBLOCK,
00256         EALREADY,
00257         EBADF,
00258         ECONNRESET,
00259         EDESTADDRREQ,
00260        EFAULT,
00261         EINTR,
00262         EINVAL,
00263         EISCONN,
00264         EMSGSIZE,
00265         ENOBUFS,
00266         ENOMEM,
00267         ENOTCONN,
00268         ENOTSOCK,
00269         EOPNOTSUPP,
00270         EPIPE
00271     }};
00272
00273     for (auto error : errors)
00274     {
00275         errno = error;
00276         REQUIRE_THROWS_AS(
00277             socket.send({1, 3, 3, 7}, IPAddress(1234567890, 14050)),
00278             std::system_error);
00279     }
00280 }
00281 }
00282
00283
00284 TEST_CASE("UnixUDPSocket's 'receive' method receives data on the socket.",
00285             "[UnixUDPSocket]")
00286 {
00287     // Mock system calls.
00288     fakeit::Mock<UnixSyscalls> mock_sys;
00289     // Mock 'socket'.
00290     fakeit::When(Method(mock_sys, socket)).AlwaysReturn(3);
00291     // Mock 'bind'.
00292     fakeit::When(Method(mock_sys, bind)).AlwaysReturn(0);
00293     // Mock 'close'.
00294     fakeit::When(Method(mock_sys, close)).AlwaysReturn(0);
00295     // Construct socket.
00296     UnixUDPSocket socket(14050, {}, 0, mock_unique(mock_sys));
00297     SECTION("Timeout, no packet (no errors).")
00298 {
00299     // Mock 'poll'.
00300     struct pollfd fds;
00301     fakeit::When(Method(mock_sys, poll)).Do(
00302         [&](auto fds_, auto nfds, auto timeout)
00303     {
00304         (void)timeout;
00305         std::memcpy(&fds, fds_, nfds * sizeof(fds));
00306     });
00307 }
```

```

00306         return 0;
00307     });
00308     // Test.
00309     REQUIRE(
00310         socket.receive(250ms) ==
00311         std::pair<std::vector<uint8_t>, IPAddress>({}, IPAddress(0)));
00312     // Verify 'poll'.
00313     fakeit::Verify(Method(mock_sys, poll).Matching(
00314         [](auto fds_, auto nfds, auto timeout)
00315     {
00316         (void)fds_;
00317         return nfds == 1 && timeout == 250;
00318     }).Once());
00319     REQUIRE(fds.fd == 3);
00320     REQUIRE(fds.events == POLLIN);
00321     REQUIRE(fds.revents == 0);
00322 }
00323 SECTION("Poll error, close and restart the socket (no other errors).")
00324 {
00325     // Mock 'bind'.
00326     struct sockaddr_in address;
00327     fakeit::When(Method(mock_sys, bind)).Do(
00328         [&](auto fd, auto addr, auto addrlen)
00329     {
00330         (void)fd;
00331         std::memcpy(&address, addr, addrlen);
00332         return 0;
00333     });
00334     // Mock 'poll'.
00335     struct pollfd fds;
00336     fakeit::When(Method(mock_sys, poll)).Do(
00337         [&](auto fds_, auto nfds, auto timeout)
00338     {
00339         (void)timeout;
00340         std::memcpy(&fds, fds_, nfds * sizeof(fds));
00341         fds->revents = POLLERR;
00342         return 1;
00343     });
00344     // Test
00345     REQUIRE(
00346         socket.receive(250ms) ==
00347         std::pair<std::vector<uint8_t>, IPAddress>({}, IPAddress(0)));
00348     // Verify 'poll'.
00349     fakeit::Verify(Method(mock_sys, poll).Matching(
00350         [](auto fds_, auto nfds, auto timeout)
00351     {
00352         (void)fds_;
00353         return nfds == 1 && timeout == 250;
00354     }).Once());
00355     REQUIRE(fds.fd == 3);
00356     REQUIRE(fds.events == POLLIN);
00357     REQUIRE(fds.revents == 0);
00358     // Verify 'socket'.
00359     fakeit::Verify(Method(mock_sys, socket).Matching(
00360         [&](auto family, auto type, auto protocol)
00361     {
00362         return family == AF_INET && type == SOCK_DGRAM && protocol == 0;
00363     }).Exactly(2));
00364     // Verify 'bind'.
00365     fakeit::Verify(Method(mock_sys, bind).Matching(
00366         [&](auto fd, auto addr, auto addrlen)
00367     {
00368         (void)addr;
00369         return fd == 3 && addrlen == sizeof(address);
00370     }).Exactly(2));
00371     REQUIRE(address.sin_family == AF_INET);
00372     REQUIRE(ntohs(address.sin_port) == 14050);
00373     REQUIRE(ntohl(address.sin_addr.s_addr) == INADDR_ANY);
00374
00375     for (size_t i = 0; i < sizeof(address.sin_zero); ++i)
00376     {
00377         REQUIRE(address.sin_zero[i] == '\0');
00378     }
00379
00380     // Verify 'close'.
00381     fakeit::Verify(Method(mock_sys, close).Using(3)).Once();
00382 }
00383 SECTION("Packet available (no errors).")
00384 {
00385     // Mock 'poll'.
00386     struct pollfd fds;

```

```

00387     fakeit::When(Method(mock_sys, poll)).Do(
00388         [&](auto fds_, auto nfds, auto timeout)
00389     {
00390         (void)timeout;
00391         std::memcpy(&fds, fds_, nfds * sizeof(fds));
00392         fds->revents = POLLIN;
00393         return 1;
00394     });
00395     // Mock 'ioctl'.
00396     fakeit::When(Method(mock_sys, ioctl)).Do(
00397         [](auto fd, auto request, auto size)
00398     {
00399         (void)fd;
00400         (void)request;
00401         *reinterpret_cast<int *>(size) = 4;
00402         return 0;
00403     });
00404     // Mock 'recvfrom'.
00405     socklen_t address_length = 0;
00406     fakeit::When(Method(mock_sys, recvfrom)).Do(
00407         [&](auto fd, auto buf, auto len, auto flags,
00408             auto addr, auto addrlen)
00409     {
00410         (void)fd;
00411         (void)flags;
00412         // Write to buffer.
00413         std::vector<uint8_t> vec = {1, 3, 3, 7};
00414         std::memcpy(buf, vec.data(), std::min(vec.size(), len));
00415         // Set address
00416         struct sockaddr_in address;
00417         address.sin_family = AF_INET;
00418         address.sin_port = htons(static_cast<uint16_t>(5000));
00419         address.sin_addr.s_addr = htonl(static_cast<uint32_t>(1234567890));
00420         memset(address.sin_zero, '\0', sizeof(address.sin_zero));
00421         std::memcpy(
00422             &addr, &address,
00423             std::min(static_cast<size_t>(*addrlen), sizeof(address)));
00424         address_length = *addrlen;
00425         *addrlen = static_cast<socklen_t>(
00426             std::min(static_cast<size_t>(*addrlen),
00427                     sizeof(address)));
00428         // Return number of received bytes.
00429         return std::min(vec.size(), len);
00430     });
00431     // Test
00432     auto [data, ip] = socket.receive(250ms);
00433     REQUIRE(data == std::vector<uint8_t>({1, 3, 3, 7}));
00434     REQUIRE(ip == IPAddress(1234567890, 5000));
00435     // Verify 'poll'.
00436     fakeit::Verify(Method(mock_sys, poll).Matching(
00437         [](auto fds_, auto nfds, auto timeout)
00438     {
00439         (void)fds_;
00440         return nfds == 1 && timeout == 250;
00441     }).Once());
00442     REQUIRE(fds.fd == 3);
00443     REQUIRE(fds.events == POLLIN);
00444     REQUIRE(fds.revents == 0);
00445     // Verify 'ioctl'.
00446     fakeit::Verify(Method(mock_sys, ioctl).Matching(
00447         [](auto fd, auto request, auto size)
00448     {
00449         (void)size;
00450         return fd == 3 && request == FIONREAD;
00451     }).Once());
00452     // Verify 'recvfrom'.
00453     fakeit::Verify(Method(mock_sys, recvfrom).Matching(
00454         [](auto fd, auto buf, auto len, auto flags,
00455             auto addr, auto addrlen)
00456     {
00457         (void)buf;
00458         (void)addr;
00459         (void)addrlen;
00460         return fd == 3 && len >= 4 && flags == 0;
00461     }).Once());
00462     REQUIRE(address_length >= sizeof(sockaddr_in));
00463 }
00464 SECTION("Packet available (not IPv4).")
00465 {
00466     // Mock poll system call.
00467     struct pollfd fds;

```

```

00468     fakeit::When(Method(mock_sys, poll)).Do(
00469         [&](auto fds_, auto nfds, auto timeout)
00470     {
00471         (void)timeout;
00472         std::memcpy(&fds, fds_, nfds * sizeof(fds));
00473         fds->revents = POLLIN;
00474         return 1;
00475     });
00476     // Mock ioctl system call.
00477     fakeit::When(Method(mock_sys, ioctl)).Do(
00478         [](auto fd, auto request, auto size)
00479     {
00480         (void)fd;
00481         (void)request;
00482         *reinterpret_cast<int *>(size) = 4;
00483         return 0;
00484     });
00485     // Mock recvfrom.
00486     socklen_t address_length = 0;
00487     fakeit::When(Method(mock_sys, recvfrom)).Do(
00488         [&](auto fd, auto buf, auto len, auto flags,
00489             auto addr, auto addrlen)
00490     {
00491         (void)fd;
00492         (void)flags;
00493         // Write to buffer.
00494         std::vector<uint8_t> vec = {1, 3, 3, 7};
00495         std::memcpy(buf, vec.data(), std::min(vec.size(), len));
00496         // Set address
00497         struct sockaddr_in address;
00498         address.sin_family = AF_INET6;
00499         address.sin_port = htons(static_cast<uint16_t>(5000));
00500         address.sin_addr.s_addr = htonl(static_cast<uint32_t>(1234567890));
00501         memset(address.sin_zero, '\0', sizeof(address.sin_zero));
00502         std::memcpy(
00503             &addr, &address,
00504             std::min(static_cast<size_t>(*addrlen), sizeof(address)));
00505         address_length = *addrlen;
00506         *addrlen = static_cast<socklen_t>(
00507             std::min(static_cast<size_t>(*addrlen),
00508                     sizeof(address)));
00509         // Return number of received bytes.
00510         return std::min(vec.size(), len);
00511     });
00512     // Test
00513     auto [data, ip] = socket.receive(250ms);
00514     REQUIRE(data == std::vector<uint8_t>());
00515     REQUIRE(ip == IPAddress(0));
00516     // Verify poll system call.
00517     fakeit::Verify(Method(mock_sys, poll).Matching(
00518         [](auto fds_, auto nfds, auto timeout)
00519     {
00520         (void)fds_;
00521         return nfds == 1 && timeout == 250;
00522     }).Once());
00523     REQUIRE(fds.fd == 3);
00524     REQUIRE(fds.events == POLLIN);
00525     REQUIRE(fds.revents == 0);
00526     // Verify ioctl system call.
00527     fakeit::Verify(Method(mock_sys, ioctl).Matching(
00528         [](auto fd, auto request, auto size)
00529     {
00530         (void)size;
00531         return fd == 3 && request == FIONREAD;
00532     }).Once());
00533     // Verify recvfrom.
00534     fakeit::Verify(Method(mock_sys, recvfrom).Matching(
00535         [](auto fd, auto buf, auto len, auto flags,
00536             auto addr, auto addrlen)
00537     {
00538         (void)buf;
00539         (void)addr;
00540         (void)addrlen;
00541         return fd == 3 && len >= 4 && flags == 0;
00542     }).Once());
00543     REQUIRE(address_length >= sizeof(sockaddr_in));
00544 }
00545 SECTION("Packet available (IP address truncated).")
00546 {
00547     // Mock poll system call.
00548     struct pollfd fds;

```

```

00549     fakeit::When(Method(mock_sys, poll)).Do(
00550         [&](auto fds_, auto nfds, auto timeout)
00551     {
00552         (void)timeout;
00553         std::memcpy(&fds, fds_, nfds * sizeof(fds));
00554         fds->revents = POLLIN;
00555         return 1;
00556     });
00557     // Mock ioctl system call.
00558     fakeit::When(Method(mock_sys, ioctl)).Do(
00559         [](auto fd, auto request, auto size)
00560     {
00561         (void)fd;
00562         (void)request;
00563         *reinterpret_cast<int *>(size) = 4;
00564         return 0;
00565     });
00566     // Mock recvfrom.
00567     socklen_t address_length = 0;
00568     fakeit::When(Method(mock_sys, recvfrom)).Do(
00569         [&](auto fd, auto buf, auto len, auto flags,
00570             auto addr, auto addrlen)
00571     {
00572         (void)fd;
00573         (void)flags;
00574         // Write to buffer.
00575         std::vector<uint8_t> vec = {1, 3, 3, 7};
00576         std::memcpy(buf, vec.data(), std::min(vec.size(), len));
00577         // Set address
00578         struct sockaddr_in address;
00579         address.sin_family = AF_INET;
00580         address.sin_port = htons(static_cast<uint16_t>(5000));
00581         address.sin_addr.s_addr = htonl(static_cast<uint32_t>(1234567890));
00582         memset(address.sin_zero, '\0', sizeof(address.sin_zero));
00583         std::memcpy(
00584             &address,
00585             std::min(static_cast<size_t>(*addrlen), sizeof(address)));
00586         address_length = *addrlen;
00587         *addrlen = static_cast<socklen_t>(
00588             std::min(static_cast<size_t>(*addrlen),
00589                     sizeof(address)) + 1);
00590         // Return number of received bytes.
00591         return std::min(vec.size(), len);
00592     });
00593     // Test
00594     auto [data, ip] = socket.receive(250ms);
00595     REQUIRE(data == std::vector<uint8_t>());
00596     REQUIRE(ip == IPAddress(0));
00597     // Verify poll system call.
00598     fakeit::Verify(Method(mock_sys, poll).Matching(
00599         [](auto fds_, auto nfds, auto timeout)
00600     {
00601         (void)fds_;
00602         return nfds == 1 && timeout == 250;
00603     }).Once());
00604     REQUIRE(fds_.fd == 3);
00605     REQUIRE(fds_.events == POLLIN);
00606     REQUIRE(fds_.revents == 0);
00607     // Verify ioctl system call.
00608     fakeit::Verify(Method(mock_sys, ioctl).Matching(
00609         [](auto fd, auto request, auto size)
00610     {
00611         (void)size;
00612         return fd == 3 && request == FIONREAD;
00613     }).Once());
00614     // Verify recvfrom.
00615     fakeit::Verify(Method(mock_sys, recvfrom).Matching(
00616         [](auto fd, auto buf, auto len, auto flags,
00617             auto addr, auto addrlen)
00618     {
00619         (void)buf;
00620         (void)addr;
00621         (void)addrlen;
00622         return fd == 3 && len >= 4 && flags == 0;
00623     }).Once());
00624     REQUIRE(address_length >= sizeof(sockaddr_in));
00625 }
00626 SECTION("Emmits errors from 'recvfrom' system call.")
00627 {
00628     // Mock poll system call.
00629     struct pollfd fds;

```

```
00630     fakeit::When(Method(mock_sys, poll)).AlwaysDo(
00631         [&](auto fds_, auto nfds, auto timeout)
00632     {
00633         (void)timeout;
00634         std::memcpy(&fds, fds_, nfds * sizeof(fds));
00635         fds->revents = POLLIN;
00636         return 1;
00637     });
00638     // Mock ioctl system call.
00639     fakeit::When(Method(mock_sys, ioctl)).AlwaysDo(
00640         [](auto fd, auto request, auto size)
00641     {
00642         (void)fd;
00643         (void)request;
00644         *reinterpret_cast<int *>(size) = 4;
00645         return 0;
00646     });
00647     // Mock recvfrom.
00648     fakeit::When(Method(mock_sys, recvfrom)).AlwaysReturn(-1);
00649     // Test
00650     std::array<int, 13> errors{
00651         EACCES,
00652         EWOULD_BLOCK,
00653         EBADF,
00654         ECONNRESET,
00655         EINTR,
00656         EINVAL,
00657         ENOTCONN,
00658         ENOTSOCK,
00659         EOPNOTSUPP,
00660         ETIMEDOUT,
00661         EIO,
00662         ENOBUFS,
00663         ENOMEM
00664     };
00665
00666     for (auto error : errors)
00667     {
00668         errno = error;
00669         REQUIRE_THROWS_AS(socket.receive(250ms), std::system_error);
00670     }
00671 }
00672 SECTION("Emmits errors from 'ioctl' system call.")
00673 {
00674     // Mock poll system call.
00675     struct pollfd fds;
00676     fakeit::When(Method(mock_sys, poll)).AlwaysDo(
00677         [&](auto fds_, auto nfds, auto timeout)
00678     {
00679         (void)timeout;
00680         std::memcpy(&fds, fds_, nfds * sizeof(fds));
00681         fds->revents = POLLIN;
00682         return 1;
00683     });
00684     // Mock ioctl system call.
00685     fakeit::When(Method(mock_sys, ioctl)).AlwaysReturn(-1);
00686     // Test
00687     std::array<int, 4> errors{
00688         EBADF,
00689        EFAULT,
00690         EINVAL,
00691         ENOTTY
00692     };
00693
00694     for (auto error : errors)
00695     {
00696         errno = error;
00697         REQUIRE_THROWS_AS(socket.receive(250ms), std::system_error);
00698     }
00699 }
00700 SECTION("Emmits errors from 'poll' system call.")
00701 {
00702     // Mock poll system call.
00703     fakeit::When(Method(mock_sys, poll)).AlwaysReturn(-1);
00704     // Test
00705     std::array<int, 4> errors{
00706        EFAULT,
00707         EINTR,
00708         EINVAL,
00709         ENOMEM
00710     };

```

```

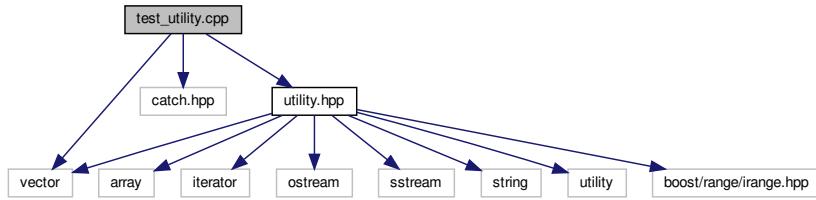
00711     for (auto error : errors)
00712     {
00713         errno = error;
00714         REQUIRE_THROWS_AS(socket.receive(250ms), std::system_error);
00715     }
00716 }
00717 }
00718 }
00719
00720
00721 TEST_CASE("UnixUDPSocket's are printable.", "[UnixUDPSocket]")
00722 {
00723     // Mock system calls.
00724     fakeit::Mock<UnixSyscalls> mock_sys;
00725     // Mock 'socket'.
00726     fakeit::When(Method(mock_sys, socket)).Return(3);
00727     // Mock 'bind'.
00728     struct sockaddr_in address;
00729     fakeit::When(Method(mock_sys, bind)).Do(
00730         [&](auto fd, auto addr, auto addrlen)
00731     {
00732         (void)fd;
00733         std::memcpy(&address, addr, addrlen);
00734         return 0;
00735     });
00736     // Mock 'close'.
00737     fakeit::When(Method(mock_sys, close)).Return(0);
00738 SECTION("Without explicit IP address.")
00739 {
00740     UnixUDPSocket socket(14050, {}, 0, mock_unique(mock_sys));
00741     REQUIRE(
00742         str(socket) ==
00743         "udp (\n"
00744         "    port 14050;\n"
00745         "}");
00746 }
00747 SECTION("With explicit IP address.")
00748 {
00749     UnixUDPSocket socket(
00750         14050, IPAddress("127.0.0.1"), 0, mock_unique(mock_sys));
00751     REQUIRE(
00752         str(socket) ==
00753         "udp (\n"
00754         "    port 14050;\n"
00755         "    address 127.0.0.1;\n"
00756         "}");
00757 }
00758 SECTION("Without explicit IP address (and maximum bitrate).")
00759 {
00760     UnixUDPSocket socket(14050, {}, 8192, mock_unique(mock_sys));
00761     REQUIRE(
00762         str(socket) ==
00763         "udp (\n"
00764         "    port 14050;\n"
00765         "    max_bitrate 8192;\n"
00766         "}");
00767 }
00768 SECTION("With explicit IP address (and maximum bitrate).")
00769 {
00770     UnixUDPSocket socket(
00771         14050, IPAddress("127.0.0.1"), 8192, mock_unique(mock_sys));
00772     REQUIRE(
00773         str(socket) ==
00774         "udp (\n"
00775         "    port 14050;\n"
00776         "    address 127.0.0.1;\n"
00777         "    max_bitrate 8192;\n"
00778         "}");
00779 }
0080 }

```

16.273 test_utility.cpp File Reference

```
#include <vector>
#include <catch.hpp>
```

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



Functions

- `TEST_CASE ("append' appends one vector to another.", "[utility]")`
- `TEST_CASE ("append' (with move semantics) appends one vector to another.", "[utility]")`
- `TEST_CASE ("to_bytes' converts numeric types to bytes.", "[utility]")`
- `TEST_CASE ("to_lower' converts string to lower case.", "[utility]")`
- `TEST_CASE ("str' converts printable types to strings.", "[utility]")`
- `TEST_CASE ("operator<<' makes vectors printable", "[utility]")`

16.273.1 Function Documentation

16.273.1.1 TEST_CASE() [1/6]

```
TEST_CASE (
    "'append' appends one vector to another." ,
    "" [utility] )
```

Definition at line 25 of file [test_utility.cpp](#).

16.273.1.2 TEST_CASE() [2/6]

```
TEST_CASE (
    "'append' (with move semantics) appends one vector to another." ,
    "" [utility] )
```

Definition at line 66 of file [test_utility.cpp](#).

16.273.1.3 TEST_CASE() [3/6]

```
TEST_CASE (
    "'to_bytes' converts numeric types to bytes." ,
    "" [utility] )
```

Definition at line 108 of file [test_utility.cpp](#).

16.273.1.4 TEST_CASE() [4/6]

```
TEST_CASE (
    "'to_lower' converts string to lower case." ,
    "" [utility] )
```

Definition at line 200 of file [test_utility.cpp](#).

16.273.1.5 TEST_CASE() [5/6]

```
TEST_CASE (
    "'str' converts printable types to strings." ,
    "" [utility] )
```

Definition at line 209 of file [test_utility.cpp](#).

16.273.1.6 TEST_CASE() [6/6]

```
TEST_CASE (
    "'operator<<' makes vectors printable" ,
    "" [utility] )
```

Definition at line 217 of file [test_utility.cpp](#).

16.274 test_utility.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <vector>
00019
00020 #include <catch.hpp>
00021
00022 #include "utility.hpp"
00023
00024
00025 TEST_CASE("'append' appends one vector to another.", "[utility]")
00026 {
00027     SECTION("{} + {} = {}")
00028     {
00029         std::vector<int> vec_1;
00030         std::vector<int> vec_2;
00031         REQUIRE(vec_1.empty());
00032         REQUIRE(vec_2.empty());
00033         append(vec_1, vec_2);
00034         REQUIRE(vec_1.empty());
00035     }
00036     SECTION("{1, 2, 3, 4} + {} = {1, 2, 3, 4}")
00037     {
00038         std::vector<int> vec_1 = {1, 2, 3, 4};
00039         std::vector<int> vec_2;
00040         REQUIRE(vec_1 == std::vector<int>({1, 2, 3, 4}));
00041         REQUIRE(vec_2.empty());
00042         append(vec_1, vec_2);
00043         REQUIRE(vec_1 == std::vector<int>({1, 2, 3, 4}));
00044     }
00045     SECTION("{} + {1, 2, 3, 4} = {1, 2, 3, 4}")
00046     {
00047         std::vector<int> vec_1;
00048         std::vector<int> vec_2 = {1, 2, 3, 4};
00049         REQUIRE(vec_1.empty());
00050         REQUIRE(vec_2 == std::vector<int>({1, 2, 3, 4}));
00051         append(vec_1, vec_2);
00052         REQUIRE(vec_1 == std::vector<int>({1, 2, 3, 4}));
00053     }
00054     SECTION("{1, 2, 3, 4} + {5, 6, 7, 8} = {1, 2, 3, 4, 5, 6, 7, 8}")
00055     {
00056         std::vector<int> vec_1 = {1, 2, 3, 4};
00057         std::vector<int> vec_2 = {5, 6, 7, 8};
00058         REQUIRE(vec_1 == std::vector<int>({1, 2, 3, 4}));
00059         REQUIRE(vec_2 == std::vector<int>({5, 6, 7, 8}));
00060         append(vec_1, vec_2);
00061         REQUIRE(vec_1 == std::vector<int>({1, 2, 3, 4, 5, 6, 7, 8}));
00062     }
00063 }
00064
00065
00066 TEST_CASE("'append' (with move semantics) appends one vector to another.",
00067             "[utility]")
00068 {
00069     SECTION("{} + {} = {}")
00070     {
00071         std::vector<int> vec_1;
00072         std::vector<int> vec_2;
00073         REQUIRE(vec_1.empty());
00074         REQUIRE(vec_2.empty());
00075         append(vec_1, std::move(vec_2));
00076         REQUIRE(vec_1.empty());
00077     }
```

```

00078     SECTION("{1, 2, 3, 4} + {} = {1, 2, 3, 4}")
00079     {
00080         std::vector<int> vec_1 = {1, 2, 3, 4};
00081         std::vector<int> vec_2;
00082         REQUIRE(vec_1 == std::vector<int>({1, 2, 3, 4}));
00083         REQUIRE(vec_2.empty());
00084         append(vec_1, std::move(vec_2));
00085         REQUIRE(vec_1 == std::vector<int>({1, 2, 3, 4}));
00086     }
00087     SECTION("{} + {1, 2, 3, 4} = {1, 2, 3, 4}")
00088     {
00089         std::vector<int> vec_1;
00090         std::vector<int> vec_2 = {1, 2, 3, 4};
00091         REQUIRE(vec_1.empty());
00092         REQUIRE(vec_2 == std::vector<int>({1, 2, 3, 4}));
00093         append(vec_1, std::move(vec_2));
00094         REQUIRE(vec_1 == std::vector<int>({1, 2, 3, 4}));
00095     }
00096     SECTION("{1, 2, 3, 4} + {5, 6, 7, 8} = {1, 2, 3, 4, 5, 6, 7, 8}")
00097     {
00098         std::vector<int> vec_1 = {1, 2, 3, 4};
00099         std::vector<int> vec_2 = {5, 6, 7, 8};
00100        REQUIRE(vec_1 == std::vector<int>({1, 2, 3, 4}));
00101        REQUIRE(vec_2 == std::vector<int>({5, 6, 7, 8}));
00102        append(vec_1, std::move(vec_2));
00103        REQUIRE(vec_1 == std::vector<int>({1, 2, 3, 4, 5, 6, 7, 8}));
00104    }
00105 }
00106
00107
00108 TEST_CASE("'to_bytes' converts numeric types to bytes.", "[utility]")
00109 {
00110     SECTION("char's can be converted to at least 1 bytes")
00111     {
00112         auto bytes = to_bytes(static_cast<char>(0x89));
00113         REQUIRE(bytes.size() >= 1);
00114         REQUIRE(bytes[0] == 0x89);
00115     }
00116     SECTION("unsigned char's can be converted to at least 1 byte")
00117     {
00118         auto bytes = to_bytes(static_cast<unsigned char>(0x89u));
00119         REQUIRE(bytes.size() >= 1);
00120         REQUIRE(bytes[0] == 0x89);
00121     }
00122     SECTION("short's can be converted to at least 2 bytes")
00123     {
00124         auto bytes = to_bytes(static_cast<short>(0xFACE));
00125         REQUIRE(bytes.size() >= 2);
00126         REQUIRE(bytes[0] == 0xCE);
00127         REQUIRE(bytes[1] == 0xFA);
00128     }
00129     SECTION("unsigned short's can be converted to at least 2 bytes")
00130     {
00131         auto bytes = to_bytes(static_cast<unsigned short>(0xFACEu));
00132         REQUIRE(bytes.size() >= 2);
00133         REQUIRE(bytes[0] == 0xCE);
00134         REQUIRE(bytes[1] == 0xFA);
00135     }
00136     SECTION("int's can be converted to at least 2 bytes")
00137     {
00138         auto bytes = to_bytes(static_cast<int>(0xFACE));
00139         REQUIRE(bytes.size() >= 2);
00140         REQUIRE(bytes[0] == 0xCE);
00141         REQUIRE(bytes[1] == 0xFA);
00142     }
00143     SECTION("unsigned int's can be converted to at least 2 bytes")
00144     {
00145         auto bytes = to_bytes(static_cast<unsigned int>(0xFACEu));
00146         REQUIRE(bytes.size() >= 2);
00147         REQUIRE(bytes[0] == 0xCE);
00148         REQUIRE(bytes[1] == 0xFA);
00149     }
00150     SECTION("long's can be converted to at least 4 bytes")
00151     {
00152         auto bytes = to_bytes(static_cast<long>(0xBA5EBA11));
00153         REQUIRE(bytes.size() >= 4);
00154         REQUIRE(bytes[0] == 0x11);
00155         REQUIRE(bytes[1] == 0xBA);
00156         REQUIRE(bytes[2] == 0x5E);
00157         REQUIRE(bytes[3] == 0xBA);
00158     }

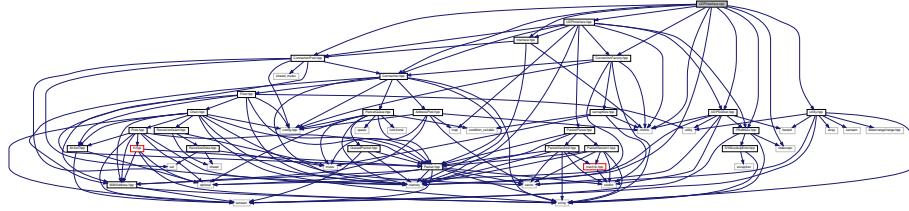
```

```
00159     SECTION("unsigned long's can be converted to at least 4 bytes")
00160     {
00161         unsigned long i = 0xBA5EBA11;
00162         REQUIRE(0xBA5EBA11 == i);
00163         auto bytes = to_bytes(static_cast<unsigned long>(0xBA5EBA11u));
00164         REQUIRE(bytes.size() >= 4);
00165         REQUIRE(bytes[0] == 0x11);
00166         REQUIRE(bytes[1] == 0xBA);
00167         REQUIRE(bytes[2] == 0x5E);
00168         REQUIRE(bytes[3] == 0xBA);
00169     }
00170     SECTION("long long's can be converted to at least 8 bytes")
00171     {
00172         auto bytes = to_bytes(static_cast<long long>(0x0123456789ABCDEF));
00173         REQUIRE(bytes.size() >= 8);
00174         REQUIRE(bytes[0] == 0xEF);
00175         REQUIRE(bytes[1] == 0xCD);
00176         REQUIRE(bytes[2] == 0xAB);
00177         REQUIRE(bytes[3] == 0x89);
00178         REQUIRE(bytes[4] == 0x67);
00179         REQUIRE(bytes[5] == 0x45);
00180         REQUIRE(bytes[6] == 0x23);
00181         REQUIRE(bytes[7] == 0x01);
00182     }
00183     SECTION("unsigned long long's can be converted to at least 8 bytes")
00184     {
00185         auto bytes =
00186             to_bytes(static_cast<unsigned long long>(0x0123456789ABCDEFu));
00187         REQUIRE(bytes.size() >= 8);
00188         REQUIRE(bytes[0] == 0xEF);
00189         REQUIRE(bytes[1] == 0xCD);
00190         REQUIRE(bytes[2] == 0xAB);
00191         REQUIRE(bytes[3] == 0x89);
00192         REQUIRE(bytes[4] == 0x67);
00193         REQUIRE(bytes[5] == 0x45);
00194         REQUIRE(bytes[6] == 0x23);
00195         REQUIRE(bytes[7] == 0x01);
00196     }
00197 }
00198
00199
00200 TEST_CASE("'to_lower' converts string to lower case.", "[utility]")
00201 {
00202     REQUIRE(to_lower("HELLO WORLD") == "hello world");
00203     REQUIRE(to_lower("Hello World") == "hello world");
00204     REQUIRE_NO_THROW(
00205         to_lower("1234567891!@#$%^&*()_+") == "1234567890!@#$%^&*()_+");
00206 }
00207
00208
00209 TEST_CASE("'str' converts printable types to strings.", "[utility]")
00210 {
00211     REQUIRE(str(256) == "256");
00212     REQUIRE(str(3.14159) == "3.14159");
00213     REQUIRE(str("Hello world") == "Hello world");
00214 }
00215
00216
00217 TEST_CASE("'operator<<' makes vectors printable", "[utility]")
00218 {
00219     SECTION("When the vector is empty.")
00220     {
00221         std::vector<int> vec = {};
00222         REQUIRE(str(vec) == "[ ]");
00223     }
00224     SECTION("When the vector has a single element.")
00225     {
00226         std::vector<int> vec = {1};
00227         REQUIRE(str(vec) == "[1]");
00228     }
00229     SECTION("When the vector has two elements.")
00230     {
00231         std::vector<int> vec = {1, 2};
00232         REQUIRE(str(vec) == "[1, 2]");
00233     }
00234     SECTION("When the vector has multiple elements.")
00235     {
00236         std::vector<int> vec = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
00237         REQUIRE(str(vec) == "[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]");
00238     }
00239 }
```

16.275 UDPInterface.cpp File Reference

```
#include <chrono>
#include <memory>
#include <stdexcept>
#include "Connection.hpp"
#include "ConnectionFactory.hpp"
#include "ConnectionPool.hpp"
#include "Interface.hpp"
#include "IPAddress.hpp"
#include "UDPInterface.hpp"
#include "UDPSocket.hpp"
#include "utility.hpp"
```

Include dependency graph for UDPInterface.cpp:



16.276 UDPInterface.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <chrono>
00019 #include <memory>
00020 #include <stdexcept>
00021
00022 #include "Connection.hpp"
00023 #include "ConnectionFactory.hpp"
00024 #include "ConnectionPool.hpp"
00025 #include "Interface.hpp"
00026 #include "IPAddress.hpp"
00027 #include "UDPInterface.hpp"
00028 #include "UDPSocket.hpp"
00029 #include "utility.hpp"
00030
00031
00032 using namespace std::chrono_literals;
00033
00034
00035 /** Update connections.
00036 *
00037 * Adds a MAVLink address to the connection corresponding to the given IP
```

```
00038 * address. If this connection does not exist, it will be constructed using
00039 * the \ref UDPInterface's connection factory.
00040 *
00041 * \param mav_address The MAVLink address of the received packet.
00042 * \param ip_address The IP address the packet was received on.
00043 */
00044 void UDPInterface::update_connections_()
00045     const MAVAddress &mav_address, const IPAddress &ip_address)
00046 {
00047     auto it = connections_.find(ip_address);
00048
00049     if (it == connections_.end())
00050     {
00051         it = connections_.insert(
00052             {ip_address, connection_factory_->get(str(ip_address))}).first;
00053         connection_pool_->add(it->second);
00054     }
00055
00056     it->second->add_address(mav_address);
00057 }
00058
00059
00060 /** Construct a UDP interface using a given socket.
00061 *
00062 * \param socket The UDP socket to communicate over.
00063 * \param connection_pool The connection pool to use for sending packets and to
00064 * register new connections with.
00065 * \param connection_factory The connection factory to use for constructing
00066 * new connections when an outside connection is made.
00067 * \throws std::invalid_argument if the serial \p port device pointer is null.
00068 * \throws std::invalid_argument if the \p connection_pool pointer is null.
00069 * \throws std::invalid_argument if the \p connection_factory pointer is null.
00070 */
00071 UDPInterface::UDPInterface(
00072     std::unique_ptr<UDPSocket> socket,
00073     std::shared_ptr<ConnectionPool> connection_pool,
00074     std::unique_ptr<ConnectionFactory> connection_factory)
00075     : socket_(std::move(socket)),
00076     connection_pool_(std::move(connection_pool)),
00077     connection_factory_(std::move(connection_factory)),
00078     last_ip_address_(IPAddress(0))
00079 {
00080     if (socket_ == nullptr)
00081     {
00082         throw std::invalid_argument("Given socket pointer is null.");
00083     }
00084
00085     if (connection_pool_ == nullptr)
00086     {
00087         throw std::invalid_argument("Given connection pool pointer is null.");
00088     }
00089
00090     if (connection_factory_ == nullptr)
00091     {
00092         throw std::invalid_argument(
00093             "Given connection factory pointer is null.");
00094     }
00095 }
00096
00097
00098 /** \copydoc Interface::send_packet(const std::chrono::nanoseconds &)
00099 *
00100 * Sends up to one packet from each connection, belonging to the interface,
00101 * over the UDP socket.
00102 */
00103 void UDPInterface::send_packet(const std::chrono::nanoseconds &timeout)
00104 {
00105     bool not_first = false;
00106
00107     // Wait for a packet on any of the interface's connections.
00108     if (connection_factory_->wait_for_packet(timeout))
00109     {
00110         for (auto &conn : connections_)
00111         {
00112             auto packet = conn.second->next_packet();
00113
00114             // If connection has a packet send it.
00115             if (packet != nullptr)
00116             {
00117                 socket_->send(packet->data(), conn.first);
00118             }
00119         }
00120     }
00121 }
```

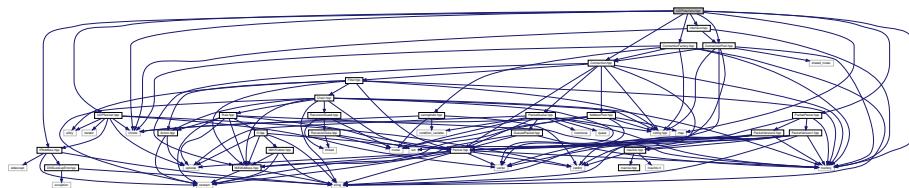
```

00119             if (not_first)
00120             {
00121                 // Decrement semaphore once for each extra packet.
00122                 connection_factory_>wait_for_packet(0s);
00123             }
00124
00125             not_first = true;
00126         }
00127     }
00128 }
00129 */
00130
00131 /**
00132  * \copydoc Interface::receive_packet(const std::chrono::nanoseconds &)
00133  *
00134  * Receives up to one UDP packet worth of data and parses it into MAVLink
00135  * packets before passing these packets onto the connection pool. Will wait up
00136  * to \p timeout for a UDP packet to be received.
00137 */
00138 void UDPInterface::receive_packet(const std::chrono::nanoseconds &timeout)
00139 {
00140     auto [buffer, ip_address] = socket_->receive(timeout);
00141
00142     if (!buffer.empty())
00143     {
00144         // Clear the parser if the IP address is different from the last UDP
00145         // packet received (we want complete MAVLink packets).
00146         if (ip_address != last_ip_address_)
00147         {
00148             parser_.clear();
00149             last_ip_address_ = ip_address;
00150         }
00151
00152         // Parse the bytes.
00153         for (auto byte : buffer)
00154         {
00155             auto packet = parser_.parse_byte(byte);
00156
00157             if (packet != nullptr)
00158             {
00159                 update_connections_(packet->source(), ip_address);
00160                 // It is a post condition of update_connections_ that there is a
00161                 // connection for ip_address.
00162                 packet->connection(connections_[ip_address]);
00163                 connection_pool_->send(std::move(packet));
00164             }
00165         }
00166     }
00167 }
00168
00169 /**
00170  * \copydoc Interface::print_(std::ostream &os) const
00171  *
00172  * Example:
00173  *   ``
00174  *   udp {
00175  *       port 14500;
00176  *       address 127.0.0.1;
00177  *   }
00178  *   ``
00179 */
00180 std::ostream &UDPInterface::print_(std::ostream &os) const
00181 {
00182     os << *socket_;
00183     return os;
00184 }
```

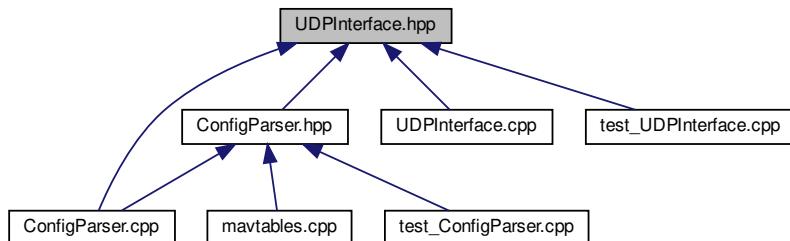
16.277 UDPInterface.hpp File Reference

```
#include <chrono>
#include <map>
#include <memory>
#include "Connection.hpp"
```

```
#include "ConnectionFactory.hpp"
#include "ConnectionPool.hpp"
#include "Interface.hpp"
#include "IPAddress.hpp"
#include "PacketParser.hpp"
#include "UDPSocket.hpp"
Include dependency graph for UDPInterface.hpp:
```



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



Classes

- class [UDPInterface](#)

16.278 UDPInterface.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
```

```

00017
00018 #ifndef UDPINTERFACE_HPP_
00019 #define UDPINTERFACE_HPP_
00020
00021
00022 #include <chrono>
00023 #include <map>
00024 #include <memory>
00025
00026 #include "Connection.hpp"
00027 #include "ConnectionFactory.hpp"
00028 #include "ConnectionPool.hpp"
00029 #include "Interface.hpp"
00030 #include "IPAddress.hpp"
00031 #include "PacketParser.hpp"
00032 #include "UDPSocket.hpp"
00033
00034
00035 /** A UDP (User Datagram Protocol) interface.
00036 *
00037 * An interface (for sending and receiving packets) implementing the user
00038 * datagram protocol.
00039 */
00040 class UDPIface : public Interface
00041 {
00042     public:
00043         UDPIface(
00044             std::unique_ptr<UDPSocket> socket,
00045             std::shared_ptr<ConnectionPool> connection_pool,
00046             std::unique_ptr<ConnectionFactory>> connection_factory);
00047     // LCOV_EXCL_START
00048     ~UDPIface() = default;
00049     // LCOV_EXCL_STOP
00050     void send_packet(const std::chrono::nanoseconds &timeout) final;
00051     void receive_packet(const std::chrono::nanoseconds &timeout) final;
00052
00053     protected:
00054         std::ostream &print_(std::ostream &os) const final;
00055
00056     private:
00057         // Variables.
00058         std::unique_ptr<UDPSocket> socket_;
00059         std::shared_ptr<ConnectionPool> connection_pool_;
00060         std::unique_ptr<ConnectionFactory>> connection_factory_;
00061         IPAddress last_ip_address_;
00062         std::map<IPAddress, std::shared_ptr<Connection>> connections_;
00063         PacketParser parser_;
00064         // Methods
00065         void update_connections_(
00066             const MAVAddress &mav_address, const IPAddress &ip_address);
00067     };
00068
00069
00070 #endif // UDPINTERFACE_HPP_

```

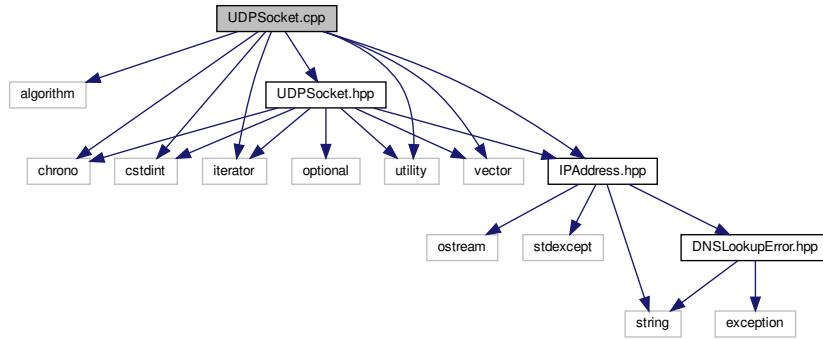
16.279 UDPSocket.cpp File Reference

```

#include <algorithm>
#include <chrono>
#include <cstdint>
#include <iterator>
#include <utility>
#include <vector>
#include "IPAddress.hpp"
#include "UDPSocket.hpp"

```

Include dependency graph for UDPSocket.cpp:



16.280 UDPSocket.cpp

```

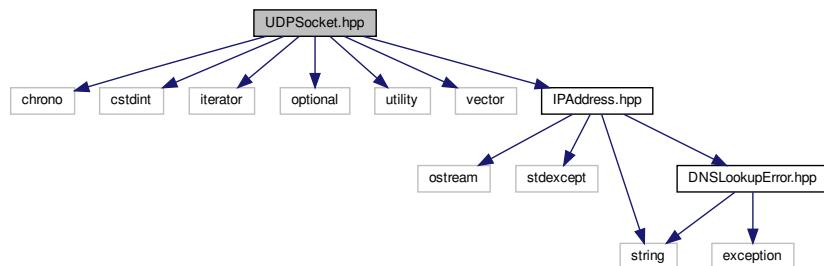
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <algorithm>
00019 #include <chrono>
00020 #include <cstdint>
00021 #include <iterator>
00022 #include <utility>
00023 #include <vector>
00024
00025 #include "IPAddress.hpp"
00026 #include "UDPSocket.hpp"
00027
00028
00029 // Placed here to avoid weak-vtables error.
00030 // LCOV_EXCL_START
00031 UDPSocket::~UDPSocket()
00032 {
00033 }
00034 // LCOV_EXCL_STOP
00035
00036
00037 /** Send data to the given address using the socket.
00038 *
00039 * \param data The bytes to send.
00040 * \param address The IP address (with port number) to send the bytes to, using
00041 * UDP.
00042 */
00043 void UDPSocket::send(const std::vector<uint8_t> &data, const
00044 IPAddress &address)
00045 {
00046     send(data.begin(), data.end(), address);
00047 }
00048
  
```

```
00048 /**
00049  ** Send data to the given address using the socket.
00050  *
00051  * \param first Iterator to first byte in range to send.
00052  * \param last Iterator to one past the last byte to send.
00053  * \param address The IP address (with port number) to send the bytes to, using
00054  *     UDP.
00055  */
00056 void UDPSocket::send(
00057     std::vector<uint8_t>::const_iterator first,
00058     std::vector<uint8_t>::const_iterator last,
00059     const IPAddress &address)
00060 {
00061     std::vector<uint8_t> vec;
00062     std::copy(first, last, std::back_inserter(vec));
00063     send(vec, address);
00064 }
00065
00066
00067 /** Receive data on the socket.
00068  *
00069  * \note The \p timeout is not guaranteed to be up to nanosecond precision, the
00070  *       actual precision is up to the operating system's implementation but is
00071  *       guaranteed to have at least millisecond precision.
00072  *
00073  * \param timeout How long to wait for data to arrive on the socket. The
00074  *                 default is to not wait.
00075  * \returns The data read from the socket and the IP address it was sent from.
00076 */
00077 std::pair<std::vector<uint8_t>, IPAddress> UDPSocket::receive(
00078     const std::chrono::nanoseconds &timeout)
00079 {
00080     std::vector<uint8_t> vec;
00081     return {vec, receive(std::back_inserter(vec), timeout)};
00082 }
00083
00084
00085 /** Receive data on the socket.
00086  *
00087  * \note The \p timeout is not guaranteed to be up to nanosecond precision, the
00088  *       actual precision is up to the operating system's implementation but is
00089  *       guaranteed to have at least millisecond precision.
00090  *
00091  * \param it A back insert iterator to read bytes into.
00092  * \param timeout How long to wait for data to arrive on the socket.
00093  *               The default is to not wait.
00094  * \returns The IP address the data was sent from, this is where a
00095  *         reply should be sent to.
00096 */
00097 IPAddress UDPSocket::receive(
00098     std::back_insert_iterator<std::vector<uint8_t>> it,
00099     const std::chrono::nanoseconds &timeout)
00100 {
00101     auto [vec, address] = receive(timeout);
00102     std::copy(vec.begin(), vec.end(), it);
00103     return address;
00104 }
00105
00106
00107 /** Print the UDP socket to the given output stream.
00108  *
00109  * \param os The output stream to print to.
00110  * \returns The output stream.
00111 */
00112 std::ostream &UDPSocket::print_(std::ostream &os) const
00113 {
00114     os << "unknown UDP socket";
00115     return os;
00116 }
00117
00118
00119 /** Print the given UDP socket to the given output stream.
00120  *
00121  * An example:
00122  * ``
00123  * udp {
00124  *     port 14555;
00125  *     address 127.0.0.1;
00126  *     max_bitrate 262144;
00127  * }
00128 * ``
00129
```

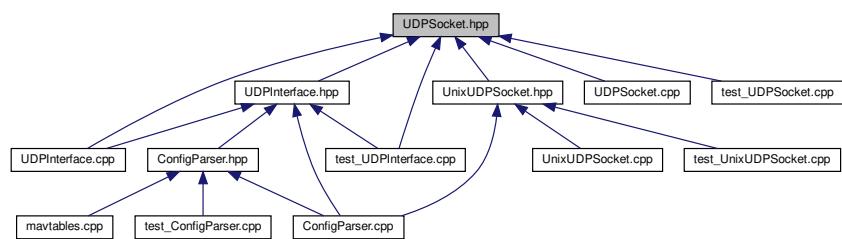
```
00129 *  
00130 * The base \ref UDPSocket class will print:  
00131 * ```  
00132 * unknown UDP socket  
00133 * ```  
00134 *  
00135 * \relates UDPSocket  
00136 * \param os The output stream to print to.  
00137 * \param udp_socket The UDP socket (or any child of \ref UDPSocket) to print.  
00138 * \returns The output stream.  
00139 */  
00140 std::ostream &operator<<(std::ostream &os, const UDPSocket &udp_socket)  
00141 {  
00142     return udp_socket.print_(os);  
00143 }
```

16.281 UDP Socket.hpp File Reference

```
#include <chrono>
#include <cstdint>
#include <iterator>
#include <optional>
#include <utility>
#include <vector>
#include "IPAddress.hpp"
Include dependency graph for UDPSocket.hpp
```



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



Classes

- class [UDPSocket](#)

Functions

- std::ostream & [operator<<](#) (std::ostream &os, const [UDPSocket](#) &udp_socket)

16.281.1 Function Documentation

16.281.1.1 [operator<<\(\)](#)

```
std::ostream& operator<< (
    std::ostream & os,
    const UDPSocket & udp_socket )
```

16.282 [UDPSocket.hpp](#)

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef UDP SOCKET _HPP_
00019 #define UDP SOCKET _HPP_
00020
00021
00022 #include <chrono>
00023 #include <cstdint>
00024 #include <iterator>
00025 #include <optional>
00026 #include <utility>
00027 #include <vector>
00028
00029 #include "IPAddress.hpp"
00030
00031
00032 /** A UDP socket, listening on a port/address combination.
00033 *
00034 * \warning This class should be treated as pure virtual and should never be
00035 * instantiated.
00036 *
00037 * \warning Either \ref send(const std::vector<uint8_t> &data, const IPAddress &) or
00038 * send(std::vector<uint8_t>::const_iterator, std::vector<uint8_t>::const_iterator, const IPAddress &)
00039 * must be overridden in child classes to avoid infinite recursion.
00040 *
```

```

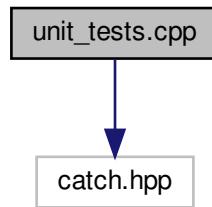
00041 * \warning Either \ref receive(const std::chrono::nanoseconds &) or
00042 *     receive(std::back_insert_iterator<std::vector<uint8_t>>, const std::chrono::nanoseconds &)
00043 *         must be overridden in child classes to avoid infinite recursion.
00044 *
00045 */
00046 class UDPSocket
00047 {
00048     public:
00049         virtual ~UDPSocket();
00050         virtual void send(
00051             const std::vector<uint8_t> &data, const IPAddress &address);
00052         virtual void send(
00053             std::vector<uint8_t>::const_iterator first,
00054             std::vector<uint8_t>::const_iterator last,
00055             const IPAddress &address);
00056         virtual std::pair<std::vector<uint8_t>, IPAddress> receive(
00057             const std::chrono::nanoseconds &timeout =
00058                 std::chrono::nanoseconds::zero());
00059         virtual IPAddress receive(
00060             std::back_insert_iterator<std::vector<uint8_t>> it,
00061             const std::chrono::nanoseconds &timeout =
00062                 std::chrono::nanoseconds::zero());
00063
00064     friend std::ostream &operator<<(
00065         std::ostream &os, const UDPSocket &udp_socket);
00066
00067     protected:
00068         /** Print the UDP socket to the given output stream.
00069         *
00070         * \param os The output stream to print to.
00071         * \returns The output stream.
00072         */
00073         virtual std::ostream &print_(std::ostream &os) const;
00074     };
00075
00076
00077 std::ostream &operator<<(std::ostream &os, const UDPSocket &udp_socket);
00078
00079
00080 #endif // UDPSOCKET_HPP_

```

16.283 unit_tests.cpp File Reference

#include "catch.hpp"

Include dependency graph for unit_tests.cpp:



Macros

- `#define CATCH_CONFIG_MAIN`

16.283.1 Macro Definition Documentation

16.283.1.1 CATCH_CONFIG_MAIN

```
#define CATCH_CONFIG_MAIN
```

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

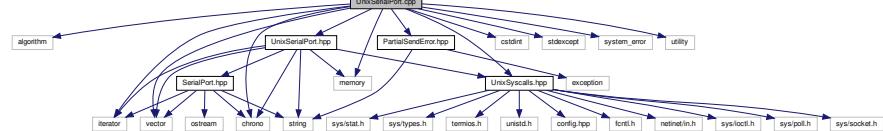
16.284 unit_tests.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #define CATCH_CONFIG_MAIN
00019 #include "catch.hpp"
```

16.285 UnixSerialPort.cpp File Reference

```
#include <algorithm>
#include <chrono>
#include <cstdint>
#include <iterator>
#include <memory>
#include <stdexcept>
#include <system_error>
#include <utility>
#include <vector>
#include "PartialSendError.hpp"
#include "UnixSerialPort.hpp"
#include "UnixSyscalls.hpp"
Include dependency graph for UnixSerialPort.cpp
```

include dependency graph for ChixConan-0.1.0.hpp.



16.286 UnixSerialPort.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <algorithm>
00019 #include <chrono>
00020 #include <cstdint>
00021 #include <iterator>
00022 #include <memory>
00023 #include <stdexcept>
00024 #include <system_error>
00025 #include <utility>
00026 #include <vector>
00027
00028 #include "PartialSendError.hpp"
00029 #include "UnixSerialPort.hpp"
00030 #include "UnixSyscalls.hpp"
00031
00032
00033 /** Construct a serial port.
00034 *
00035 * \param device The string representing the serial port. For example
00036 *   "/dev/ttyUSB0".
00037 * \param baud_rate The baud rate in bits per second, the default value is 9600
00038 *   bps.
00039 * \param features A bitflag of the features to enable, default is to not
00040 *   enable any features. See \ref SerialPort::Feature for flags.
00041 * \param syscalls The object to use for unix system calls. It is default
00042 *   constructed to the production implementation. This argument is only
00043 *   used for testing.
00044 * \throws std::invalid_argument if the baud rate is not supported.
00045 * \throws std::system_error if a system call produces an error.
00046 */
00047 UnixSerialPort::UnixSerialPort(
00048     std::string device,
00049     unsigned long baud_rate,
00050     SerialPort::Feature features,
00051     std::unique_ptr<UnixSyscalls> syscalls)
00052     : device_(std::move(device)), baud_rate_(baud_rate),
00053       features_(features), syscalls_(std::move(syscalls)), port_(-1)
00054 {
00055     open_port_();
00056 }
00057
00058
00059 /** The serial port destructor.
00060 *
00061 * Closes the underlying file descriptor of the serial port device.
00062 */
00063 // LCOV_EXCL_START
00064 UnixSerialPort::~UnixSerialPort()
00065 {
00066     syscalls_->close(port_);
00067 }
00068 // LCOV_EXCL_STOP
00069
00070
00071 /** \copydoc SerialPort::read(const std::chrono::nanoseconds &)
00072 *
00073 * \note The timeout precision of this implementation is 1 millisecond.
00074 *
00075 * \throws std::system_error if a system call produces an error.
00076 */
00077 std::vector<uint8_t> UnixSerialPort::read(
```

```

00078     const std::chrono::nanoseconds &timeout)
00079     {
00080         std::chrono::milliseconds timeout_ms =
00081             std::chrono::duration_cast<std::chrono::milliseconds>(timeout);
00082         struct pollfd fds = {port_, POLLIN, 0};
00083         auto result = syscalls_->poll(
00084             &fds, 1, static_cast<int>(timeout_ms.count()));
00085
00086         // Poll error
00087         if (result < 0)
00088         {
00089             throw std::system_error(std::error_code(errno, std::system_category()));
00090         }
00091         // Success
00092         else if (result > 0)
00093         {
00094             // Port error
00095             if (fds.revents & POLLERR)
00096             {
00097                 syscalls_->close(port_);
00098                 open_port_();
00099                 return std::vector<uint8_t>();
00100             }
00101             // Data available for reading.
00102             else if (fds.revents & POLLIN)
00103             {
00104                 return read_();
00105             }
00106         }
00107
00108         // Timed out
00109         return std::vector<uint8_t>();
00110     }
00111
00112
00113 /** \copydoc SerialPort::write(const std::vector<uint8_t> &)
00114 *
00115 * \throws std::system_error if a system call produces an error.
00116 * \throws PartialSendError if it fails to write all the data it is given.
00117 */
00118 void UnixSerialPort::write(const std::vector<uint8_t> &data)
00119 {
00120     // Write the data.
00121     auto err = syscalls_->write(port_, data.data(), data.size());
00122
00123     // Handle system call errors.
00124     if (err < 0)
00125     {
00126         throw std::system_error(std::error_code(errno, std::system_category()));
00127     }
00128
00129     // Could not write all data.
00130     if (static_cast<size_t>(err) < data.size())
00131     {
00132         throw PartialSendError(static_cast<unsigned long>(err), data.size());
00133     }
00134 }
00135
00136 /** Configure serial port.
00137 *
00138 * \param baud_rate The bitrate to configure for the port.
00139 * \param features A bitmask representing the features to use. See \ref
00140 *     SerialPort::Feature for documentation.
00141 * \throws std::invalid_argument if the baud rate is not supported.
00142 * \throws std::system_error if a system call produces an error.
00143 */
00144 void UnixSerialPort::configure_port_
00145     unsigned long baud_rate, SerialPort::Feature features)
00146 {
00147     // Get attribute structure.
00148     struct termios tty;
00149
00150     if (syscalls_->tcgetattr(port_, &tty) < 0)
00151     {
00152         throw std::system_error(std::error_code(errno, std::system_category()));
00153     }
00154
00155     // Set baud rate.
00156     speed_t speed = speed_constant_(baud_rate);
00157
00158     if (cfsetispeed(&tty, speed) < 0)

```

```

00159     {
00160         // This is unreachable assuming the speed_constant_ method is properly
00161         // written.
00162         // LCOV_EXCL_START
00163         throw std::system_error(std::error_code(errno, std::system_category()));
00164         // LCOV_EXCL_STOP
00165     }
00166
00167     if (cfsetospeed(&tty, speed) < 0)
00168     {
00169         // This is unreachable assuming the speed_constant_ method is properly
00170         // written.
00171         // LCOV_EXCL_START
00172         throw std::system_error(std::error_code(errno, std::system_category()));
00173         // LCOV_EXCL_STOP
00174     }
00175
00176     // Enable receiver and set local mode.
00177     tty.c_cflag |= static_cast<tcflag_t>(CLOCAL | CREAD);
00178     // Use 8N1 mode (8 data bits, no parity, 1 stop bit).
00179     tty.c_cflag &= static_cast<tcflag_t>(~PARENB);
00180     tty.c_cflag &= static_cast<tcflag_t>(~CSTOPB);
00181     tty.c_cflag &= static_cast<tcflag_t>(~CSIZE);
00182     tty.c_cflag |= static_cast<tcflag_t>(CS8);
00183
00184     // Enable/disable hardware flow control.
00185     if (features & SerialPort::FLOW_CONTROL)
00186     {
00187         tty.c_cflag |= static_cast<tcflag_t>(CRTSCTS);
00188     }
00189     else
00190     {
00191         tty.c_cflag &= static_cast<tcflag_t>(~CRTSCTS);
00192     }
00193
00194     // Use raw input.
00195     tty.c_lflag &= static_cast<tcflag_t>(~(ICANON | ECHO | ECHOE | ISIG));
00196     tty.c_iflag &= static_cast<tcflag_t>(~(IGNBRK | BRKINT | PARMRK | ISTRIP));
00197     tty.c_iflag &= static_cast<tcflag_t>(~(INLCR | IGNCR | ICRNL));
00198     // Disable software flow control.
00199     tty.c_iflag &= static_cast<tcflag_t>(~(IXON | IXOFF | IXANY));
00200     // Use raw output.
00201     tty.c_oflag &= static_cast<tcflag_t>(~OPOST);
00202     // Non blocking mode, using poll.
00203     // See: http://unixwiz.net/techtips/termios-vmin-vtime.html
00204     tty.c_cc[VMIN] = 0;
00205     tty.c_cc[VTIME] = 0;
00206
00207     // Apply settings to serial port.
00208     if (syscalls_->tcsetattr(port_, TCSANOW, &tty) < 0)
00209     {
00210         throw std::system_error(std::error_code(errno, std::system_category()));
00211     }
00212 }
00213
00214
00215 /** Open the serial port.
00216 *
00217 * \throws std::invalid_argument if the baud rate is not supported.
00218 * \throws std::system_error if a system call produces an error.
00219 */
00220 void UnixSerialPort::open_port_()
00221 {
00222     port_ = -1;
00223     // Open serial port.
00224     port_ = syscalls_->open(device_.c_str(), O_RDWR | O_NOCTTY | O_SYNC);
00225
00226     if (port_ < 0)
00227     {
00228         throw std::system_error(
00229             std::error_code(errno, std::system_category()),
00230             "Failed to open \"" + device_ + "\".");
00231     }
00232
00233     // Configure serial port.
00234     try
00235     {
00236         configure_port_(baud_rate_, features_);
00237     }
00238     catch (...)
00239     {

```

```
00240         syscalls_->close(port_);
00241         throw;
00242     }
00243 }
00244
00245
00246 /** Read data from the serial port.
00247 *
00248 * \warning There must be data to read, otherwise calling this method is
00249 *         undefined.
00250 *
00251 * \returns The data read from the port, up to 1024 bytes at a time.
00252 * \throws std::system_error if a system call produces an error.
00253 */
00254 std::vector<uint8_t> UnixSerialPort::read_()
00255 {
00256     std::vector<uint8_t> buffer;
00257     buffer.resize(1024);
00258     auto size = syscalls_->read(port_, buffer.data(), buffer.size());
00259
00260     if (size < 0)
00261     {
00262         throw std::system_error(std::error_code(errno, std::system_category()));
00263     }
00264
00265     buffer.resize(static_cast<size_t>(size));
00266     return buffer;
00267 }
00268
00269
00270 /** Convert integer to termios baud rate constant.
00271 *
00272 * See \ref UnixSerialPort::UnixSerialPort for valid baud rates.
00273 *
00274 * \param baud_rate The baud rate to convert.
00275 * \returns The termios baud rate constant.
00276 * \throws std::invalid_argument if the given baud rate is not supported.
00277 */
00278 speed_t UnixSerialPort::speed_constant_(unsigned long baud_rate)
00279 {
00280     switch (baud_rate)
00281     {
00282         case 0:
00283             return B0;
00284
00285         case 50:
00286             return B50;
00287
00288         case 75:
00289             return B75;
00290
00291         case 110:
00292             return B110;
00293
00294         case 134: // actually 134.5
00295         case 135: // actually 134.5
00296             return B134;
00297
00298         case 150:
00299             return B150;
00300
00301         case 200:
00302             return B200;
00303
00304         case 300:
00305             return B300;
00306
00307         case 600:
00308             return B600;
00309
00310         case 1200:
00311             return B1200;
00312
00313         case 1800:
00314             return B1800;
00315
00316         case 2400:
00317             return B2400;
00318
00319         case 4800:
00320             return B4800;
```

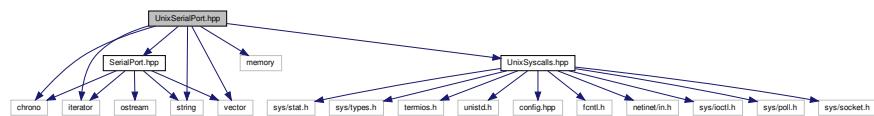
```

00321         case 9600:
00322             return B9600;
00323
00324         case 19200:
00325             return B19200;
00326
00327         case 38400:
00328             return B38400;
00329
00330         case 57600:
00331             return B57600;
00332
00333         case 115200:
00334             return B115200;
00335
00336         case 230400:
00337             return B230400;
00338
00339     default:
00340         throw std::invalid_argument(
00341             std::to_string(baud_rate) + " bps is not a valid baud rate.");
00342     }
00343 }
00345
00346
00347 /** \copydoc SerialPort::print_(std::ostream &os) const
00348 *
00349 * Example:
00350 * ``
00351 * serial {
00352 *   device /dev/ttyUSB0;
00353 *   baudrate 115200;
00354 *   flow_control yes;
00355 * }
00356 * ``
00357 *
00358 * \param os The output stream to print to.
00359 */
00360 std::ostream &UnixSerialPort::print_(std::ostream &os) const
00361 {
00362     os << "serial {" << std::endl;
00363     os << "    device " << device_ << ";" << std::endl;
00364     os << "    baudrate " << std::to_string(baud_rate_) << ";" << std::endl;
00365
00366     if ((features_ & SerialPort::FLOW_CONTROL) != 0)
00367     {
00368         os << "    flow_control yes;" << std::endl;
00369     }
00370     else
00371     {
00372         os << "    flow_control no;" << std::endl;
00373     }
00374
00375     os << "}";
00376     return os;
00377 }
```

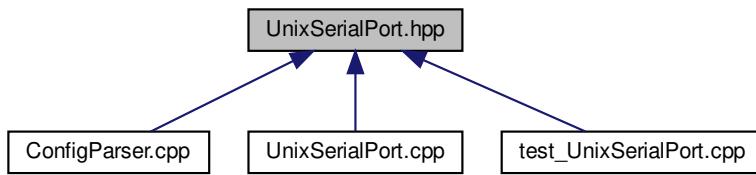
16.287 UnixSerialPort.hpp File Reference

```
#include <chrono>
#include <iterator>
#include <memory>
#include <string>
#include <vector>
#include "SerialPort.hpp"
#include "UnixSyscalls.hpp"
```

Include dependency graph for UnixSerialPort.hpp:



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



Classes

- class [UnixSerialPort](#)

16.288 UnixSerialPort.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef UNIXSERIALPORT_HPP_
00019 #define UNIXSERIALPORT_HPP_
00020
00021
00022 #include <chrono>
00023 #include <iterator>
00024 #include <memory>
00025 #include <string>
00026 #include <vector>
00027
00028 #include "SerialPort.hpp"
00029 #include "UnixSyscalls.hpp"
  
```

```

00030
00031
00032 /** A unix serial port.
00033  */
00034 class UnixSerialPort : public SerialPort
00035 {
00036     public:
00037         UnixSerialPort(
00038             std::string device,
00039             unsigned long baud_rate = 9600,
00040             SerialPort::Feature features =
00041             SerialPort::DEFAULT,
00042             std::unique_ptr<UnixSyscalls> syscalls =
00043             std::make_unique<UnixSyscalls>());
00044     virtual ~UnixSerialPort();
00045     virtual std::vector<uint8_t> read(
00046         const std::chrono::nanoseconds &timeout =
00047             std::chrono::nanoseconds::zero()) final;
00048     virtual void write(const std::vector<uint8_t> &data) final;
00049
00050     protected:
00051         std::ostream &print_(std::ostream &os) const final;
00052
00053     private:
00054         // Variables
00055         std::string device_;
00056         unsigned long baud_rate_;
00057         SerialPort::Feature features_;
00058         std::unique_ptr<UnixSyscalls> syscalls_;
00059         int port_;
00060         // Methods
00061         void configure_port_(
00062             unsigned long baud_rate, SerialPort::Feature features);
00063         void open_port_();
00064         std::vector<uint8_t> read_();
00065         speed_t speed_constant_(unsigned long baud_rate);
00066     };
00067
00068 #endif // UNIXSERIALPORT_HPP_

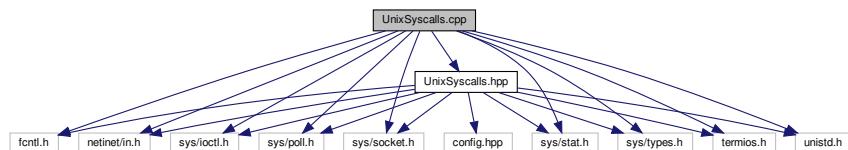
```

16.289 UnixSyscalls.cpp File Reference

```

#include <fcntl.h>
#include <netinet/in.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <termios.h>
#include <unistd.h>
#include "UnixSyscalls.hpp"
Include dependency graph for UnixSyscalls.cpp:

```



16.290 UnixSyscalls.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <fcntl.h>      // open, fcntl
00019 #include <netinet/in.h> // sockaddr_in
00020 #include <sys/ioctl.h>  // ioctl
00021 #include <sys/poll.h>   // poll
00022 #include <sys/socket.h> // socket, bind, sendto, recvfrom
00023 #include <sys/stat.h>   // fstat
00024 #include <sys/types.h>  // socklen_t type on old BSD systems
00025 #include <termios.h>    // terminal control
00026 #include <unistd.h>    // read, write, close
00027
00028 #include "UnixSyscalls.hpp"
00029
00030
00031 /** Bind a name to a socket.
00032 *
00033 * See [man 2 bind] (http://man7.org/linux/man-pages/man2/bind.2.html) for
00034 * documentation.
00035 *
00036 * \param sockfd Socket file descriptor.
00037 * \param addr Address to assign to socket.
00038 * \param addrlen Size of address structure in bytes.
00039 */
00040 int UnixSyscalls::bind(
00041     int sockfd, const struct sockaddr *addr, socklen_t addrlen)
00042 {
00043     return ::bind(sockfd, addr, addrlen);
00044 }
00045
00046
00047 /** Close a file descriptor.
00048 *
00049 * See [man 2 close] (http://man7.org/linux/man-pages/man2/close.2.html) for
00050 * documentation.
00051 *
00052 * \param fd The file descriptor to close.
00053 */
00054 int UnixSyscalls::close(int fd)
00055 {
00056     return ::close(fd);
00057 }
00058
00059
00060 /** Control device.
00061 *
00062 * See [man 2 ioctl] (http://man7.org/linux/man-pages/man2/ioctl.2.html) for
00063 * documentation.
00064 *
00065 * \param fd The file descriptor to control.
00066 * \param request Request code, defined in <sys/ioctl.h>
00067 * \param argp Pointer to input/output, dependent on request code.
00068 */
00069 int UnixSyscalls::ioctl(int fd, unsigned long request, void *argp)
00070 {
00071     return ::ioctl(fd, request, argp);
00072 }
00073
00074
00075 /** Open and possibly create a file.
00076 *
00077 * See [man 2 open] (http://man7.org/linux/man-pages/man2/open.2.html) for
```

```
00078 * documentation.
00079 */
00080 int UnixSyscalls::open(const char *pathname, int flags)
00081 {
00082     return ::open(pathname, flags);
00083 }
00084
00085
00086 /** Wait for some event on a file descriptor.
00087 *
00088 * See [man 2 poll] (http://man7.org/linux/man-pages/man2/poll.2.html) for
00089 * documentation.
00090 *
00091 * \param fds File descriptor event structures.
00092 * \param nfds Number of file descriptor event structures.
00093 * \param timeout The timeout in milliseconds.
00094 */
00095 int UnixSyscalls::poll(struct pollfd *fds, nfds_t nfds, int timeout)
00096 {
00097     return ::poll(fds, nfds, timeout);
00098 }
00099
00100
00101 /** Read from a file descriptor.
00102 *
00103 * See [man 2 read] (http://man7.org/linux/man-pages/man2/read.2.html) for
00104 * documentation.
00105 */
00106 ssize_t UnixSyscalls::read(int fd, void *buf, size_t count)
00107 {
00108     return ::read(fd, buf, count);
00109 }
00110
00111
00112 /** Receive a message from a socket.
00113 *
00114 * See [man 2 recvfrom] (http://man7.org/linux/man-pages/man2/recv.2.html) for
00115 * documentation.
00116 *
00117 * \param sockfd Socket file descriptor to receive data on.
00118 * \param buf The buffer to write the data into.
00119 * \param len The length of the buffer.
00120 * \param flags Option flags.
00121 * \param src_addr Source address buffer.
00122 * \param addrlen Before call, length of source address buffer. After call,
00123 * actual length of address data.
00124 * \returns The number of bytes written to \p or -1 if an error occurred.
00125 */
00126 ssize_t UnixSyscalls::recvfrom(
00127     int sockfd, void *buf, size_t len, int flags,
00128     struct sockaddr *src_addr, socklen_t *addrlen)
00129 {
00130     return ::recvfrom(sockfd, buf, len, flags, src_addr, addrlen);
00131 }
00132
00133
00134 /** Send a message on a socket.
00135 *
00136 * See [man 2 sendto] (http://man7.org/linux/man-pages/man2/send.2.html) for
00137 * documentation.
00138 */
00139 ssize_t UnixSyscalls::sendto(
00140     int sockfd, const void *buf, size_t len, int flags,
00141     const struct sockaddr *dest_addr, socklen_t addrlen)
00142 {
00143     return ::sendto(sockfd, buf, len, flags, dest_addr, addrlen);
00144 }
00145
00146
00147 /** Create an endpoint for communication.
00148 *
00149 * See [man 2 socket] (http://man7.org/linux/man-pages/man2/socket.2.html) for
00150 * documentation.
00151 *
00152 * \param domain Protocol family to use for communication.
00153 * \param type Socket type.
00154 * \param protocol Protocol to use for socket.
00155 */
00156 int UnixSyscalls::socket(int domain, int type, int protocol)
00157 {
00158     return ::socket(domain, type, protocol);
```

```

00159 }
00160
00161
00162 /** Get serial port parameters associated with the given file descriptor.
00163 *
00164 * See [man 2 termios] (http://man7.org/linux/man-pages/man3/termios.3.html) for
00165 * documentation.
00166 */
00167 int UnixSyscalls::tcgetattr(int fd, struct termios *termios_p)
00168 {
00169     return ::tcgetattr(fd, termios_p);
00170 }
00171
00172
00173 /** Set serial port parameters for given file descriptor.
00174 *
00175 * See [man 2 termios] (http://man7.org/linux/man-pages/man3/termios.3.html) for
00176 * documentation.
00177 */
00178 int UnixSyscalls::tcsetattr(
00179     int fd, int optional_actions, const struct termios *termios_p)
00180 {
00181     return ::tcsetattr(fd, optional_actions, termios_p);
00182 }
00183
00184
00185 /** Write to a file descriptor.
00186 *
00187 * See [man 2 write] (http://man7.org/linux/man-pages/man2/write.2.html) for
00188 * documentation.
00189 */
00190 ssize_t UnixSyscalls::write(int fd, const void *buf, size_t count)
00191 {
00192     return ::write(fd, buf, count);
00193 }

```

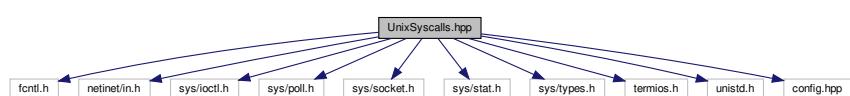
16.291 UnixSyscalls.hpp File Reference

```

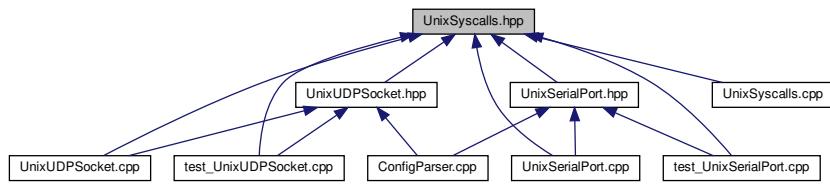
#include <fcntl.h>
#include <netinet/in.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <termios.h>
#include <unistd.h>
#include "config.hpp"

```

Include dependency graph for UnixSyscalls.hpp:



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



Classes

- class [UnixSyscalls](#)

16.292 UnixSyscalls.hpp

```

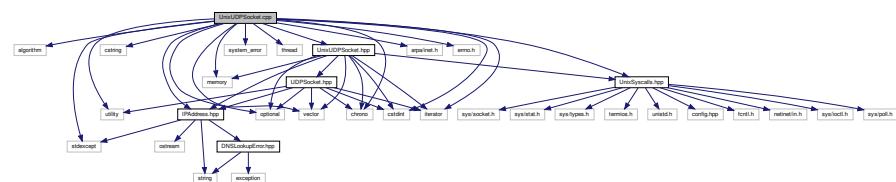
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef UNIXSYSCALLS_HPP_
00019 #define UNIXSYSCALLS_HPP_
00020
00021
00022 #include <fcntl.h>      // open, fcntl
00023 #include <netinet/in.h> // sockaddr_in
00024 #include <sys/ioctl.h> // ioctl
00025 #include <sys/poll.h>   // poll
00026 #include <sys/socket.h> // socket, bind, sendto, recvfrom
00027 #include <sys/stat.h>   // fstat
00028 #include <sys/types.h>  // socklen_t type on old BSD systems
00029 #include <termios.h>    // terminal control
00030 #include <unistd.h>    // read, write, close
00031
00032 #include "config.hpp"
00033
00034
00035 /** A thin wrapper around Unix system calls.
00036 *
00037 * The purpose of this is to allow system calls to be mocked during testing.
00038 *
00039 * See the following man pages for documentation:
00040 * * [man 2 bind](http://man7.org/linux/man-pages/man2/bind.2.html)
00041 * * [man 2 close](http://man7.org/linux/man-pages/man2/close.2.html)
00042 * * [man 2 socket](http://man7.org/linux/man-pages/man2/socket.2.html)
00043 * * [man 2 ioctl](http://man7.org/linux/man-pages/man2/ioctl.2.html)
00044 * * [man 7 ip](http://man7.org/linux/man-pages/man7/ip.7.html)
00045 * * [man 2 open](http://man7.org/linux/man-pages/man2/open.2.html)
00046 * * [man 2 poll](http://man7.org/linux/man-pages/man2/poll.2.html)
00047 * * [man 2 read](http://man7.org/linux/man-pages/man2/read.2.html)
  
```

```
00048 *   * [man 2 recvfrom] (http://man7.org/linux/man-pages/man2/recv.2.html)
00049 *   * [man 2 sendto] (http://man7.org/linux/man-pages/man2/send.2.html)
00050 *   * [man 2 termios] (http://man7.org/linux/man-pages/man3/termios.3.html)
00051 *   * [man 2 write] (http://man7.org/linux/man-pages/man2/write.2.html)
00052 *
00053 */
00054 class UnixSyscalls
00055 {
00056     public:
00057         TEST_VIRTUAL ~UnixSyscalls() = default;
00058         TEST_VIRTUAL int bind(
00059             int sockfd, const struct sockaddr *addr, socklen_t addrlen);
00060         TEST_VIRTUAL int close(int fd);
00061         TEST_VIRTUAL int ioctl(int fd, unsigned long request, void *argp);
00062         TEST_VIRTUAL int open(const char *pathname, int flags);
00063         TEST_VIRTUAL int poll(struct pollfd *fds, nfds_t nfds, int timeout);
00064         TEST_VIRTUAL ssize_t read(int fd, void *buf, size_t count);
00065         TEST_VIRTUAL ssize_t recvfrom(
00066             int sockfd, void *buf, size_t len, int flags,
00067             struct sockaddr *src_addr, socklen_t *addrlen);
00068         TEST_VIRTUAL ssize_t sendto(
00069             int sockfd, const void *buf, size_t len, int flags,
00070             const struct sockaddr *dest_addr, socklen_t addrlen);
00071         TEST_VIRTUAL int socket(int domain, int type, int protocol);
00072         TEST_VIRTUAL int tcgetattr(int fd, struct termios *termios_p);
00073         TEST_VIRTUAL int tcsetattr(
00074             int fd, int optional_actions, const struct termios *termios_p);
00075         TEST_VIRTUAL ssize_t write(int fd, const void *buf, size_t count);
00076    };
00077
00078
00079 #endif // UNIXSYSCALLS_HPP_
```

16.293 UnixUDPSocket.cpp File Reference

```
#include <algorithm>
#include <chrono>
#include <cstdint>
#include <cstring>
#include <iterator>
#include <memory>
#include <optional>
#include <stdexcept>
#include <system_error>
#include <thread>
#include <utility>
#include <vector>
#include <arpa/inet.h>
#include <errno.h>
#include "IPAddress.hpp"
#include "UnixSyscalls.hpp"
#include "UnixUDPSocket.hpp"
```

Include dependency graph for UnixUDPSocket.cpp:



16.294 UnixUDPSocket.cpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <algorithm>
00019 #include <chrono>
00020 #include <cstdint>
00021 #include <cstring>
00022 #include <iterator>
00023 #include <memory>
00024 #include <optional>
00025 #include <stdexcept>
00026 #include <system_error>
00027 #include <thread>
00028 #include <utility>
00029 #include <vector>
00030
00031 #include <arpa/inet.h>
00032 #include <errno.h>
00033
00034 #include "IPAddress.hpp"
00035 #include "UnixSyscalls.hpp"
00036 #include "UnixUDPSocket.hpp"
00037
00038 using namespace std::chrono_literals;
00039
00040
00041 /**
00042  * Construct a UDP socket.
00043  *
00044  * \param port The port number to listen on.
00045  * \param address The address to listen on (the port portion of the address is
00046  *     ignored). The default is to listen on any address.
00047  * \param max_bitrate The maximum number of bits per second to transmit on the
00048  *     UDP interface. The default is 0, which indicates no limit.
00049  * \param syscalls The object to use for unix system calls. It is default
00050  *     constructed to the production implementation. This argument is only
00051  *     used for testing.
00052  * \throws std::system_error if a system call produces an error.
00053 */
00054 UnixUDPSocket::UnixUDPSocket(
00055     unsigned int port, std::optional<IPAddress> address,
00056     unsigned long max_bitrate, std::unique_ptr<UnixSyscalls> syscalls)
00057 : port_(port), address_(std::move(address)), max_bitrate_(max_bitrate),
00058   syscalls_(std::move(syscalls)), socket_(-1),
00059   next_time_(std::chrono::steady_clock::now())
00060 {
00061     create_socket_();
00062 }
00063
00064
00065 /**
00066  * Closes the underlying file descriptor of the UDP socket.
00067 */
00068 // LCOV_EXCL_START
00069 UnixUDPSocket::~UnixUDPSocket()
00070 {
00071     syscalls_->close(socket_);
00072 }
00073
00074 // LCOV_EXCL_STOP
00075
00076
00077 /**
00078  * \copydoc UDPSocket::send(const std::vector<uint8_t> &, const IPAddress &)
```

```

00078  *
00079  * \throws PartialSendError if it fails to write all the data it is given.
00080  */
00081 void UnixUDPSocket::send(
00082     const std::vector<uint8_t> &data, const IPAddress &address)
00083 {
00084     if (max_bitrate_ != 0)
00085     {
00086         // Implement rate limit.
00087         auto now = std::chrono::steady_clock::now();
00088
00089         if (now < next_time_)
00090         {
00091             std::this_thread::sleep_for(next_time_ - now);
00092         }
00093
00094         next_time_ = now + (1000 * 1000 * data.size() * 8) / max_bitrate_ * lus;
00095     }
00096
00097     // Destination address structure.
00098     struct sockaddr_in addr;
00099     addr.sin_family = AF_INET;
00100    addr.sin_port = htons(static_cast<uint16_t>(address.port()));
00101    addr.sin_addr.s_addr =
00102        htonl(static_cast<uint32_t>(address.address()));
00103    std::memset(addr.sin_zero, '\0', sizeof(addr.sin_zero));
00104
00105    // Send the packet.
00106    auto err = syscalls_->sendto(
00107        socket_, data.data(), data.size(), 0,
00108        reinterpret_cast<struct sockaddr *>(&addr), sizeof(addr));
00109
00110    if (err < 0)
00111    {
00112        throw std::system_error(std::error_code(errno, std::system_category()));
00113    }
00114
00115
00116 /** \copydoc UDPSocket::receive(const std::chrono::nanoseconds &)
00117 *
00118 * \note The timeout precision of this implementation is 1 millisecond.
00119 *
00120 * \throws std::system_error if a system call produces an error.
00121 */
00122 std::pair<std::vector<uint8_t>, IPAddress> UnixUDPSocket::receive(
00123     const std::chrono::nanoseconds &timeout)
00124 {
00125     std::chrono::milliseconds timeout_ms =
00126         std::chrono::duration_cast<std::chrono::milliseconds>(timeout);
00127     struct pollfd fds = {socket_, POLLIN, 0};
00128     auto result = syscalls_->poll(
00129         &fds, 1, static_cast<int>(timeout_ms.count()));
00130
00131     // Poll error
00132     if (result < 0)
00133     {
00134         throw std::system_error(std::error_code(errno, std::system_category()));
00135     }
00136     // Success
00137     else if (result > 0)
00138     {
00139         // Socket error
00140         if (fds.revents & POLLERR)
00141         {
00142             syscalls_->close(socket_);
00143             create_socket_();
00144             return {std::vector<uint8_t>(), IPAddress(0)};
00145         }
00146         // Datagram available for reading.
00147         else if (fds.revents & POLLIN)
00148         {
00149             return receive_();
00150         }
00151     }
00152
00153     // Timed out
00154     return {std::vector<uint8_t>(), IPAddress(0)};
00155 }
00156
00157
00158 /** Create socket using the 'port_' and 'address_' member variables.

```

```
00159  * \throws std::system_error if a system call produces an error.
00160  */
00162 void UnixUDPSocket::create_socket_()
00163 {
00164     socket_ = -1;
00165
00166     // Create socket.
00167     if ((socket_ = syscalls_->socket(AF_INET, SOCK_DGRAM, 0)) < 0)
00168     {
00169         throw std::system_error(std::error_code(errno, std::system_category()));
00170     }
00171
00172     // Bind socket to port (and optionally an IP address).
00173     struct sockaddr_in addr;
00174     addr.sin_family = AF_INET;
00175     addr.sin_port = htons(static_cast<uint16_t>(port_));
00176
00177     if (address_)
00178     {
00179         addr.sin_addr.s_addr =
00180             htonl(static_cast<uint32_t>(address_.value().address()));
00181     }
00182     else
00183     {
00184         addr.sin_addr.s_addr = htonl(INADDR_ANY);
00185     }
00186
00187     std::memset(&addr.sin_zero, '\0', sizeof(addr.sin_zero));
00188
00189     if ((syscalls_->bind(socket_, reinterpret_cast<struct sockaddr *>(&addr),
00190                           sizeof(addr))) < 0)
00191     {
00192         throw std::system_error(std::error_code(errno, std::system_category()));
00193     }
00194 }
00195
00196
00197 /** Read data from socket.
00198 */
00199 * \note There must be a packet to receive, otherwise calling this method is
00200 *       undefined.
00201 *
00202 * \returns The data read from the socket and the IP address it was sent from.
00203 * \throws std::system_error if a system call produces an error.
00204 */
00205 std::pair<std::vector<uint8_t>, IPAddress> UnixUDPSocket::receive_()
00206 {
00207     // Get needed buffer size.
00208     int packet_size;
00209
00210     if ((syscalls_->ioctl(socket_, FIONREAD, &packet_size)) < 0)
00211     {
00212         throw std::system_error(std::error_code(errno, std::system_category()));
00213     }
00214
00215     // Read datagram.
00216     std::vector<uint8_t> buffer;
00217     buffer.resize(static_cast<size_t>(packet_size));
00218     struct sockaddr_in addr;
00219     socklen_t addrlen = sizeof(addr);
00220     auto size = syscalls_->recvfrom(
00221         socket_, buffer.data(), buffer.size(), 0,
00222         reinterpret_cast<struct sockaddr *>(&addr), &addrlen);
00223
00224     // Handle errors and extract IP address.
00225     if (size < 0)
00226     {
00227         throw std::system_error(std::error_code(errno, std::system_category()));
00228     }
00229     else if (size > 0)
00230     {
00231         if (addrlen <= sizeof(addr) && addr.sin_family == AF_INET)
00232         {
00233             auto ip =
00234                 IPAddress(ntohl(addr.sin_addr.s_addr), ntohs(addr.sin_port));
00235             return {buffer, ip};
00236         }
00237     }
00238
00239     // Failed to read datagram.
```

```

00240     return {std::vector<uint8_t>(), IPAddress(0)};
00241 }
00242
00243
00244 /** \copydoc UDPSocket::print_(std::ostream &os) const
00245 *
00246 * An example:
00247 * ``
00248 * udp {
00249 *     port 14555;
00250 *     address 127.0.0.1;
00251 *     max_bitrate 262144;
00252 * }
00253 * ``
00254 *
00255 * \param os The output stream to print to.
00256 */
00257 std::ostream &UnixUDPSocket::print_(std::ostream &os) const
00258 {
00259     os << "udp {" << std::endl;
00260     os << "    port " << std::to_string(port_) << ";" << std::endl;
00261
00262     if (address_.has_value())
00263     {
00264         os << "    address " << address_.value() << ";" << std::endl;
00265     }
00266
00267     if (max_bitrate_ != 0)
00268     {
00269         os << "    max_bitrate " << max_bitrate_ << ";" << std::endl;
00270     }
00271
00272     os << "}";
00273     return os;
00274 }

```

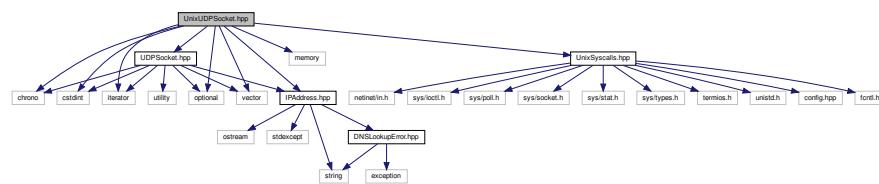
16.295 UnixUDPSocket.hpp File Reference

```

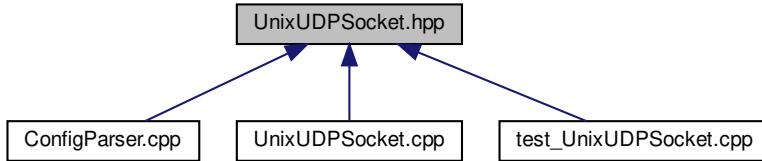
#include <chrono>
#include <cstdint>
#include <iterator>
#include <memory>
#include <optional>
#include <vector>
#include "IPAddress.hpp"
#include "UDPSocket.hpp"
#include "UnixSyscalls.hpp"

Include dependency graph for UnixUDPSocket.hpp:

```



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



Classes

- class [UnixUDPSocket](#)

16.296 UnixUDPSocket.hpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef UNIXUDPSOCKET_HPP_
00019 #define UNIXUDPSOCKET_HPP_
00020
00021
00022 #include <chrono>
00023 #include <cstdint>
00024 #include <iterator>
00025 #include <memory>
00026 #include <optional>
00027 #include <vector>
00028
00029 #include "IPAddress.hpp"
00030 #include "UDPSocket.hpp"
00031 #include "UnixSyscalls.hpp"
00032
00033
00034 /** A unix UDP socket, listening on a port/address combination.
00035 */
00036 class UnixUDPSocket : public UDPSocket
00037 {
00038     public:
00039         UnixUDPSocket(
00040             unsigned int port, std::optional<IPAddress> address = {},
00041             unsigned long max_bitrate = 0,
00042             std::unique_ptr<UnixSyscalls> syscalls =
00043                 std::make_unique<UnixSyscalls>());
00044         virtual ~UnixUDPSocket();
00045         virtual void send(
  
```

```

00046     const std::vector<uint8_t> &data, const IPAddress &address) final;
00047     virtual std::pair<std::vector<uint8_t>, IPAddress> receive(
00048         const std::chrono::nanoseconds &timeout =
00049             std::chrono::nanoseconds::zero()) final;
00050
00051     protected:
00052         std::ostream &print_(std::ostream &os) const final;
00053
00054     private:
00055         // Variables
00056         unsigned int port_;
00057         std::optional<IPAddress> address_;
00058         unsigned long max_bitrate_;
00059         std::unique_ptr<UnixSyscalls> syscalls_;
00060         int socket_;
00061         std::chrono::time_point<std::chrono::steady_clock> next_time_;
00062         // Methods
00063         void create_socket_();
00064         std::pair<std::vector<uint8_t>, IPAddress> receive_();
00065     };
00066
00067
00068 #endif // UNIXUDPSOCKET_HPP_

```

16.297 user_manual.md File Reference

16.298 user_manual.md

```

00001 MAVTables User Manual {#user_manual}
00002 =====
00003
00004 * [Project] (https://github.com/shamuproject/mavtables)
00005 * [Installing] (#installing)
00006   * [Requirements] (#requirements)
00007   * [CMake Installation] (#cmake-installation)
00008   * [Manual Installation] (#manual-installation)
00009 * [Usage] (#usage)
00010   * [Running] (#running)
00011   * [Log Level] (#log-level)
00012     * [--loglevel 0] (#--loglevel-0)
00013     * [--loglevel 1] (#--loglevel-1)
00014     * [--loglevel 2] (#--loglevel-2)
00015     * [--loglevel 3] (#--loglevel-3)
00016   * [Abstract Syntax Tree] (#abstract-syntax-tree)
00017 * [Configuration] (configuration.md)
00018
00019
00020 # Installing
00021
00022 ## Requirements
00023
00024 In order to compile you will need the following packages:
00025
00026 * GCC 7+ or Clang 5+ (needed for C++17 support)
00027 * [CMake v3.3+] (https://cmake.org/)
00028 * [Boost v1.54+] (https://www.boost.org/)
00029
00030 on either
00031
00032 * Linux
00033 * Mac OS X
00034 * BSD
00035
00036 mavtables is tested on both Linux and Mac OS X and should work on any unix
00037 compatible system.
00038
00039
00040 ## CMake Installation
00041
00042 To install via CMake simple run:
00043 ``
00044 $ make

```

```
00045 $ make install
00046 ``
00047
00048 The installation prefix is '/usr/local' by default but can be changed with
00049 ``
00050 $ make PREFIX=/desired/install/path
00051 # make DESDIR=/desired/install/path install
00052 ``
00053 'PREFIX' changes where mavtables expects to be installed while 'DESTDIR' changes
00054 where it will actually be installed.
00055
00056 To change the MAVLink dialect use the 'DIALECT' environment variable. For
00057 example, to use the 'common' dialect:
00058 ``
00059 $ make DIALECT=common
00060 # make install
00061 ``
00062
00063 To change the MAVLink implementation completely use the 'MDIR' environment
00064 variable to set the path to the MAVLink library instead of downloading the
00065 upstream implementation. For example, if a custom implementation is at
00066 '/tmp/mavlink/v2' then:
00067 ``
00068 $ make MDIR=/tmp/mavlink/v2
00069 # make install
00070 ``
00071 Both the 'MDIR' and 'DIALECT' variables can be used together.
00072
00073
00074 ## Manual Installation
00075
00076 To manually install mavtables first make a release build:
00077 ``
00078 $ make
00079 ``
00080 Remembering to use the 'PREFIX' option if the final installation destination is
00081 not '/usr/local'.
00082
00083 Copy binary, configuration, and systemd unit files to their proper locations
00084 ``
00085 # cp build/mavtables /usr/local/bin/
00086 # cp examples/mavtables.conf /usr/local/etc/
00087 # cp build/mavtables.service /usr/local/lib/systemd/system/
00088 ``
00089
00090
00091 # Usage
00092
00093 ## Running
00094
00095 To run mavtables and begin routing packets
00096 ``
00097 $ mavtables
00098 ``
00099 This will use the first configuration file it finds in the configuration file
00100 priority order:
00101
00102 1. The target of the 'MAVTABLES_CONFIG_PATH' environment variable.
00103 2. '.mavtablesrc' in the current directory.
00104 3. '.mavtablesrc' at '$HOME/.mavtablesrc'.
00105 4. The main configuration file at '/etc/mavtables.conf'.
00106
00107 To specify the configuration file use
00108 ``
00109 $ mavtables --config path/to/config/file
00110 ``
00111
00112 If the configuration file contains an error, mavtables will exit immediately and
00113 print an error message to stderr.
00114
00115 The installer will install a system wide configuration file at
00116 'PREFIX/etc/mavtables.conf' and a systemd unit file at
00117 'PREFIX/lib/systemd/system/mavtables.service'. Therefore, on Linux (with
00118 systemd) mavtables can be run as a daemon with
00119 ``
00120 # systemctl start mavtables
00121 ``
00122 or enabled for startup on each boot with
00123 ``
00124 # systemctl enable mavtables
00125 ``
```

```

00126
00127 systemd will keep mavtables running (even if it crashes) and will use the
00128 'PREFIX/etc/mavtables.conf' configuration file. It will also use 'loglevel' 1,
00129 discussed in the [Log Level](#log-level) section below.
00130
00131
00132 ## Log Level
00133
00134 The '--loglevel' flag can be used to set the logging level. This is the
00135 verbosity to print to stdout when mavtables is running. Each loglevel,
00136 0 through 3, is documented below.
00137
00138 ### --loglevel 0
00139
00140 Do not log anything to stdout.
00141
00142 ### --loglevel 1
00143
00144 Log each new component. Therefore, every time a new system/component address is
00145 connected to the router it will be printed to stdout. An example log output
00146 is:
00147 ``
00148 new component 127.1 on 127.0.0.1
00149 new component 192.168 on 127.0.0.1
00150 new component 172.128 on ./ttyS0
00151 new component 10.10 on 127.0.0.1
00152 ``
00153
00154 ### --loglevel 2
00155
00156 In addition to everything in '--loglevel 1' this will log received packets to
00157 stdout. An example log output is:
00158 ``
00159 new component 127.1 on 127.0.0.1
00160 received HEARTBEAT (#0) from 127.1 (v2.0) source 127.0.0.1
00161 new component 192.168 on 127.0.0.1
00162 received HEARTBEAT (#0) from 192.168 (v2.0) source 127.0.0.1
00163 new component 172.128 on ./ttyS0
00164 received HEARTBEAT (#0) from 172.128 (v2.0) source ./ttyS0
00165 new component 10.10 on 127.0.0.1
00166 received HEARTBEAT (#0) from 10.10 (v2.0) source 127.0.0.1
00167 received POWER_STATUS (#125) from 10.10 (v2.0) source 127.0.0.1
00168 received SYS_STATUS (#1) from 10.10 (v2.0) source 127.0.0.1
00169 received GPS_RAW_INT (#24) from 10.10 (v2.0) source 127.0.0.1
00170 received LOCAL_POSITION_NED_COV (#64) from 10.10 (v2.0) source 127.0.0.1
00171 received MISSION_REQUEST_PARTIAL_LIST (#37) from 10.10 to 172.168 (v2.0) source 127.0.0.1
00172 received LOCAL_POSITION_NED_SYSTEM_GLOBAL_OFFSET (#89) from 10.10 (v2.0) source 127.0.0.1
00173 received MANUAL_CONTROL (#69) from 10.10 (v2.0) source 127.0.0.1
00174 received GPS_INPUT (#232) from 10.10 (v2.0) source 127.0.0.1
00175 received VFR_HUD (#74) from 10.10 (v2.0) source 127.0.0.1
00176 received LANDING_TARGET (#149) from 10.10 (v2.0) source 127.0.0.1
00177 received SCALED_IMU (#26) from 10.10 (v2.0) source 127.0.0.1
00178 received PARAM_REQUEST_READ (#20) from 10.10 to 192.168 (v2.0) source 127.0.0.1
00179 ``
00180
00181 ### --loglevel 3
00182
00183 In addition to everything in '--loglevel 1' and '--loglevel 2' this will log
00184 routed packets, indicating whether they are 'accepted' or 'rejected', to
00185 stdout. An example log output is:
00186 ``
00187 new component 127.1 on 127.0.0.1
00188 received HEARTBEAT (#0) from 127.1 (v2.0) source 127.0.0.1
00189 rejected HEARTBEAT (#0) from 127.1 (v2.0) source 127.0.0.1 dest ./ttyS0
00190 new component 192.168 on 127.0.0.1
00191 received HEARTBEAT (#0) from 192.168 (v2.0) source 127.0.0.1
00192 rejected HEARTBEAT (#0) from 192.168 (v2.0) source 127.0.0.1 dest ./ttyS0
00193 rejected HEARTBEAT (#0) from 192.168 (v2.0) source 127.0.0.1 dest 127.0.0.1
00194 new component 172.128 on ./ttyS0
00195 received HEARTBEAT (#0) from 172.128 (v2.0) source ./ttyS0
00196 rejected HEARTBEAT (#0) from 172.128 (v2.0) source ./ttyS0 dest 127.0.0.1
00197 rejected HEARTBEAT (#0) from 172.128 (v2.0) source ./ttyS0 dest 127.0.0.1
00198 new component 10.10 on 127.0.0.1
00199 received HEARTBEAT (#0) from 10.10 (v2.0) source 127.0.0.1
00200 rejected HEARTBEAT (#0) from 10.10 (v2.0) source 127.0.0.1 dest ./ttyS0
00201 rejected HEARTBEAT (#0) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
00202 rejected HEARTBEAT (#0) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
00203 received POWER_STATUS (#125) from 10.10 (v2.0) source 127.0.0.1
00204 rejected POWER_STATUS (#125) from 10.10 (v2.0) source 127.0.0.1 dest ./ttyS0
00205 accepted POWER_STATUS (#125) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
00206 rejected POWER_STATUS (#125) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1

```

```

00207 received SYS_STATUS (#1) from 10.10 (v2.0) source 127.0.0.1
00208 accepted SYS_STATUS (#1) from 10.10 (v2.0) source 127.0.0.1 dest ./ttyS0
00209 accepted SYS_STATUS (#1) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
00210 accepted SYS_STATUS (#1) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
00211 received GPS_RAW_INT (#24) from 10.10 (v2.0) source 127.0.0.1
00212 rejected GPS_RAW_INT (#24) from 10.10 (v2.0) source 127.0.0.1 dest ./ttyS0
00213 accepted GPS_RAW_INT (#24) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
00214 accepted GPS_RAW_INT (#24) from 10.10 (v2.0) source 127.0.0.1 dest 127.0.0.1
00215 ``
00216
00217 ## Abstract Syntax Tree
00218
00219 mavtables can print the abstract syntax tree of the configuration file instead
00220 of routing packets. When run in this mode it will return immediately This is
00221 accomplished with:
00222 ``
00223 $ mavtables --ast
00224 ``
00225 or
00226 ``
00227 $ mavtables --ast --config path/to/config/file
00228 ``
00229 which will print something similar to
00230 ``
00231 ===== /etc/mavtables.conf =====
00232 :005: default_action
00233 :005: | reject
00234 :008: udp
00235 :009: | port 14500
00236 :010: | address 127.0.0.1
00237 :022: chain default
00238 :023: | accept
00239 ``
00240
00241 This feature can be used to debug configuration files.
00242
00243
00244 For and explanation of configuration files see
00245 [Configuration](configuration.md).

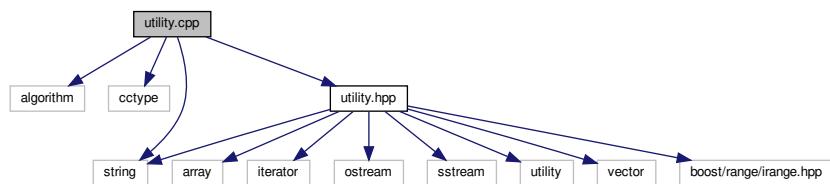
```

16.299 utility.cpp File Reference

```

#include <algorithm>
#include <cctype>
#include <string>
#include "utility.hpp"
Include dependency graph for utility.cpp:

```



Functions

- std::string [to_lower](#) (std::string string)

16.300 utility.cpp

```

00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #include <algorithm>
00019 #include <cctype>
00020 #include <string>
00021
00022 #include "utility.hpp"
00023
00024
00025 /** \defgroup utility Utility Functions
00026 *
00027 * Utility functions that don't warrant their own file.
00028 */
00029
00030
00031 /** Convert string to lower case.
00032 *
00033 * \ingroup utility
00034 * \param string The string to convert to lower case.
00035 * \returns The \a string converted to lower case.
00036 */
00037 std::string to_lower(std::string string)
00038 {
00039     std::transform(string.begin(), string.end(), string.begin(), ::tolower);
00040     return string;
00041 }

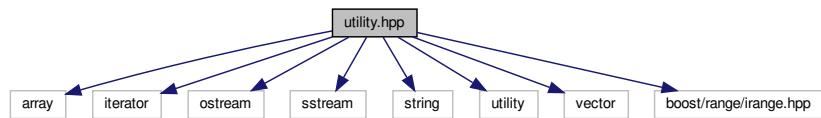
```

16.301 utility.hpp File Reference

```

#include <array>
#include <iterator>
#include <ostream>
#include <sstream>
#include <string>
#include <utility>
#include <vector>
#include <boost/range/irange.hpp>
Include dependency graph for utility.hpp:

```



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



Functions

- template<typename T >
std::vector< T >::iterator **append** (std::vector< T > &dest, const std::vector< T > &source)
- template<typename T >
std::vector< T >::iterator **append** (std::vector< T > &dest, std::vector< T > &&source)
- template<class T >
std::string **str** (const T &object)
- template<class ByteType = unsigned char, class T >
std::array< ByteType, sizeof(T)> **to_bytes** (T number)
- std::string **to_lower** (std::string string)
- template<class T >
std::ostream & **operator<<** (std::ostream &os, const std::vector< T > &vector)

16.302 utility.hpp

```
00001 // MAVLink router and firewall.
00002 // Copyright (C) 2017-2018 Michael R. Shannon <mrshannon.aerospace@gmail.com>
00003 //
00004 // This program is free software; you can redistribute it and/or modify
00005 // it under the terms of the GNU General Public License as published by
00006 // the Free Software Foundation; either version 2 of the License, or
00007 // (at your option) any later version.
00008 //
00009 // This program is distributed in the hope that it will be useful,
00010 // but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 // MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 // GNU General Public License for more details.
00013 //
00014 // You should have received a copy of the GNU General Public License
00015 // along with this program. If not, see <http://www.gnu.org/licenses/>.
00016
00017
00018 #ifndef UTILITY_HPP_
00019 #define UTILITY_HPP_
00020
00021
00022 #include <array>
00023 #include <iterator>
00024 #include <ostream>
00025 #include <sstream>
00026 #include <string>
00027 #include <utility>
00028 #include <vector>
00029
00030 #include <boost/range/irange.hpp>
00031
00032
00033 template <typename T>
00034 typename std::vector<T>::iterator append(
00035     std::vector<T> &dest, const std::vector<T> &source);
00036
00037 template <typename T>
00038 typename std::vector<T>::iterator append(
00039     std::vector<T> &dest, std::vector<T> &&source);
00040
00041 template <class T>
00042 std::string str(const T &object);
```

```

00044 template <class ByteType = unsigned char, class T>
00045 std::array<ByteType, sizeof(T)> to_bytes(T number);
00046
00047 std::string to_lower(std::string string);
00048
00049 template <class T>
00050 std::ostream &operator<<(std::ostream &os, const std::vector<T> &vector);
00051
00052
00053 /** Append one vector to another.
00054 *
00055 * Taken from https://stackoverflow.com/a/37210097
00056 *
00057 * \ingroup utility
00058 * \param dest Vector to append to.
00059 * \param source Vector to append the elements from.
00060 * \returns Iterator pointing to the first element appended, or the end of the
00061 * destination vector if the source vector is empty.
00062 */
00063 template <typename T>
00064 typename std::vector<T>::iterator append(
00065     std::vector<T> &dest, const std::vector<T> &source)
00066 {
00067     typename std::vector<T>::iterator result;
00068
00069     if (dest.empty())
00070     {
00071         dest = source;
00072         result = std::begin(dest);
00073     }
00074     else
00075     {
00076         result =
00077             dest.insert(std::end(dest), std::cbegin(source), std::cend(source));
00078     }
00079
00080     return result;
00081 }
00082
00083
00084 /** Append one vector to another (move from source).
00085 *
00086 * Taken from https://stackoverflow.com/a/37210097
00087 *
00088 * \ingroup utility
00089 * \param dest Vector to append to.
00090 * \param source Vector to append the elements from. \p source will be a valid
00091 * empty vector after this call.
00092 * \returns Iterator pointing to the first element appended, or the end of the
00093 * destination vector if the source vector is empty.
00094 */
00095 template <typename T>
00096 typename std::vector<T>::iterator append(
00097     std::vector<T> &dest, std::vector<T> &&source)
00098 {
00099     typename std::vector<T>::iterator result;
00100
00101     if (dest.empty())
00102     {
00103         dest = std::move(source);
00104         result = std::begin(dest);
00105     }
00106     else
00107     {
00108         result = dest.insert(
00109             std::end(dest),
00110             std::make_move_iterator(std::begin(source)),
00111             std::make_move_iterator(std::end(source)));
00112     }
00113
00114     source.clear();
00115     source.shrink_to_fit();
00116
00117     return result;
00118 }
00119
00120 /** Convert any object supporting the output stream operator (<<)
00121 * to a string.
00122 * \ingroup utility
00123 * \tparam T Type of the object to convert to a string.
00124 * \param object The object to convert to a string.

```

```
00125 * \returns The string representing the object.
00126 */
00127 template <class T>
00128 std::string str(const T &object)
00129 {
00130     std::ostringstream oss;
00131     oss << object;
00132     return oss.str();
00133 }
00134
00135
00136 /** Convert numeric types to bytes.
00137 *
00138 * \ingroup utility
00139 * \tparam ByteType Numeric type to return in the array of bytes.
00140 * \tparam T Type of the number being converted to bytes.
00141 * \param number Number to convert to bytes
00142 * \returns The array of bytes from the given number, in LSB order.
00143 */
00144 template <class ByteType, class T>
00145 std::array<ByteType, sizeof(T)> to_bytes(T number)
00146 {
00147     std::array<ByteType, sizeof(T)> n;
00148     n.fill(10);
00149
00150     for (auto i : boost::irange(static_cast<size_t>(0), sizeof(T)))
00151     {
00152         n[i] = (number >> i * 8) & 0xFF;
00153     }
00154
00155     return n;
00156 }
00157
00158
00159 /** Print a vector to the given output stream.
00160 *
00161 * \ingroup utility
00162 * \tparam T The type stored in the vector, it must support the << operator.
00163 * \param os The output stream to print to.
00164 * \param vector The vector of elements to print.
00165 * \returns The output stream.
00166 */
00167 template <class T>
00168 std::ostream &operator<<(std::ostream &os, const std::vector<T> &vector)
00169 {
00170     os << "[";
00171
00172     if (vector.size() > 0)
00173     {
00174         auto it = vector.begin();
00175         os << *(it++);
00176
00177         for (; it != vector.end(); ++it)
00178         {
00179             os << ", " << *it;
00180         }
00181     }
00182
00183     os << "]";
00184     return os;
00185 }
00186
00187
00188 #endif // UTILITY_HPP_
```


Index

~AddressPool
 AddressPool, 94
~Chain
 Chain, 108
~Connection
 Connection, 128
~ConnectionFactory
 ConnectionFactory, 134
~ConnectionPool
 ConnectionPool, 136
~Filesystem
 Filesystem, 142
~Filter
 Filter, 145
~Interface
 Interface, 167
~InterfaceThreader
 InterfaceThreader, 172
~MockCErr
 MockCErr, 218
~MockCOut
 MockCOut, 219
~Packet
 Packet, 230
~PacketQueue
 PacketQueue, 283
~RecursionGuard
 RecursionGuard, 303
~Rule
 Rule, 312
~SerialInterface
 SerialInterface, 323
~SerialPort
 SerialPort, 328
~UDPIInterface
 UDPIInterface, 336
~UDPSocket
 UDPSocket, 340
~UnixSerialPort
 UnixSerialPort, 348
~UnixSyscalls
 UnixSyscalls, 355
~UnixUDPSocket
 UnixUDPSocket, 362

Accept, 71

Accept, 73
action, 74
clone, 75
condition_, 77
operator!=, 75
operator==, 76
print_, 76
Accept.cpp, 373
Accept.hpp, 375, 376
Action, 78
 Action, 79, 81
 action, 81
 make_accept, 82
 make_continue, 83
 make_default, 84
 make_reject, 85
 operator!=, 89
 operator<<, 90
 operator=, 86, 87
 operator==, 91
 Option, 79
 priority, 87, 88
action
 Accept, 74
 Action, 81
 Call, 101
 Chain, 108
 GoTo, 151
 Reject, 306
 Rule, 312
 Action.cpp, 377, 378
 Action.hpp, 380, 382
 operator!=, 381
 operator<<, 381
 operator==, 382
action_a
 test_Action.cpp, 638
action_b
 test_Action.cpp, 638
add
 AddressPool, 94
 ConnectionPool, 136
add_address
 Connection, 128
address

IPAddress, 181
 MAVAddress, 199
 address_a
 test_IPAddress.cpp, 793
 test_MAVAddress.cpp, 804
 address_b
 test_IPAddress.cpp, 793
 test_MAVAddress.cpp, 804
 AddressPool
 ~AddressPool, 94
 add, 94
 AddressPool, 93
 addresses, 94
 contains, 95
 AddressPool< TC >, 92
 AddressPool.hpp, 383, 384
 addresses
 AddressPool, 94
 ansi_codes.sh, 386
 App, 96
 App, 96
 run, 97
 App.cpp, 387, 388
 App.hpp, 389, 390
 append
 Chain, 109
 test_Chain.cpp, 656
 Utility Functions, 52
 ast
 Options, 223
 bind
 UnixSyscalls, 355
 buffer
 MockCErr, 218
 MockCOut, 220
 bytes_parsed
 PacketParser, 279
 CATCH_CONFIG_MAIN
 unit_tests.cpp, 982
 CHECKSUM_LENGTH
 packet_v1, 66
 packet_v2, 68
 CONTRIBUTING.md, 453
 Call, 97
 action, 101
 Call, 100, 101
 clone, 102
 condition_, 105
 operator!=, 103
 operator==, 103
 print_, 104
 call
 test_Call.cpp, 648, 650
 Call.cpp, 390, 391
 Call.hpp, 393, 394
 Chain, 105
 ~Chain, 108
 action, 108
 append, 109
 Chain, 106, 107
 name, 110
 operator!=, 111
 operator<<, 112
 operator=, 110, 111
 operator==, 112
 Chain.cpp, 395, 396
 Chain.hpp, 399, 401
 operator!=, 400
 operator<<, 400
 operator==, 400
 chain_a
 test_Chain.cpp, 660
 chain_a_compare
 test_Chain.cpp, 660
 chain_b
 test_Chain.cpp, 660
 chain_b_compare
 test_Chain.cpp, 660
 check
 If, 158
 clear
 PacketParser, 280
 clone
 Accept, 75
 Call, 102
 GoTo, 152
 Reject, 307
 Rule, 313
 close
 PacketQueue, 284
 UnixSyscalls, 356
 common.hpp, 402, 404
 mock_shared, 402
 mock_unique, 403
 common_Packet.hpp, 405, 406
 common_Rule.hpp, 411
 compat_flags
 mavlink::v2_header, 370
 compid
 mavlink::v1_header, 368
 mavlink::v2_header, 370
 component
 MAVAddress, 200
 condition_
 Accept, 77
 Call, 105
 GoTo, 155

Reject, 309
Rule, 316
config, 57
 parse, 57
config_file
 Options, 223
config_grammar.cpp, 412, 413
config_grammar.hpp, 416, 417
ConfigParser, 113
 ConfigParser, 115, 116
 in_, 126
 init_chains, 117
 make_app, 116
 operator<<, 118
 operator=, 117
 parse_action, 119
 parse_chain, 120
 parse_condition, 121
 parse_filter, 122
 parse_interfaces, 123
 parse_serial, 124
 parse_udp, 125
 root_, 126
ConfigParser.cpp, 422, 423
ConfigParser.hpp, 429, 432
 init_chains, 430
 operator<<, 430
 parse_action, 431
 parse_chain, 431
 parse_condition, 431
 parse_filter, 431
 parse_interfaces, 431
 parse_serial, 431
 parse_udp, 432
Configuration functions., 45
 operator<<, 45
 print_node, 46
configuration.md, 433
Connection, 126
 ~Connection, 128
 add_address, 128
 Connection, 127
 next_packet, 129
 operator<<, 131
 send, 130
connection
 Packet, 230, 231
 packet_v1::Packet, 245, 246
 packet_v2::Packet, 263, 264
Connection.cpp, 439, 440
 operator<<, 440
Connection.hpp, 445, 446
 operator<<, 446
ConnectionFactory
 ~ConnectionFactory, 134
 ConnectionFactory, 133
 get, 134
 wait_for_packet, 134
ConnectionFactory< C, AP, PQ >, 132
ConnectionFactory.hpp, 447, 448
ConnectionPool, 135
 ~ConnectionPool, 136
 add, 136
 remove, 137
 send, 137
ConnectionPool.cpp, 450
ConnectionPool.hpp, 451, 452
contains
 AddressPool, 95
 MAVSubnet, 213
DNSLookupError, 139
 DNSLookupError, 140
 what, 140
DNSLookupError.cpp, 454, 455
DNSLookupError.hpp, 456
data
 Packet, 232
 packet_v1::Packet, 246
 packet_v2::Packet, 265
dest
 Packet, 233
 packet_v1::Packet, 248
 packet_v2::Packet, 266
dnslookup
 IPAddress, 183
 IPAddress.hpp, 496
empty
 PacketQueue, 284
exists
 Filesystem, 142
Feature
 SerialPort, 327
 UnixSerialPort, 347
Filesystem, 141
 ~Filesystem, 142
 exists, 142
 path, 141
Filesystem.cpp, 457
Filesystem.hpp, 458, 459
Filter, 143
 ~Filter, 145
 Filter, 144
 operator!=, 146
 operator=, 145
 operator==, 147
 will_accept, 146

Filter.cpp, 460
 Filter.hpp, 461, 463
 operator!=, 462
 operator==, 463
 filter_a
 test_Filter.cpp, 762
 filter_b
 test_Filter.cpp, 763
 find_config
 Options, 225
 Options.hpp, 539
 from
 If, 159

 get
 ConnectionFactory, 134
 GoTo, 148
 action, 151
 clone, 152
 condition_, 155
 GoTo, 150, 151
 operator!=, 153
 operator==, 153
 print_, 154
 GoTo.cpp, 464
 GoTo.hpp, 467, 468
 goto_
 test_GoTo.cpp, 766, 768

 HEADER_LENGTH
 packet_v1, 66
 packet_v2, 69
 header
 packet_v1, 64
 packet_v1::Packet, 252
 packet_v2, 67
 packet_v2::Packet, 270
 header_complete
 packet_v1, 65
 packet_v1::Packet, 253
 packet_v2, 67
 packet_v2::Packet, 271
 heartbeat
 logger, 58

 IPAddress, 176
 address, 181
 dnslookup, 183
 IPAddress, 177–180
 operator!=, 184
 operator<, 185
 operator<<, 186
 operator<=, 187
 operator>, 188
 operator>=, 189
 operator=, 182
 operator==, 187
 port, 183
 IPAddress.cpp, 487, 489
 IPAddress.hpp, 494, 498
 dnslookup, 496
 operator!=, 496
 operator<, 496
 operator<<, 496
 operator<=, 497
 operator>, 497
 operator>=, 498
 operator==, 497
 id
 MAVLink Library and Helpers, 49
 Packet, 234
 packet_v1::Packet, 249
 packet_v2::Packet, 267
 If, 155
 check, 158
 from, 159
 If, 156, 157
 operator!=, 164
 operator<<, 164
 operator=, 160
 operator==, 165
 to, 161
 type, 162, 163
 If.cpp, 469, 470
 operator<<, 469
 If.hpp, 473, 476
 operator!=, 474
 operator<<, 474
 operator==, 475
 if_a
 test_If.cpp, 776
 if_b
 test_If.cpp, 777
 in_
 ConfigParser, 126
 incompat_flags
 mavlink::v2_header, 370
 init_chains
 ConfigParser, 117
 ConfigParser.hpp, 430
 Interface, 166
 ~Interface, 167
 operator<<, 169
 print_, 167
 receive_packet, 168
 send_packet, 168
 Interface.cpp, 477
 Interface.hpp, 478, 479

operator<<, 479
InterfaceThreader, 170
 ~InterfaceThreader, 172
 InterfaceThreader, 171, 172
 operator=, 173
 shutdown, 173
 start, 173
 Threads, 171
InterfaceThreader.cpp, 480
InterfaceThreader.hpp, 482, 483
InvalidPacketIDError, 174
 InvalidPacketIDError, 175
 what, 176
InvalidPacketIDError.cpp, 484, 485
InvalidPacketIDError.hpp, 486
ioctl
 UnixSyscalls, 356
is_signed
 packet_v2, 68
 packet_v2::Packet, 272

LICENSE.md, 499
len
 mavlink::v1_header, 368
 mavlink::v2_header, 371
level
 Logger, 191
log
 Logger, 192, 193
Logger, 190
 level, 191
 log, 192, 193
logger, 58
 heartbeat, 58
 main, 59
 parse_args, 59
 signal_handler, 60
 start_heartbeats, 60
Logger.cpp, 503, 504
Logger.hpp, 505, 506
logger.py, 507
loglevel
 Options, 224

MAVAddress, 194
 address, 199
 component, 200
 MAVAddress, 195–198
 operator!=, 202
 operator<, 203
 operator<<, 204
 operator<=, 205
 operator>, 207
 operator>=, 207
 operator=, 201
 operator==, 206
 system, 201
MAVAddress.cpp, 512
MAVAddress.hpp, 516, 518
 operator!=, 517
 operator<, 517
 operator<<, 517
 operator<=, 517
 operator>, 518
 operator>=, 518
 operator==, 517
MAVLINK_USE_MESSAGE_INFO
 mavlink.hpp, 522
MAVLink Library and Helpers, 49
 id, 49
 name, 50
MAVSubnet, 208
 contains, 213
 MAVSubnet, 210, 211
 operator!=, 215
 operator<<, 215
 operator=, 214
 operator==, 216
MAVSubnet.cpp, 524
MAVSubnet.hpp, 529, 531
 operator!=, 530
 operator<<, 531
 operator==, 531
Macros, 48
 PACKED, 48
macros.hpp, 509
magic
 mavlink::v1_header, 368
 mavlink::v2_header, 371
main
 logger, 59
 mavtables.cpp, 533
 packet_scripter, 61
mainpage.md, 510
make_accept
 Action, 82
make_app
 ConfigParser, 116
make_continue
 Action, 83
make_default
 Action, 84
make_reject
 Action, 85
make_targets.md, 510
mavlink, 61
mavlink.cpp, 519, 520
mavlink.hpp, 521, 523
 MAVLINK_USE_MESSAGE_INFO, 522

mavlink::v1_header, 367
 compid, 368
 len, 368
 magic, 368
 msgid, 368
 seq, 368
 sysid, 369

mavlink::v2_header, 369
 compat_flags, 370
 compid, 370
 incompat_flags, 370
 len, 371
 magic, 371
 msgid, 371
 seq, 371
 sysid, 371

mavtables.cpp, 532, 534
 main, 533

mock_shared
 common.hpp, 402

mock_unique
 common.hpp, 403

MockCErr, 217
 ~MockCErr, 218
 buffer, 218
 MockCErr, 217
 reset, 218

MockCOut, 219
 ~MockCOut, 219
 buffer, 220
 MockCOut, 219
 reset, 220

msgid
 mavlink::v1_header, 368
 mavlink::v2_header, 371

name
 Chain, 110
 MAVLink Library and Helpers, 50
 Packet, 234
 packet_v1::Packet, 250
 packet_v2::Packet, 268

next_packet
 Connection, 129

notify
 semaphore, 318

open
 UnixSyscalls, 357

operator bool
 Options, 224

operator!=
 Accept, 75
 Action, 89
 Action.hpp, 381

Call, 103
 Chain, 111
 Chain.hpp, 400
 Filter, 146
 Filter.hpp, 462
 GoTo, 153
 IPAddress, 184
 IPAddress.hpp, 496
 If, 164
 If.hpp, 474
 MAVAddress, 202
 MAVAddress.hpp, 517
 MAVSubnet, 215
 MAVSubnet.hpp, 530
 Packet, 237
 Packet.hpp, 544
 packet_v1::Packet, 254
 packet_v2::Packet, 273
 QueuedPacket, 292
 QueuedPacket.hpp, 589
 Reject, 307
 Rule, 314

operator<
 IPAddress, 185
 IPAddress.hpp, 496
 MAVAddress, 203
 MAVAddress.hpp, 517
 QueuedPacket, 293
 QueuedPacket.hpp, 589

operator<<
 Action, 90
 Action.hpp, 381
 Chain, 112
 Chain.hpp, 400
 ConfigParser, 118
 ConfigParser.hpp, 430
 Configuration functions., 45
 Connection, 131
 Connection.cpp, 440
 Connection.hpp, 446
 IPAddress, 186
 IPAddress.cpp, 487
 IPAddress.hpp, 496
 If, 164
 If.cpp, 469
 If.hpp, 474
 Interface, 169
 Interface.hpp, 479
 MAVAddress, 204
 MAVAddress.hpp, 517
 MAVSubnet, 215
 MAVSubnet.hpp, 531
 Packet, 238
 Packet.hpp, 544

packet_v1::Packet, 255
packet_v2::Packet, 274
QueuedPacket, 293
QueuedPacket.hpp, 589
Rule, 315
Rule.hpp, 606
SerialPort, 332
SerialPort.hpp, 626
UDPSocket, 344
UDPSocket.hpp, 980
Utility Functions, 53
operator<= IPAddress, 187
IPAddress.hpp, 497
MAVAddress, 205
MAVAddress.hpp, 517
QueuedPacket, 294
QueuedPacket.hpp, 589
operator> IPAddress, 188
IPAddress.hpp, 497
MAVAddress, 207
MAVAddress.hpp, 518
QueuedPacket, 295
QueuedPacket.hpp, 589
operator>= IPAddress, 189
IPAddress.hpp, 498
MAVAddress, 207
MAVAddress.hpp, 518
QueuedPacket, 296
QueuedPacket.hpp, 590
operator= Action, 86, 87
Chain, 110, 111
ConfigParser, 117
Filter, 145
IPAddress, 182
If, 160
InterfaceThreader, 173
MAVAddress, 201
MAVSubnet, 214
Packet, 235
packet_v1::Packet, 251
packet_v2::Packet, 269
PacketParser, 280, 281
QueuedPacket, 291
RecursionData, 298
semaphore, 318, 319
operator== Accept, 76
Action, 91
Action.hpp, 382
Call, 103
Chain, 112
Chain.hpp, 400
Filter, 147
Filter.hpp, 463
GoTo, 153
IPAddress, 187
IPAddress.hpp, 497
If, 165
If.hpp, 475
MAVAddress, 206
MAVAddress.hpp, 517
MAVSubnet, 216
MAVSubnet.hpp, 531
Packet, 239
Packet.hpp, 544
packet_v1::Packet, 256
packet_v2::Packet, 275
QueuedPacket, 294
QueuedPacket.hpp, 589
Reject, 308
Rule, 314
Option Action, 79
Options, 221 ast, 223 config_file, 223 find_config, 225 loglevel, 224 operator bool, 224 Options, 222 run, 225 Options.cpp, 535 Options.hpp, 538, 540 find_config, 539
PACKED Macros, 48
Packet, 227 ~Packet, 230 connection, 230, 231 data, 232 dest, 233 id, 234 name, 234 operator!=, 237 operator<<, 238 operator=, 235 operator==, 239 Packet, 229, 230 packet_v1::Packet, 244 packet_v2::Packet, 262 source, 236 V1, 229 V2, 229

Version, 229
 version, 236
 packet
 QueuedPacket, 292
 Packet.cpp, 540, 541
 Packet.hpp, 543, 545
 operator!=, 544
 operator<<, 544
 operator==, 544
 packet_a
 test_Packet.cpp, 831
 test_PacketVersion1.cpp, 854
 test_PacketVersion2.cpp, 869
 packet_b
 test_Packet.cpp, 828
 test_PacketVersion1.cpp, 854
 test_PacketVersion2.cpp, 869
 packet_complete
 packet_v1, 65
 packet_v1::Packet, 257
 packet_v2, 68
 packet_v2::Packet, 276
 packet_scripter, 61
 main, 61
 parse_args, 61
 parse_file, 62
 parse_line, 62
 send_packet, 63
 start_connection, 63
 packet_scripter.py, 546, 547
 packet_v1, 64
 CHECKSUM_LENGTH, 66
 HEADER_LENGTH, 66
 header, 64
 header_complete, 65
 packet_complete, 65
 START_BYTE, 66
 VERSION, 66
 packet_v1::Packet, 240
 connection, 245, 246
 data, 246
 dest, 248
 header, 252
 header_complete, 253
 id, 249
 name, 250
 operator!=, 254
 operator<<, 255
 operator=, 251
 operator==, 256
 Packet, 244
 packet_complete, 257
 source, 251
 Version, 243
 version, 252
 packet_v2, 67
 CHECKSUM_LENGTH, 68
 HEADER_LENGTH, 69
 header, 67
 header_complete, 67
 is_signed, 68
 packet_complete, 68
 SIGNATURE_LENGTH, 69
 START_BYTE, 69
 VERSION, 69
 packet_v2::Packet, 258
 connection, 263, 264
 data, 265
 dest, 266
 header, 270
 header_complete, 271
 id, 267
 is_signed, 272
 name, 268
 operator!=, 273
 operator<<, 274
 operator=, 269
 operator==, 275
 Packet, 262
 packet_complete, 276
 source, 269
 Version, 261
 version, 270
 PacketParser, 277
 bytes_parsed, 279
 clear, 280
 operator=, 280, 281
 PacketParser, 278, 279
 parse_byte, 281
 PacketParser.cpp, 553
 PacketParser.hpp, 556, 557
 PacketQueue, 282
 ~PacketQueue, 283
 close, 284
 empty, 284
 PacketQueue, 283
 pop, 284, 285
 push, 286
 PacketQueue.cpp, 558
 PacketQueue.hpp, 561, 562
 PacketVersion1.cpp, 563, 564
 PacketVersion1.hpp, 567, 568
 PacketVersion2.cpp, 569, 570
 PacketVersion2.hpp, 574, 575
 Parity
 SerialPort, 328
 UnixSerialPort, 347
 parse

config, 57
parse_action
 ConfigParser, 119
 ConfigParser.hpp, 431
parse_args
 logger, 59
 packet_scripter, 61
parse_byte
 PacketParser, 281
parse_chain
 ConfigParser, 120
 ConfigParser.hpp, 431
parse_condition
 ConfigParser, 121
 ConfigParser.hpp, 431
parse_file
 packet_scripter, 62
parse_filter
 ConfigParser, 122
 ConfigParser.hpp, 431
parse_interfaces
 ConfigParser, 123
 ConfigParser.hpp, 431
parse_line
 packet_scripter, 62
parse_serial
 ConfigParser, 124
 ConfigParser.hpp, 431
parse_tree.hpp, 576, 577
parse_udp
 ConfigParser, 125
 ConfigParser.hpp, 432
PartialSendError, 286
 PartialSendError, 288
 what, 288
PartialSendError.cpp, 582
PartialSendError.hpp, 583, 584
path
 Filesystem, 141
ping
 test_Call.cpp, 650
 test_GoTo.cpp, 768
poll
 UnixSyscalls, 357
pop
 PacketQueue, 284, 285
port
 IPAddress, 183
print_
 Accept, 76
 Call, 104
 GoTo, 154
 Interface, 167
 Reject, 309
 Rule, 315
 SerialInterface, 323
 SerialPort, 328
 UDPIInterface, 336
 UDPSocket, 340
 UnixSerialPort, 349
 UnixUDPSocket, 363
print_node
 Configuration functions., 46
priority
 Action, 87, 88
push
 PacketQueue, 286
QueuedPacket, 289
 operator!=, 292
 operator<, 293
 operator<<, 293
 operator<=, 294
 operator>, 295
 operator>=, 296
 operator=, 291
 operator==, 294
 packet, 292
 QueuedPacket, 290
 QueuedPacket.cpp, 584, 585
 QueuedPacket.hpp, 588, 590
 operator!=, 589
 operator<, 589
 operator<<, 589
 operator<=, 589
 operator>, 589
 operator>=, 590
 operator==, 589
README.md, 591
read
 SerialPort, 329, 330
 UnixSerialPort, 349, 351
 UnixSyscalls, 357
receive
 UDPSocket, 341, 342
 UnixUDPSocket, 363, 364
receive_packet
 Interface, 168
 SerialInterface, 324
 UDPIInterface, 337
RecursionData, 296
 operator=, 298
 RecursionData, 297, 298
 RecursionGuard, 299
RecursionData.hpp, 593, 594
RecursionError, 299
 RecursionError, 300
 what, 301

RecursionError.cpp, 595
 RecursionError.hpp, 596
 RecursionGuard, 301
 ~RecursionGuard, 303
 RecursionData, 299
 RecursionGuard, 302
 RecursionGuard.cpp, 597, 598
 RecursionGuard.hpp, 598, 599
 recvfrom
 UnixSyscalls, 357
 Reject, 304
 action, 306
 clone, 307
 condition_, 309
 operator!=, 307
 operator==, 308
 print_, 309
 Reject, 306
 reject
 test_Reject.cpp, 892, 893
 Reject.cpp, 600, 601
 Reject.hpp, 602, 603
 remove
 ConnectionPool, 137
 reset
 MockCErr, 218
 MockCOut, 220
 root_
 ConfigParser, 126
 Rule, 310
 ~Rule, 312
 action, 312
 clone, 313
 condition_, 316
 operator!=, 314
 operator<<, 315
 operator==, 314
 print_, 315
 Rule, 312
 rule
 test_Call.cpp, 651
 test_GoTo.cpp, 769
 test_Reject.cpp, 893
 Rule.cpp, 604
 Rule.hpp, 605, 607
 operator<<, 606
 run
 App, 97
 Options, 225
 run_tests.sh, 608, 609
 SIGNATURE_LENGTH
 packet_v2, 69
 START_BYTE

packet_v1, 66
 packet_v2, 69
 semaphore, 317
 notify, 318
 operator=, 318, 319
 semaphore, 318
 wait, 319
 wait_for, 319
 wait_until, 319
 semaphore.cpp, 615, 616
 semaphore.hpp, 616, 617
 send
 Connection, 130
 ConnectionPool, 137
 UDPSocket, 343
 UnixUDPSocket, 365, 366
 send_packet
 Interface, 168
 packet_scripter, 63
 SerialInterface, 325
 UDPIInterface, 338
 sendto
 UnixSyscalls, 358
 seq
 mavlink::v1_header, 368
 mavlink::v2_header, 371
 SerialInterface, 320
 ~SerialInterface, 323
 print_, 323
 receive_packet, 324
 send_packet, 325
 SerialInterface, 323
 SerialInterface.cpp, 619
 SerialInterface.hpp, 621, 622
 SerialPort, 325
 ~SerialPort, 328
 Feature, 327
 operator<<, 332
 Parity, 328
 print_, 328
 read, 329, 330
 write, 331
 SerialPort.cpp, 623
 SerialPort.hpp, 625, 627
 operator<<, 626
 shutdown
 InterfaceThreader, 173
 signal_handler
 logger, 60
 socket
 UnixSyscalls, 358
 source
 Packet, 236
 packet_v1::Packet, 251

packet_v2::Packet, 269
start
 InterfaceThreader, 173
start_connection
 packet_scripter, 63
start_heartbeats
 logger, 60
str
 Utility Functions, 54
subnet_a
 test_MAVSubnet.cpp, 813
subnet_b
 test_MAVSubnet.cpp, 814
ssid
 mavlink::v1_header, 369
 mavlink::v2_header, 371
system
 MAVAddress, 201

TEST_CASE
 test_Accept.cpp, 628–630
 test_Action.cpp, 634–638
 test_AddressPool.cpp, 642
 test_Call.cpp, 648–650
 test_Chain.cpp, 656–659
 test_ConfigParser.cpp, 707–710
 test_Connection.cpp, 725–730
 test_ConnectionFactory.cpp, 747, 748
 test_ConnectionPool.cpp, 752, 753
 test_DNSLookupError.cpp, 757
 test_Filesystem.cpp, 759
 test_Filter.cpp, 761, 762
 test_GoTo.cpp, 766–768
 test_IPAddress.cpp, 790–792
 test_If.cpp, 774–776
 test_Interface.cpp, 782–784
 test_InterfaceThreader.cpp, 787
 test_Logger.cpp, 799
 test_MAVAddress.cpp, 802–804
 test_MAVSubnet.cpp, 811–813
 test_Options.cpp, 820, 821
 test_Packet.cpp, 828–831
 test_PacketParser.cpp, 835, 836
 test_PacketQueue.cpp, 840–842
 test_PacketVersion1.cpp, 847–853
 test_PacketVersion2.cpp, 862–868
 test_PartialSendError.cpp, 880
 test_QueuedPacket.cpp, 882–884
 test_RecursionError.cpp, 888
 test_RecursionGuard.cpp, 890
 test_Reject.cpp, 892, 893
 test_Rule.cpp, 896, 897
 test_Rule_comparison.cpp, 900
 test_SerialInterface.cpp, 908, 909
 test_SerialPort.cpp, 916–918
 test_UDPInterface.cpp, 921–925
 test_UDPSocket.cpp, 937–939
 test_UxSerialPort.cpp, 942–944
 test_UxUDPSocket.cpp, 956
 test_config_grammar.cpp, 665–671
 test_mavlink.cpp, 809
 test_semaphore.cpp, 902, 903
 test_utility.cpp, 967, 968
tcgetattr
 UnixSyscalls, 359
tcsetattr
 UnixSyscalls, 359
test_Accept.cpp, 628, 631
 TEST_CASE, 628–630
test_Action.cpp, 634, 639
 action_a, 638
 action_b, 638
 TEST_CASE, 634–638
test_AddressPool.cpp, 641, 643
 TEST_CASE, 642
test_Call.cpp, 647, 651
 call, 648, 650
 ping, 650
 rule, 651
 TEST_CASE, 648–650
test_Chain.cpp, 655, 661
 append, 656
 chain_a, 660
 chain_a_compare, 660
 chain_b, 660
 chain_b_compare, 660
 TEST_CASE, 656–659
test_ConfigParser.cpp, 706, 710
 TEST_CASE, 707–710
test_Connection.cpp, 724, 731
 TEST_CASE, 725–730
test_ConnectionFactory.cpp, 747, 749
 TEST_CASE, 747, 748
test_ConnectionPool.cpp, 751, 754
 TEST_CASE, 752, 753
test_DNSLookupError.cpp, 757, 758
 TEST_CASE, 757
test_Filesystem.cpp, 759, 760
 TEST_CASE, 759
test_Filter.cpp, 760, 763
 filter_a, 762
 filter_b, 763
 TEST_CASE, 761, 762
test_GoTo.cpp, 765, 769
 goto_, 766, 768
 ping, 768
 rule, 769
 TEST_CASE, 766–768

test_IPAddress.cpp, 789, 793
 address_a, 793
 address_b, 793
 TEST_CASE, 790–792
 test_If.cpp, 773, 777
 if_a, 776
 if_b, 777
 TEST_CASE, 774–776
 test_Interface.cpp, 782, 784
 TEST_CASE, 782–784
 test_InterfaceThreader.cpp, 786, 787
 TEST_CASE, 787
 test_Logger.cpp, 798, 800
 TEST_CASE, 799
 test_MAVAddress.cpp, 801, 805
 address_a, 804
 address_b, 804
 TEST_CASE, 802–804
 test_MAVSubnet.cpp, 810, 814
 subnet_a, 813
 subnet_b, 814
 TEST_CASE, 811–813
 test_Options.cpp, 819, 821
 TEST_CASE, 820, 821
 test_Packet.cpp, 827, 832
 packet_a, 831
 packet_b, 828
 TEST_CASE, 828–831
 test_PacketParser.cpp, 834, 836
 TEST_CASE, 835, 836
 test_PacketQueue.cpp, 839, 842
 TEST_CASE, 840–842
 test_PacketVersion1.cpp, 846, 855
 packet_a, 854
 packet_b, 854
 TEST_CASE, 847–853
 test_PacketVersion2.cpp, 861, 870
 packet_a, 869
 packet_b, 869
 TEST_CASE, 862–868
 test_PartialSendError.cpp, 879, 881
 TEST_CASE, 880
 test_QueuedPacket.cpp, 882, 884
 TEST_CASE, 882–884
 test_RecursionError.cpp, 887, 888
 TEST_CASE, 888
 test_RecursionGuard.cpp, 889, 890
 TEST_CASE, 890
 test_Reject.cpp, 891, 894
 reject, 892, 893
 rule, 893
 TEST_CASE, 892, 893
 test_Rule.cpp, 896, 898
 TEST_CASE, 896, 897
 test_Rule_comparison.cpp, 900, 901
 TEST_CASE, 900
 test_SerialInterface.cpp, 907, 910
 TEST_CASE, 908, 909
 test_SerialPort.cpp, 915, 918
 TEST_CASE, 916–918
 test_UDPInterface.cpp, 920, 926
 TEST_CASE, 921–925
 test_UDPSocket.cpp, 937, 940
 TEST_CASE, 937–939
 test_UinxSerialPort.cpp, 942, 944
 TEST_CASE, 942–944
 test_UinxUDPSocket.cpp, 955, 957
 TEST_CASE, 956
 test_config_grammar.cpp, 664, 672
 TEST_CASE, 665–671
 test_mavlink.cpp, 808, 809
 TEST_CASE, 809
 test_semaphore.cpp, 902, 904
 TEST_CASE, 902, 903
 test_utility.cpp, 966, 969
 TEST_CASE, 967, 968
 Threads
 InterfaceThreader, 171
 to
 If, 161
 to_bytes
 Utility Functions, 54
 to_lower
 Utility Functions, 55
 type
 If, 162, 163
 UDPInterface, 334
 ~UDPInterface, 336
 print_, 336
 receive_packet, 337
 send_packet, 338
 UDPInterface, 336
 UDPInterface.cpp, 972
 UDPInterface.hpp, 974, 975
 UDPSocket, 339
 ~UDPSocket, 340
 operator<<, 344
 print_, 340
 receive, 341, 342
 send, 343
 UDPSocket.cpp, 976, 977
 UDPSocket.hpp, 979, 980
 operator<<, 980
 unit_tests.cpp, 981, 982
 CATCH_CONFIG_MAIN, 982
 UnixSerialPort, 345
 ~UnixSerialPort, 348

Feature, 347
Parity, 347
print_, 349
read, 349, 351
UnixSerialPort, 348
write, 352
UnixSerialPort.cpp, 982, 983
UnixSerialPort.hpp, 987, 988
UnixSyscalls, 353
 ~UnixSyscalls, 355
 bind, 355
 close, 356
 ioctl, 356
 open, 357
 poll, 357
 read, 357
 recvfrom, 357
 sendto, 358
 socket, 358
 tcgetattr, 359
 tcsetattr, 359
 write, 359
UnixSyscalls.cpp, 989, 990
UnixSyscalls.hpp, 992, 993
UnixUDPSocket, 360
 ~UnixUDPSocket, 362
 print_, 363
 receive, 363, 364
 send, 365, 366
 UnixUDPSocket, 362
UnixUDPSocket.cpp, 994, 995
UnixUDPSocket.hpp, 998, 999
user_manual.md, 1000
Utility Functions, 52
 append, 52
 operator<<, 53
 str, 54
 to_bytes, 54
 to_lower, 55
utility.cpp, 1003, 1004
utility.hpp, 1004, 1005

V1
 Packet, 229

V2
 Packet, 229

VERSION
 packet_v1, 66
 packet_v2, 69

Version
 Packet, 229
 packet_v1::Packet, 243
 packet_v2::Packet, 261

version