

LibZMQUtils

2307.3

Generated by Doxygen 1.9.7

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	9
5.1 amelas Namespace Reference	9
5.2 amelas::common Namespace Reference	9
5.2.1 Typedef Documentation	10
5.2.1.1 ControllerCallback	10
5.2.1.2 GetDatetimeCallback	10
5.2.1.3 GetHomePositionCallback	10
5.2.1.4 SetHomePositionCallback	10
5.2.2 Enumeration Type Documentation	10
5.2.2.1 AmelasServerCommand	10
5.2.2.2 AmelasServerResult	11
5.2.2.3 ControllerError	11
5.2.3 Variable Documentation	11
5.2.3.1 AmelasServerCommandStr	11
5.2.3.2 AmelasServerResultStr	11
5.2.3.3 kMaxCmdId	12
5.2.3.4 kMinCmdId	12
5.3 amelas::utils Namespace Reference	12
5.3.1 Function Documentation	12
5.3.1.1 makeCallback()	12
5.4 zmq Namespace Reference	12
5.4.1 Typedef Documentation	13
5.4.1.1 fd_t	13
5.4.1.2 free_fn	13
5.4.1.3 pollitem_t	13
5.4.2 Function Documentation	14
5.4.2.1 operator!=(())	14
5.4.2.2 operator<()	14
5.4.2.3 operator<<()	14
5.4.2.4 operator<=()	14
5.4.2.5 operator==(())	14
5.4.2.6 operator>()	14

5.4.2.7 operator>=()	15
5.4.2.8 poll() [1/2]	15
5.4.2.9 poll() [2/2]	15
5.4.2.10 proxy() [1/2]	15
5.4.2.11 proxy() [2/2]	15
5.4.2.12 proxy_steerable() [1/2]	15
5.4.2.13 proxy_steerable() [2/2]	16
5.4.2.14 swap() [1/3]	16
5.4.2.15 swap() [2/3]	16
5.4.2.16 swap() [3/3]	16
5.4.2.17 version()	16
5.4.3 Variable Documentation	16
5.4.3.1 from_handle	16
5.5 zmq::detail Namespace Reference	17
5.5.1 Function Documentation	17
5.5.1.1 poll()	17
5.6 zmqutils Namespace Reference	17
5.7 zmqutils::common Namespace Reference	17
5.7.1 Typedef Documentation	18
5.7.1.1 CommandType	18
5.7.1.2 ResultType	19
5.7.2 Enumeration Type Documentation	19
5.7.2.1 ClientResult	19
5.7.2.2 ServerCommand	19
5.7.2.3 ServerResult	19
5.7.3 Variable Documentation	20
5.7.3.1 kClientAlivePeriodMsec	20
5.7.3.2 kDefaultClientAliveTimeoutMsec	20
5.7.3.3 kDefaultServerAliveTimeoutMsec	20
5.7.3.4 kMaxBaseCmdId	21
5.7.3.5 kMinBaseCmdId	21
5.7.3.6 kServerReconnTimes	21
5.7.3.7 kZmqEFSMError	21
5.7.3.8 ServerCommandStr	21
5.7.3.9 ServerResultStr	22
5.8 zmqutils::utils Namespace Reference	22
5.8.1 Typedef Documentation	23
5.8.1.1 HRTIMEPOINTSTD	23
5.8.1.2 SCTIMEPOINTSTD	23
5.8.2 Function Documentation	23
5.8.2.1 binarySerializeDeserialize()	23
5.8.2.2 currentISO8601Date()	23

5.8.2.3 <code>getCurrentPID()</code>	24
5.8.2.4 <code>getHostIPsWithInterfaces()</code>	24
5.8.2.5 <code>getHostname()</code>	24
5.8.2.6 <code>joinArraysConstexpr()</code>	24
5.8.2.7 <code>timePointToIso8601()</code>	24
5.8.2.8 <code>timePointToString()</code>	24
5.9 <code>zmqtls::utils::internal</code> Namespace Reference	25
5.9.1 Function Documentation	25
5.9.1.1 <code>joinArrays()</code>	25
6 Class Documentation	27
6.1 <code>zmq::from_handle_t::_private</code> Struct Reference	27
6.1.1 Detailed Description	27
6.2 <code>amelas::common::AltAzPos</code> Struct Reference	27
6.2.1 Detailed Description	27
6.2.2 Constructor & Destructor Documentation	28
6.2.2.1 <code>AltAzPos()</code> [1/2]	28
6.2.2.2 <code>AltAzPos()</code> [2/2]	28
6.2.3 Member Data Documentation	28
6.2.3.1 <code>az</code>	28
6.2.3.2 <code>el</code>	28
6.3 <code>amelas::AmelasClient</code> Class Reference	28
6.3.1 Detailed Description	29
6.3.2 Constructor & Destructor Documentation	29
6.3.2.1 <code>AmelasClient()</code>	29
6.3.3 Member Function Documentation	29
6.3.3.1 <code>onSendCommand()</code> [1/2]	29
6.3.3.2 <code>onSendCommand()</code> [2/2]	29
6.3.3.3 <code>resetClient()</code>	30
6.3.3.4 <code>sendCommand()</code>	30
6.3.3.5 <code>setClientHostIP()</code>	30
6.3.3.6 <code>setClientId()</code>	30
6.3.3.7 <code>startAutoAlive()</code>	30
6.3.3.8 <code>startClient()</code>	30
6.3.3.9 <code>stopAutoAlive()</code>	30
6.3.3.10 <code>stopClient()</code>	31
6.4 <code>amelas::AmelasController</code> Class Reference	31
6.4.1 Detailed Description	31
6.4.2 Constructor & Destructor Documentation	31
6.4.2.1 <code>AmelasController()</code>	31
6.4.3 Member Function Documentation	31
6.4.3.1 <code>getDatetime()</code>	31

6.4.3.2 getHomePosition()	32
6.4.3.3 setHomePosition()	32
6.5 amelas::AmelasServer Class Reference	32
6.5.1 Detailed Description	33
6.5.2 Constructor & Destructor Documentation	34
6.5.2.1 AmelasServer()	34
6.5.3 Member Function Documentation	34
6.5.3.1 clearCallbacks()	34
6.5.3.2 getCallbackMap()	34
6.5.3.3 getConnectedClients()	34
6.5.3.4 getServerAddresses()	35
6.5.3.5 getServerEndpoint()	35
6.5.3.6 getServerPort()	35
6.5.3.7 getServerWorkerFuture()	36
6.5.3.8 isCallbackSet()	36
6.5.3.9 isWorking()	36
6.5.3.10 onCommandReceived() [1/2]	36
6.5.3.11 onCommandReceived() [2/2]	36
6.5.3.12 onConnected() [1/2]	37
6.5.3.13 onConnected() [2/2]	37
6.5.3.14 onCustomCommandReceived() [1/2]	37
6.5.3.15 onCustomCommandReceived() [2/2]	38
6.5.3.16 onDeadClient() [1/2]	38
6.5.3.17 onDeadClient() [2/2]	38
6.5.3.18 onDisconnected() [1/2]	39
6.5.3.19 onDisconnected() [2/2]	39
6.5.3.20 onInvalidMsgReceived() [1/2]	39
6.5.3.21 onInvalidMsgReceived() [2/2]	39
6.5.3.22 onSendingResponse() [1/2]	40
6.5.3.23 onSendingResponse() [2/2]	40
6.5.3.24 onServerError()	40
6.5.3.25 onServerStart()	41
6.5.3.26 onServerStop()	41
6.5.3.27 onWaitingCommand()	42
6.5.3.28 removeCallback()	42
6.5.3.29 setCallback() [1/2]	42
6.5.3.30 setCallback() [2/2]	42
6.5.3.31 setClientStatusCheck()	42
6.5.3.32 startServer()	43
6.5.3.33 stopServer()	43
6.6 zmqutils::CommandClientBase Class Reference	43
6.6.1 Detailed Description	44

6.6.2 Constructor & Destructor Documentation	44
6.6.2.1 CommandClientBase()	44
6.6.2.2 ~CommandClientBase()	44
6.6.3 Member Function Documentation	44
6.6.3.1 onSendCommand()	44
6.6.3.2 resetClient()	45
6.6.3.3 sendCommand()	45
6.6.3.4 setClientHostIP()	45
6.6.3.5 setClientId()	45
6.6.3.6 startAutoAlive()	45
6.6.3.7 startClient()	45
6.6.3.8 stopAutoAlive()	45
6.6.3.9 stopClient()	46
6.7 zmqutils::common::CommandReply Struct Reference	46
6.7.1 Detailed Description	46
6.7.2 Constructor & Destructor Documentation	46
6.7.2.1 CommandReply()	46
6.7.3 Member Data Documentation	46
6.7.3.1 params	46
6.7.3.2 params_size	47
6.7.3.3 raw_msg	47
6.7.3.4 result	47
6.8 zmqutils::common::CommandRequest Struct Reference	47
6.8.1 Detailed Description	47
6.8.2 Constructor & Destructor Documentation	48
6.8.2.1 CommandRequest()	48
6.8.3 Member Data Documentation	48
6.8.3.1 client	48
6.8.3.2 command	48
6.8.3.3 params	48
6.8.3.4 params_size	48
6.8.3.5 raw_msg	48
6.9 zmqutils::CommandServerBase Class Reference	49
6.9.1 Detailed Description	50
6.9.2 Usage	50
6.9.3 Callbacks	51
6.9.4 Constructor & Destructor Documentation	51
6.9.4.1 CommandServerBase()	51
6.9.4.2 ~CommandServerBase()	52
6.9.5 Member Function Documentation	52
6.9.5.1 getConnectedClients()	52
6.9.5.2 getServerAddresses()	52

6.9.5.3	getServerEndpoint()	53
6.9.5.4	getServerPort()	53
6.9.5.5	getServerWorkerFuture()	53
6.9.5.6	isWorking()	53
6.9.5.7	onCommandReceived()	53
6.9.5.8	onConnected()	54
6.9.5.9	onCustomCommandReceived()	54
6.9.5.10	onDeadClient()	55
6.9.5.11	onDisconnected()	55
6.9.5.12	onInvalidMsgReceived()	56
6.9.5.13	onSendingResponse()	56
6.9.5.14	onServerError()	56
6.9.5.15	onServerStart()	57
6.9.5.16	onServerStop()	57
6.9.5.17	onWaitingCommand()	58
6.9.5.18	setClientStatusCheck()	58
6.9.5.19	startServer()	58
6.9.5.20	stopServer()	59
6.10	zmq::context_t Class Reference	59
6.10.1	Detailed Description	59
6.10.2	Constructor & Destructor Documentation	59
6.10.2.1	context_t() [1/2]	59
6.10.2.2	context_t() [2/2]	60
6.10.2.3	~context_t()	60
6.10.3	Member Function Documentation	60
6.10.3.1	close()	60
6.10.3.2	getctxopt()	60
6.10.3.3	handle()	60
6.10.3.4	operator bool()	60
6.10.3.5	operator void *()	60
6.10.3.6	operator void const *()	61
6.10.3.7	setctxopt()	61
6.10.3.8	shutdown()	61
6.10.3.9	swap()	61
6.11	zmq::error_t Class Reference	61
6.11.1	Detailed Description	62
6.11.2	Constructor & Destructor Documentation	62
6.11.2.1	error_t() [1/2]	62
6.11.2.2	error_t() [2/2]	62
6.11.3	Member Function Documentation	62
6.11.3.1	num()	62
6.11.3.2	what()	62

6.12 <code>zmq::from_handle_t</code> Struct Reference	62
6.12.1 Detailed Description	63
6.12.2 Constructor & Destructor Documentation	63
6.12.2.1 <code>from_handle_t()</code>	63
6.13 <code>zmqutils::common::HostClient</code> Struct Reference	63
6.13.1 Detailed Description	64
6.13.2 Constructor & Destructor Documentation	64
6.13.2.1 <code>HostClient()</code> [1/4]	64
6.13.2.2 <code>HostClient()</code> [2/4]	64
6.13.2.3 <code>HostClient()</code> [3/4]	64
6.13.2.4 <code>HostClient()</code> [4/4]	64
6.13.3 Member Function Documentation	64
6.13.3.1 <code>operator=()</code> [1/2]	64
6.13.3.2 <code>operator=()</code> [2/2]	64
6.13.4 Member Data Documentation	65
6.13.4.1 <code>hostname</code>	65
6.13.4.2 <code>id</code>	65
6.13.4.3 <code>info</code>	65
6.13.4.4 <code>ip</code>	65
6.13.4.5 <code>last_connection</code>	65
6.13.4.6 <code>pid</code>	65
6.14 <code>zmq::message_t</code> Class Reference	66
6.14.1 Detailed Description	66
6.14.2 Constructor & Destructor Documentation	66
6.14.2.1 <code>message_t()</code> [1/5]	66
6.14.2.2 <code>message_t()</code> [2/5]	67
6.14.2.3 <code>message_t()</code> [3/5]	67
6.14.2.4 <code>message_t()</code> [4/5]	67
6.14.2.5 <code>message_t()</code> [5/5]	67
6.14.2.6 <code>~message_t()</code>	67
6.14.3 Member Function Documentation	67
6.14.3.1 <code>copy()</code> [1/2]	67
6.14.3.2 <code>copy()</code> [2/2]	68
6.14.3.3 <code>data()</code> [1/4]	68
6.14.3.4 <code>data()</code> [2/4]	68
6.14.3.5 <code>data()</code> [3/4]	68
6.14.3.6 <code>data()</code> [4/4]	68
6.14.3.7 <code>empty()</code>	68
6.14.3.8 <code>equal()</code>	68
6.14.3.9 <code>get()</code>	69
6.14.3.10 <code>gets()</code>	69
6.14.3.11 <code>handle()</code> [1/2]	69

6.14.3.12 handle() [2/2]	69
6.14.3.13 more()	69
6.14.3.14 move() [1/2]	69
6.14.3.15 move() [2/2]	69
6.14.3.16 operator"!=()	70
6.14.3.17 operator==(70
6.14.3.18 rebuild() [1/5]	70
6.14.3.19 rebuild() [2/5]	70
6.14.3.20 rebuild() [3/5]	70
6.14.3.21 rebuild() [4/5]	70
6.14.3.22 rebuild() [5/5]	70
6.14.3.23 size()	71
6.14.3.24 str()	71
6.14.3.25 swap()	71
6.14.3.26 to_string()	71
6.15 zmq::monitor_t Class Reference	71
6.15.1 Detailed Description	72
6.15.2 Constructor & Destructor Documentation	72
6.15.2.1 monitor_t()	72
6.15.2.2 ~monitor_t()	72
6.15.3 Member Function Documentation	72
6.15.3.1 abort()	72
6.15.3.2 check_event()	73
6.15.3.3 init() [1/2]	73
6.15.3.4 init() [2/2]	73
6.15.3.5 monitor() [1/2]	73
6.15.3.6 monitor() [2/2]	73
6.15.3.7 on_event_accept_failed()	73
6.15.3.8 on_event_accepted()	74
6.15.3.9 on_event_bind_failed()	74
6.15.3.10 on_event_close_failed()	74
6.15.3.11 on_event_closed()	74
6.15.3.12 on_event_connect_delayed()	74
6.15.3.13 on_event_connect_retried()	74
6.15.3.14 on_event_connected()	75
6.15.3.15 on_event_disconnected()	75
6.15.3.16 on_event_handshake_failed_auth()	75
6.15.3.17 on_event_handshake_failed_no_detail()	75
6.15.3.18 on_event_handshake_failed_protocol()	75
6.15.3.19 on_event_handshake_succeeded()	75
6.15.3.20 on_event_listening()	76
6.15.3.21 on_event_unknown()	76

6.15.3.22 on_monitor_started()	76
6.16 zmqutils::utils::NetworkAdapterInfo Struct Reference	76
6.16.1 Detailed Description	76
6.16.2 Member Data Documentation	76
6.16.2.1 descr	76
6.16.2.2 id	77
6.16.2.3 ip	77
6.16.2.4 name	77
6.17 zmqutils::common::RequestData Struct Reference	77
6.17.1 Detailed Description	77
6.17.2 Constructor & Destructor Documentation	78
6.17.2.1 RequestData() [1/2]	78
6.17.2.2 RequestData() [2/2]	78
6.17.3 Member Data Documentation	78
6.17.3.1 command	78
6.17.3.2 params	78
6.17.3.3 params_size	78
6.18 zmq::detail::socket_base Class Reference	78
6.18.1 Detailed Description	79
6.18.2 Constructor & Destructor Documentation	79
6.18.2.1 socket_base() [1/2]	79
6.18.2.2 socket_base() [2/2]	79
6.18.3 Member Function Documentation	80
6.18.3.1 bind() [1/2]	80
6.18.3.2 bind() [2/2]	80
6.18.3.3 connect() [1/2]	80
6.18.3.4 connect() [2/2]	80
6.18.3.5 connected()	80
6.18.3.6 disconnect() [1/2]	80
6.18.3.7 disconnect() [2/2]	80
6.18.3.8 getsockopt() [1/2]	81
6.18.3.9 getsockopt() [2/2]	81
6.18.3.10 send() [1/2]	81
6.18.3.11 send() [2/2]	81
6.18.3.12 setsockopt() [1/2]	81
6.18.3.13 setsockopt() [2/2]	81
6.18.3.14 unbind() [1/2]	82
6.18.3.15 unbind() [2/2]	82
6.18.3.16 ZMQ_CPP11_DEPRECATED()	82
6.18.4 Member Data Documentation	82
6.18.4.1 flags_	82
6.18.4.2 last	82

6.19 zmq::socket_ref Class Reference	82
6.19.1 Detailed Description	83
6.19.2 Constructor & Destructor Documentation	83
6.19.2.1 socket_ref() [1/2]	83
6.19.2.2 socket_ref() [2/2]	83
6.19.3 Member Function Documentation	84
6.19.3.1 bind() [1/2]	84
6.19.3.2 bind() [2/2]	84
6.19.3.3 connect() [1/2]	84
6.19.3.4 connect() [2/2]	84
6.19.3.5 connected()	84
6.19.3.6 disconnect() [1/2]	84
6.19.3.7 disconnect() [2/2]	84
6.19.3.8 getsockopt() [1/2]	85
6.19.3.9 getsockopt() [2/2]	85
6.19.3.10 send() [1/2]	85
6.19.3.11 send() [2/2]	85
6.19.3.12 setsockopt() [1/2]	85
6.19.3.13 setsockopt() [2/2]	85
6.19.3.14 unbind() [1/2]	86
6.19.3.15 unbind() [2/2]	86
6.19.3.16 ZMQ_CPP11_DEPRECATED()	86
6.19.4 Member Data Documentation	86
6.19.4.1 flags_	86
6.19.4.2 last	86
6.20 zmq::socket_t Class Reference	86
6.20.1 Detailed Description	87
6.20.2 Constructor & Destructor Documentation	87
6.20.2.1 socket_t() [1/2]	87
6.20.2.2 socket_t() [2/2]	88
6.20.2.3 ~socket_t()	88
6.20.3 Member Function Documentation	88
6.20.3.1 bind() [1/2]	88
6.20.3.2 bind() [2/2]	88
6.20.3.3 close()	88
6.20.3.4 connect() [1/2]	88
6.20.3.5 connect() [2/2]	88
6.20.3.6 connected()	89
6.20.3.7 disconnect() [1/2]	89
6.20.3.8 disconnect() [2/2]	89
6.20.3.9 getsockopt() [1/2]	89
6.20.3.10 getsockopt() [2/2]	89

6.20.3.11 operator socket_ref()	89
6.20.3.12 operator void *()	89
6.20.3.13 operator void const *()	90
6.20.3.14 send() [1/2]	90
6.20.3.15 send() [2/2]	90
6.20.3.16 setsockopt() [1/2]	90
6.20.3.17 setsockopt() [2/2]	90
6.20.3.18 swap()	90
6.20.3.19 unbind() [1/2]	91
6.20.3.20 unbind() [2/2]	91
6.20.3.21 ZMQ_CPP11_DEPRECATED()	91
6.20.4 Friends And Related Symbol Documentation	91
6.20.4.1 monitor_t	91
6.20.5 Member Data Documentation	91
6.20.5.1 flags_	91
6.20.5.2 last	91
6.21 zmq_event_t Struct Reference	92
6.21.1 Detailed Description	92
6.21.2 Member Data Documentation	92
6.21.2.1 event	92
6.21.2.2 value	92
6.22 zmq_msg_t Struct Reference	92
6.22.1 Detailed Description	92
6.22.2 Member Data Documentation	93
6.22.2.1 _	93
6.23 zmq_pollitem_t Struct Reference	93
6.23.1 Detailed Description	93
6.23.2 Member Data Documentation	93
6.23.2.1 events	93
6.23.2.2 fd	93
6.23.2.3 revents	93
6.23.2.4 socket	93
7 File Documentation	95
7.1 examples/ExampleZMQCommandClientAmelas/AmelasExampleClient/amelas_client.cpp File Reference	95
7.2 amelas_client.cpp	95
7.3 examples/ExampleZMQCommandClientAmelas/AmelasExampleClient/amelas_client.h File Reference	96
7.4 amelas_client.h	96
7.5 examples/ExampleZMQCommandClientAmelas/ExampleZMQClientAmelas.cpp File Reference	97
7.5.1 Function Documentation	98
7.5.1.1 main()	98
7.5.1.2 parseCommand()	98

7.6 ExampleZMQClientAmelas.cpp	98
7.7 examples/ExampleZMQCommanServerAmelas/AmelasExampleController/amelas_controller.h File Reference	101
7.8 amelas_controller.h	101
7.9 examples/ExampleZMQCommanServerAmelas/AmelasExampleController/common.h File Reference	103
7.10 common.h	104
7.11 examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/common.h File Reference	105
7.12 common.h	106
7.13 includes/LibZMQUtils/CommandServerClient/common.h File Reference	106
7.13.1 Detailed Description	108
7.14 common.h	108
7.15 examples/ExampleZMQCommanServerAmelas/AmelasExampleController/utils.h File Reference	112
7.16 utils.h	112
7.17 includes/LibZMQUtils/utils.h File Reference	112
7.17.1 Detailed Description	113
7.17.2 Macro Definition Documentation	114
7.17.2.1 MKGMTIME	114
7.18 utils.h	114
7.19 examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/amelas_server.cpp File Reference	116
7.20 amelas_server.cpp	116
7.21 examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/amelas_server.h File Reference	120
7.22 amelas_server.h	120
7.23 examples/ExampleZMQCommanServerAmelas/ExampleZMQServerAmelas.cpp File Reference	122
7.23.1 Function Documentation	122
7.23.1.1 main()	122
7.23.2 Variable Documentation	123
7.23.2.1 gExitCv	123
7.23.2.2 gMtx	123
7.23.2.3 gSignInterrupt	123
7.24 ExampleZMQServerAmelas.cpp	123
7.25 external/zmq/includes/zmq/zmq.h File Reference	125
7.25.1 Macro Definition Documentation	130
7.25.1.1 EADDRINUSE	130
7.25.1.2 EADDRNOTAVAIL	130
7.25.1.3 EAFNOSUPPORT	130
7.25.1.4 ECONNABORTED	130
7.25.1.5 ECONNREFUSED	130
7.25.1.6 ECONNRESET	130
7.25.1.7 EFSM	131
7.25.1.8 EHOSTUNREACH	131
7.25.1.9 EINPROGRESS	131

7.25.1.10 EMSGSIZE	131
7.25.1.11 EMTHREAD	131
7.25.1.12 ENETDOWN	131
7.25.1.13 ENETRESET	131
7.25.1.14 ENETUNREACH	131
7.25.1.15 ENOBUFS	131
7.25.1.16 ENOCOMPATPROTO	131
7.25.1.17 ENOTCONN	131
7.25.1.18 ENOTSOCK	131
7.25.1.19 ENOTSUP	132
7.25.1.20 EPROTONOSUPPORT	132
7.25.1.21 ETERM	132
7.25.1.22 ETIMEDOUT	132
7.25.1.23 ZMQ_AFFINITY	132
7.25.1.24 ZMQ_BACKLOG	132
7.25.1.25 ZMQ_BINDTODEVICE	132
7.25.1.26 ZMQ_BLOCKY	132
7.25.1.27 ZMQ_CONFLATE	132
7.25.1.28 ZMQ_CONNECT_RID	132
7.25.1.29 ZMQ_CONNECT_ROUTING_ID	132
7.25.1.30 ZMQ_CONNECT_TIMEOUT	132
7.25.1.31 ZMQ_CURVE	133
7.25.1.32 ZMQ_CURVE_PUBLICKEY	133
7.25.1.33 ZMQ_CURVE_SECRETKEY	133
7.25.1.34 ZMQ_CURVE_SERVER	133
7.25.1.35 ZMQ_CURVE_SERVERKEY	133
7.25.1.36 ZMQ DEALER	133
7.25.1.37 ZMQ_DEFINED_STDINT	133
7.25.1.38 ZMQ_DELAY_ATTACH_ON_CONNECT	133
7.25.1.39 ZMQ_DONTWAIT	133
7.25.1.40 ZMQ_EVENT_ACCEPT_FAILED	133
7.25.1.41 ZMQ_EVENT_ACCEPTED	133
7.25.1.42 ZMQ_EVENT_ALL	133
7.25.1.43 ZMQ_EVENT_BIND_FAILED	134
7.25.1.44 ZMQ_EVENT_CLOSE_FAILED	134
7.25.1.45 ZMQ_EVENT_CLOSED	134
7.25.1.46 ZMQ_EVENT_CONNECT_DELAYED	134
7.25.1.47 ZMQ_EVENT_CONNECT_RETRIED	134
7.25.1.48 ZMQ_EVENT_CONNECTED	134
7.25.1.49 ZMQ_EVENT_DISCONNECTED	134
7.25.1.50 ZMQ_EVENT_HANDSHAKE_FAILED_AUTH	134
7.25.1.51 ZMQ_EVENT_HANDSHAKE_FAILED_NO_DETAIL	134

7.25.1.52 ZMQ_EVENT_HANDSHAKE_FAILED_PROTOCOL	134
7.25.1.53 ZMQ_EVENT_HANDSHAKE_SUCCEEDED	134
7.25.1.54 ZMQ_EVENT_LISTENING	134
7.25.1.55 ZMQ_EVENT_MONITOR_STOPPED	135
7.25.1.56 ZMQ_EVENTS	135
7.25.1.57 ZMQ_EXPORT	135
7.25.1.58 ZMQ_FAIL_UNROUTABLE	135
7.25.1.59 ZMQ_FD	135
7.25.1.60 ZMQ_FORWARDER	135
7.25.1.61 ZMQ_GROUP_MAX_LENGTH	135
7.25.1.62 ZMQ_GSSAPI	135
7.25.1.63 ZMQ_GSSAPI_NT_HOSTBASED	135
7.25.1.64 ZMQ_GSSAPI_NT_KRB5_PRINCIPAL	135
7.25.1.65 ZMQ_GSSAPI_NT_USER_NAME	135
7.25.1.66 ZMQ_GSSAPI_PLAINTEXT	135
7.25.1.67 ZMQ_GSSAPI_PRINCIPAL	136
7.25.1.68 ZMQ_GSSAPI_PRINCIPAL_NAMETYPE	136
7.25.1.69 ZMQ_GSSAPI_SERVER	136
7.25.1.70 ZMQ_GSSAPI_SERVICE_PRINCIPAL	136
7.25.1.71 ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAMETYPE	136
7.25.1.72 ZMQ_HANDSHAKE_IVL	136
7.25.1.73 ZMQ_HAS_CAPABILITIES	136
7.25.1.74 ZMQ_HAUSNUMERO	136
7.25.1.75 ZMQ_HAVE_TIMERS	136
7.25.1.76 ZMQ_HEARTBEAT_IVL	136
7.25.1.77 ZMQ_HEARTBEAT_TIMEOUT	136
7.25.1.78 ZMQ_HEARTBEAT_TTL	136
7.25.1.79 ZMQ_IDENTITY	137
7.25.1.80 ZMQ_IMMEDIATE	137
7.25.1.81 ZMQ_INVERT_MATCHING	137
7.25.1.82 ZMQ_IO_THREADS	137
7.25.1.83 ZMQ_IO_THREADS_DFLT	137
7.25.1.84 ZMQ_IPC_FILTER_GID	137
7.25.1.85 ZMQ_IPC_FILTER_PID	137
7.25.1.86 ZMQ_IPC_FILTER_UID	137
7.25.1.87 ZMQ_IPV4ONLY	137
7.25.1.88 ZMQ_IPV6	137
7.25.1.89 ZMQ_LAST_ENDPOINT	137
7.25.1.90 ZMQ_LINGER	137
7.25.1.91 ZMQ_MAKE_VERSION	138
7.25.1.92 ZMQ_MAX_MSGSZ	138
7.25.1.93 ZMQ_MAX_SOCKETS	138

7.25.1.94 ZMQ_MAX_SOCKETS_DFLT	138
7.25.1.95 ZMQ_MAXMSGSIZE	138
7.25.1.96 ZMQ_MECHANISM	138
7.25.1.97 ZMQ_MORE	138
7.25.1.98 ZMQ_MSG_T_SIZE	138
7.25.1.99 ZMQ_MULTICAST_HOPS	138
7.25.1.100 ZMQ_MULTICAST_MAXTPDU	138
7.25.1.101 ZMQ_NOBLOCK	138
7.25.1.102 ZMQ_NULL	139
7.25.1.103 ZMQ_PAIR	139
7.25.1.104 ZMQ_PLAIN	139
7.25.1.105 ZMQ_PLAIN_PASSWORD	139
7.25.1.106 ZMQ_PLAIN_SERVER	139
7.25.1.107 ZMQ_PLAIN_USERNAME	139
7.25.1.108 ZMQ_POLLERR	139
7.25.1.109 ZMQ_POLLIN	139
7.25.1.110 ZMQ_POLLITEMS_DFLT	139
7.25.1.111 ZMQ_POLLOUT	139
7.25.1.112 ZMQ_POLLPRI	139
7.25.1.113 ZMQ_PROBE_ROUTER	139
7.25.1.114 ZMQ_PROTOCOL_ERROR_WS_UNSPECIFIED	140
7.25.1.115 ZMQ_PROTOCOL_ERROR_ZAP_BAD_REQUEST_ID	140
7.25.1.116 ZMQ_PROTOCOL_ERROR_ZAP_BAD_VERSION	140
7.25.1.117 ZMQ_PROTOCOL_ERROR_ZAP_INVALID_METADATA	140
7.25.1.118 ZMQ_PROTOCOL_ERROR_ZAP_INVALID_STATUS_CODE	140
7.25.1.119 ZMQ_PROTOCOL_ERROR_ZAP_MALFORMED_REPLY	140
7.25.1.120 ZMQ_PROTOCOL_ERROR_ZAP_UNSPECIFIED	140
7.25.1.121 ZMQ_PROTOCOL_ERROR_ZMTP_CRYPTOGRAPHIC	140
7.25.1.122 ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_METADATA	140
7.25.1.123 ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_SEQUENCE	140
7.25.1.124 ZMQ_PROTOCOL_ERROR_ZMTP_KEY_EXCHANGE	140
7.25.1.125 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_ERROR	140
7.25.1.126 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_HELLO	141
7.25.1.127 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_INITIATE	141
7.25.1.128 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_MESSAGE	141
7.25.1.129 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_READY	141
7.25.1.130 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_UNSPECIFIED	141
7.25.1.131 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_WELCOME	141
7.25.1.132 ZMQ_PROTOCOL_ERROR_ZMTP_MECHANISM_MISMATCH	141
7.25.1.133 ZMQ_PROTOCOL_ERROR_ZMTP_UNEXPECTED_COMMAND	141
7.25.1.134 ZMQ_PROTOCOL_ERROR_ZMTP_UNSPECIFIED	141
7.25.1.135 ZMQ_PUB	141

7.25.1.136 ZMQ_PULL	141
7.25.1.137 ZMQ_PUSH	141
7.25.1.138 ZMQ_QUEUE	142
7.25.1.139 ZMQ_RATE	142
7.25.1.140 ZMQ_RCVBUF	142
7.25.1.141 ZMQ_RCVHWM	142
7.25.1.142 ZMQ_RCVMORE	142
7.25.1.143 ZMQ_RCVTIMEO	142
7.25.1.144 ZMQ_RECONNECT_IVL	142
7.25.1.145 ZMQ_RECONNECT_IVL_MAX	142
7.25.1.146 ZMQ_RECOVERY_IVL	142
7.25.1.147 ZMQ_REP	142
7.25.1.148 ZMQ_REQ	142
7.25.1.149 ZMQ_REQ_CORRELATE	142
7.25.1.150 ZMQ_REQ_RELAXED	143
7.25.1.151 ZMQ_ROUTER	143
7.25.1.152 ZMQ_ROUTER_BEHAVIOR	143
7.25.1.153 ZMQ_ROUTER_HANOVER	143
7.25.1.154 ZMQ_ROUTER_MANDATORY	143
7.25.1.155 ZMQ_ROUTER_RAW	143
7.25.1.156 ZMQ_ROUTING_ID	143
7.25.1.157 ZMQ_SHARED	143
7.25.1.158 ZMQ_SNDBUF	143
7.25.1.159 ZMQ_SNDHWM	143
7.25.1.160 ZMQ_SNDMORE	143
7.25.1.161 ZMQ_SNDTIMEO	143
7.25.1.162 ZMQ_SOCKET_LIMIT	144
7.25.1.163 ZMQ SOCKS_PROXY	144
7.25.1.164 ZMQ_SRCFD	144
7.25.1.165 ZMQ_STREAM	144
7.25.1.166 ZMQ_STREAM_NOTIFY	144
7.25.1.167 ZMQ_STREAMER	144
7.25.1.168 ZMQ_SUB	144
7.25.1.169 ZMQ_SUBSCRIBE	144
7.25.1.170 ZMQ_TCP_ACCEPT_FILTER	144
7.25.1.171 ZMQ_TCP_KEEPALIVE	144
7.25.1.172 ZMQ_TCP_KEEPALIVE_CNT	144
7.25.1.173 ZMQ_TCP_KEEPALIVE_IDLE	144
7.25.1.174 ZMQ_TCP_KEEPALIVE_INTVL	145
7.25.1.175 ZMQ_TCP_MAXRT	145
7.25.1.176 ZMQ_THREAD_AFFINITY_CPU_ADD	145
7.25.1.177 ZMQ_THREAD_AFFINITY_CPU_REMOVE	145

7.25.1.178 ZMQ_THREAD_NAME_PREFIX	145
7.25.1.179 ZMQ_THREAD_PRIORITY	145
7.25.1.180 ZMQ_THREAD_PRIORITY_DFLT	145
7.25.1.181 ZMQ_THREAD_SAFE	145
7.25.1.182 ZMQ_THREAD_SCHED_POLICY	145
7.25.1.183 ZMQ_THREAD_SCHED_POLICY_DFLT	145
7.25.1.184 ZMQ_TOS	145
7.25.1.185 ZMQ_TYPE	145
7.25.1.186 ZMQ_UNSUBSCRIBE	146
7.25.1.187 ZMQ_USE_FD	146
7.25.1.188 ZMQ_VERSION	146
7.25.1.189 ZMQ_VERSION_MAJOR	146
7.25.1.190 ZMQ_VERSION_MINOR	146
7.25.1.191 ZMQ_VERSION_PATCH	146
7.25.1.192 ZMQ_VMCI_BUFFER_MAX_SIZE	146
7.25.1.193 ZMQ_VMCI_BUFFER_MIN_SIZE	146
7.25.1.194 ZMQ_VMCI_BUFFER_SIZE	146
7.25.1.195 ZMQ_VMCI_CONNECT_TIMEOUT	146
7.25.1.196 ZMQ_XPUB	146
7.25.1.197 ZMQ_XPUB_MANUAL	146
7.25.1.198 ZMQ_XPUB_NODROP	147
7.25.1.199 ZMQ_XPUB_VERBOSE	147
7.25.1.200 ZMQ_XPUB_VERBOSESER	147
7.25.1.201 ZMQ_XPUB_WELCOME_MSG	147
7.25.1.202 ZMQ_XREP	147
7.25.1.203 ZMQ_XREQ	147
7.25.1.204 ZMQ_XSUB	147
7.25.1.205 ZMQ_ZAP_DOMAIN	147
7.25.2 Typedef Documentation	147
7.25.2.1 zmq_fd_t	147
7.25.2.2 zmq_free_fn	147
7.25.2.3 zmq_msg_t	147
7.25.2.4 zmq_pollitem_t	147
7.25.2.5 zmq_thread_fn	148
7.25.2.6 zmq_timer_fn	148
7.25.3 Function Documentation	148
7.25.3.1 zmq_atomic_counter_dec()	148
7.25.3.2 zmq_atomic_counter_destroy()	148
7.25.3.3 zmq_atomic_counter_inc()	148
7.25.3.4 zmq_atomic_counter_new()	148
7.25.3.5 zmq_atomic_counter_set()	148
7.25.3.6 zmq_atomic_counter_value()	148

7.25.3.7 <code>zmq_bind()</code>	148
7.25.3.8 <code>zmq_close()</code>	148
7.25.3.9 <code>zmq_connect()</code>	148
7.25.3.10 <code>zmq_ctx_destroy()</code>	149
7.25.3.11 <code>zmq_ctx_get()</code>	149
7.25.3.12 <code>zmq_ctx_new()</code>	149
7.25.3.13 <code>zmq_ctx_set()</code>	149
7.25.3.14 <code>zmq_ctx_shutdown()</code>	149
7.25.3.15 <code>zmq_ctx_term()</code>	149
7.25.3.16 <code>zmq_curve_keypair()</code>	149
7.25.3.17 <code>zmq_curve_public()</code>	149
7.25.3.18 <code>zmq_device()</code>	149
7.25.3.19 <code>zmq_disconnect()</code>	149
7.25.3.20 <code>zmq_errno()</code>	150
7.25.3.21 <code>zmq_getsockopt()</code>	150
7.25.3.22 <code>zmq_has()</code>	150
7.25.3.23 <code>zmq_init()</code>	150
7.25.3.24 <code>zmq_msg_close()</code>	150
7.25.3.25 <code>zmq_msg_copy()</code>	150
7.25.3.26 <code>zmq_msg_data()</code>	150
7.25.3.27 <code>zmq_msg_get()</code>	150
7.25.3.28 <code>zmq_msg_gets()</code>	150
7.25.3.29 <code>zmq_msg_init()</code>	150
7.25.3.30 <code>zmq_msg_init_data()</code>	151
7.25.3.31 <code>zmq_msg_init_size()</code>	151
7.25.3.32 <code>zmq_msg_more()</code>	151
7.25.3.33 <code>zmq_msg_move()</code>	151
7.25.3.34 <code>zmq_msg_recv()</code>	151
7.25.3.35 <code>zmq_msg_send()</code>	151
7.25.3.36 <code>zmq_msg_set()</code>	151
7.25.3.37 <code>zmq_msg_size()</code>	151
7.25.3.38 <code>zmq_poll()</code>	151
7.25.3.39 <code>zmq_proxy()</code>	152
7.25.3.40 <code>zmq_proxy_steerable()</code>	152
7.25.3.41 <code>zmq_recv()</code>	152
7.25.3.42 <code>zmq_recviov()</code>	152
7.25.3.43 <code>zmq_recvmsg()</code>	152
7.25.3.44 <code>zmq_send()</code>	152
7.25.3.45 <code>zmq_send_const()</code>	152
7.25.3.46 <code>zmq_sendiov()</code>	152
7.25.3.47 <code>zmq_sendmsg()</code>	153
7.25.3.48 <code>zmq_setsockopt()</code>	153

7.25.3.49 <code>zmq_sleep()</code>	153
7.25.3.50 <code>zmq_socket()</code>	153
7.25.3.51 <code>zmq_socket_monitor()</code>	153
7.25.3.52 <code>zmq_stopwatch_intermediate()</code>	153
7.25.3.53 <code>zmq_stopwatch_start()</code>	153
7.25.3.54 <code>zmq_stopwatch_stop()</code>	153
7.25.3.55 <code>zmq_strerror()</code>	153
7.25.3.56 <code>zmq_term()</code>	153
7.25.3.57 <code>zmq_threadclose()</code>	154
7.25.3.58 <code>zmq_threadstart()</code>	154
7.25.3.59 <code>zmq_timers_add()</code>	154
7.25.3.60 <code>zmq_timers_cancel()</code>	154
7.25.3.61 <code>zmq_timers_destroy()</code>	154
7.25.3.62 <code>zmq_timers_execute()</code>	154
7.25.3.63 <code>zmq_timers_new()</code>	154
7.25.3.64 <code>zmq_timers_reset()</code>	154
7.25.3.65 <code>zmq_timers_set_interval()</code>	154
7.25.3.66 <code>zmq_timers_timeout()</code>	154
7.25.3.67 <code>zmq_unbind()</code>	155
7.25.3.68 <code>zmq_version()</code>	155
7.25.3.69 <code>zmq_z85_decode()</code>	155
7.25.3.70 <code>zmq_z85_encode()</code>	155
7.26 <code>zmq.h</code>	155
7.27 external/zmq/includes/zmq/zmq.hpp File Reference	164
7.27.1 Macro Definition Documentation	166
7.27.1.1 <code>CPPZMQ_HAS_INCLUDE_CPP17</code>	166
7.27.1.2 <code>CPPZMQ_HAS_OPTIONAL</code>	166
7.27.1.3 <code>CPPZMQ_HAS_STRING_VIEW</code>	166
7.27.1.4 <code>CPPZMQ_LANG</code>	166
7.27.1.5 <code>CPPZMQ_VERSION</code>	166
7.27.1.6 <code>CPPZMQ_VERSION_MAJOR</code>	166
7.27.1.7 <code>CPPZMQ_VERSION_MINOR</code>	166
7.27.1.8 <code>CPPZMQ_VERSION_PATCH</code>	166
7.27.1.9 <code>ZMQ_ASSERT</code>	166
7.27.1.10 <code>ZMQ_CONSTEXPR_FN</code>	166
7.27.1.11 <code>ZMQ_CONSTEXPR_IF</code>	167
7.27.1.12 <code>ZMQ_CONSTEXPR_VAR</code>	167
7.27.1.13 <code>ZMQ_CPP11_DEPRECATED</code>	167
7.27.1.14 <code>ZMQ_DELETED_FUNCTION</code>	167
7.27.1.15 <code>ZMQ_DEPRECATED</code>	167
7.27.1.16 <code>ZMQ_EXPLICIT</code>	167
7.27.1.17 <code>ZMQ_HAS_PROXY_STEERABLE</code>	167

7.27.1.18 ZMQ_INLINE_VAR	167
7.27.1.19 ZMQ_NEW_MONITOR_EVENT_LAYOUT	167
7.27.1.20 ZMQ_NODISCARD	167
7.27.1.21 ZMQ_NOTHROW	167
7.27.1.22 ZMQ_NULLPTR	167
7.27.1.23 ZMQ_OVERRIDE	168
7.28 zmq.hpp	168
7.29 external/zmq/includes/zmq/zmq_addon.hpp File Reference	199
7.30 zmq_addon.hpp	200
7.31 includes/LibZMQUtils/CommandServerClient/command_client.h File Reference	208
7.31.1 Detailed Description	209
7.32 command_client.h	209
7.33 includes/LibZMQUtils/CommandServerClient/command_server.h File Reference	211
7.33.1 Detailed Description	211
7.34 command_server.h	212
7.35 includes/LibZMQUtils/libzmqutils_global.h File Reference	214
7.35.1 Macro Definition Documentation	214
7.35.1.1 LIBZMQUTILS_EXPORT	214
7.36 libzmqutils_global.h	214
7.37 sources/CommandServerClient/command_client.cpp File Reference	215
7.38 command_client.cpp	215
7.39 sources/CommandServerClient/command_server.cpp File Reference	220
7.39.1 Detailed Description	221
7.40 command_server.cpp	221
7.41 sources/CommandServerClient/common.cpp File Reference	229
7.42 common.cpp	229
7.43 sources/Utils.cpp File Reference	230
7.43.1 Macro Definition Documentation	230
7.43.1.1 _WIN32_WINNT	230
7.44 utils.cpp	230

Index	235
--------------	------------

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

amelas	9
amelas::common	9
amelas::utils	12
zmq	12
zmq::detail	17
zmqutils	17
zmqutils::common	17
zmqutils::utils	22
zmqutils::utils::internal	25

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

zmq::from_handle_t::_private	27
amelas::common::AltAzPos	27
amelas::AmelasController	31
zmquutils::CommandClientBase	43
amelas::AmelasClient	28
zmquutils::common::CommandReply	46
zmquutils::common::CommandRequest	47
zmquutils::CommandServerBase	49
amelas::AmelasServer	32
zmq::context_t	59
std::exception	
zmq::error_t	61
zmq::from_handle_t	62
zmquutils::common::HostClient	63
zmq::message_t	66
zmq::monitor_t	71
zmquutils::utils::NetworkAdapterInfo	76
zmquutils::common::RequestData	77
zmq::detail::socket_base	78
zmq::socket_ref	82
zmq::socket_t	86
zmq_event_t	92
zmq_msg_t	92
zmq_pollitem_t	93

Chapter 3

Class Index

3.1 Class List

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

zmq::from_handle_t::_private	27
amelas::common::AltAzPos	27
amelas::AmelasClient	28
amelas::AmelasController	31
amelas::AmelasServer	32
zmquutils::CommandClientBase	43
zmquutils::common::CommandReply	46
zmquutils::common::CommandRequest	47
zmquutils::CommandServerBase	
This class provides the base structure for a ZeroMQ based command server	49
zmq::context_t	59
zmq::error_t	61
zmq::from_handle_t	62
zmquutils::common::HostClient	63
zmq::message_t	66
zmq::monitor_t	71
zmquutils::utils::NetworkAdapterInfo	76
zmquutils::common::RequestData	77
zmq::detail::socket_base	78
zmq::socket_ref	82
zmq::socket_t	86
zmq_event_t	92
zmq_msg_t	92
zmq_pollitem_t	93

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

examples/ExampleZMQCommandClientAmelas/ ExampleZMQClientAmelas.cpp	97
examples/ExampleZMQCommandClientAmelas/AmelasExampleClient/ amelas_client.cpp	95
examples/ExampleZMQCommandClientAmelas/AmelasExampleClient/ amelas_client.h	96
examples/ExampleZMQCommanServerAmelas/ ExampleZMQServerAmelas.cpp	122
examples/ExampleZMQCommanServerAmelas/AmelasExampleController/ amelas_controller.h	101
examples/ExampleZMQCommanServerAmelas/AmelasExampleController/ common.h	103
examples/ExampleZMQCommanServerAmelas/AmelasExampleController/ utils.h	112
examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/ amelas_server.cpp	116
examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/ amelas_server.h	120
examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/ common.h	105
external/zmq/includes/zmq/ zmq.h	125
external/zmq/includes/zmq/ zmq.hpp	164
external/zmq/includes/zmq/ zmq_addon.hpp	199
includes/LibZMQUtils/ libzmqutils_global.h	214
includes/LibZMQUtils/ utils.h This file contains the declaration of several utilities for the project development	112
includes/LibZMQUtils/CommandServerClient/ command_client.h This file contains the declaration of the CommandClientBase class and related	208
includes/LibZMQUtils/CommandServerClient/ command_server.h This file contains the declaration of the CommandServerBase class and related	211
includes/LibZMQUtils/CommandServerClient/ common.h This file contains common elements for the whole library	106
sources/ utils.cpp	230
sources/CommandServerClient/ command_client.cpp	215
sources/CommandServerClient/ command_server.cpp This file contains the implementation of the CommandServerBase class and related	220
sources/CommandServerClient/ common.cpp	229

Chapter 5

Namespace Documentation

5.1 amelas Namespace Reference

Namespaces

- namespace [common](#)
- namespace [utils](#)

Classes

- class [AmelasClient](#)
- class [AmelasController](#)
- class [AmelasServer](#)

5.2 amelas::common Namespace Reference

Classes

- struct [AltAzPos](#)

Typedefs

- using [SetHomePositionCallback](#) = std::function< [ControllerError](#)(const [AltAzPos](#) &)>
- using [GetHomePositionCallback](#) = std::function< [ControllerError](#)([AltAzPos](#) &)>
- using [GetDatetimeCallback](#) = std::function< [ControllerError](#)(std::string &)>
- using [ControllerCallback](#) = std::variant< [SetHomePositionCallback](#), [GetHomePositionCallback](#), [GetDatetimeCallback](#) >

Enumerations

- enum class [ControllerError](#) : std::uint32_t { [SUCCESS](#) = 0 , [INVALID_POSITION](#) = 1 , [UNSAFE_POSITION](#) = 2 }
- enum class [AmelasServerCommand](#) : zmqutils::common::CommandType { [REQ_SET_DATETIME](#) = 11 , [REQ_GET_DATETIME](#) = 12 , [REQ_SET_HOME_POSITION](#) = 13 , [REQ_GET_HOME_POSITION](#) = 14 , [END_AMELAS_COMMANDS](#) }
- enum class [AmelasServerResult](#) : zmqutils::common::ResultType { [EMPTY_CALLBACK](#) = 21 , [INVALID_CALLBACK](#) = 22 }

Variables

- static constexpr auto [AmelasServerCommandStr](#)
- static constexpr auto [AmelasServerResultStr](#)
- constexpr int [kMinCmdId](#) = static_cast<int>([zmqutils::common::ServerCommand::END_BASE_COMMANDS](#)) + 1
- constexpr int [kMaxCmdId](#) = static_cast<int>([AmelasServerCommand::END_AMELAS_COMMANDS](#)) - 1

5.2.1 Typedef Documentation

5.2.1.1 ControllerCallback

```
using amelas::common::ControllerCallback = typedef std::variant<SetHomePositionCallback, GetHomePositionCallback, GetDatetimeCallback>
```

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

5.2.1.2 GetDatetimeCallback

```
using amelas::common::GetDatetimeCallback = typedef std::function<ControllerError(std::string&)>
```

Definition at line 50 of file [common.h](#).

5.2.1.3 GetHomePositionCallback

```
using amelas::common::GetHomePositionCallback = typedef std::function<ControllerError(AltAzPos&)>
```

Definition at line 49 of file [common.h](#).

5.2.1.4 SetHomePositionCallback

```
using amelas::common::SetHomePositionCallback = typedef std::function<ControllerError(const AltAzPos&)>
```

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

5.2.2 Enumeration Type Documentation

5.2.2.1 AmelasServerCommand

```
enum class amelas::common::AmelasServerCommand : zmqutils::common::CommandType [strong]
```

Enumerator

REQ_SET_DATETIME	
REQ_GET_DATETIME	
REQ_SET_HOME_POSITION	
REQ_GET_HOME_POSITION	
END_AMELAS_COMMANDS	

Definition at line 19 of file [common.h](#).

5.2.2.2 AmelasServerResult

```
enum class amelas::common::AmelasServerResult : zmqutils::common::ResultType [strong]
```

Enumerator

EMPTY_CALLBACK	
INVALID_CALLBACK	

Definition at line 29 of file [common.h](#).

5.2.2.3 ControllerError

```
enum class amelas::common::ControllerError : std::uint32_t [strong]
```

Enumerator

SUCCESS	
INVALID_POSITION	
UNSAFE_POSITION	

Definition at line 29 of file [common.h](#).

5.2.3 Variable Documentation

5.2.3.1 AmelasServerCommandStr

```
constexpr auto amelas::common::AmelasServerCommandStr [static], [constexpr]
```

Initial value:

```
= zmqutils::utils::joinArraysConstexpr(
    zmqutils::common::ServerCommandStr,
    std::array<const char*, 5>
    {
        "REQ_SET_DATETIME",
        "REQ_GET_DATETIME",
        "REQ_SET_HOME_POSITION",
        "REQ_GET_HOME_POSITION",
        "END_DRGG_COMMANDS"
    })
```

Definition at line 36 of file [common.h](#).

5.2.3.2 AmelasServerResultStr

```
constexpr auto amelas::common::AmelasServerResultStr [static], [constexpr]
```

Initial value:

```
= zmqutils::utils::joinArraysConstexpr(
    zmqutils::common::ServerResultStr,
    std::array<const char*, 2>
    {
        "EMPTY_CALLBACK - The external callback for the command is empty.",
        "INVALID_CALLBACK - The external callback for the command is invalid."
    })
```

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

5.2.3.3 kMaxCmdId

```
constexpr int amelas::common::kMaxCmdId = static_cast<int>(AmelasServerCommand::END_AMELAS_COMMANDS)
- 1 [constexpr]
```

Definition at line 58 of file [common.h](#).

5.2.3.4 kMinCmdId

```
constexpr int amelas::common::kMinCmdId = static_cast<int>(zmqutils::common::ServerCommand::END_BASE_COMMANDS
+ 1 [constexpr]
```

Definition at line 57 of file [common.h](#).

5.3 amelas::utils Namespace Reference

Functions

- [template<typename ClassType , typename ReturnType , typename... Args>](#)
static std::function< ReturnType(Args...)> [makeCallback](#) (ClassType *object, ReturnType(ClassType↵
::*memberFunction)(Args...))

5.3.1 Function Documentation

5.3.1.1 makeCallback()

```
template<typename ClassType , typename ReturnType , typename... Args>
static std::function< ReturnType(Args...)> amelas::utils::makeCallback (
    ClassType * object,
    ReturnType (ClassType::*) (Args...) memberFunction ) [static]
```

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

5.4 zmq Namespace Reference

Namespaces

- namespace [detail](#)

Classes

- class [context_t](#)
- class [error_t](#)
- struct [from_handle_t](#)
- class [message_t](#)
- class [monitor_t](#)
- class [socket_ref](#)
- class [socket_t](#)

Typedefs

- typedef [zmq_free_fn](#) [free_fn](#)
- typedef [zmq_pollitem_t](#) [pollitem_t](#)
- typedef int [fd_t](#)

Functions

- int [poll](#) ([zmq_pollitem_t](#) *items_, size_t nitems_, long timeout_=-1)
- int [poll](#) ([zmq_pollitem_t](#) const *items_, size_t nitems_, long timeout_=-1)
- void [version](#) (int *major_, int *minor_, int *patch_)
- void [swap](#) ([message_t](#) &a, [message_t](#) &b) [ZMQ_NOTHROW](#)
- void [swap](#) ([context_t](#) &a, [context_t](#) &b) [ZMQ_NOTHROW](#)
- bool [operator==](#) (const [detail::socket_base](#) &a, const [detail::socket_base](#) &b) [ZMQ_NOTHROW](#)
- bool [operator!=](#) (const [detail::socket_base](#) &a, const [detail::socket_base](#) &b) [ZMQ_NOTHROW](#)
- bool [operator<](#) (const [detail::socket_base](#) &a, const [detail::socket_base](#) &b) [ZMQ_NOTHROW](#)
- bool [operator>](#) (const [detail::socket_base](#) &a, const [detail::socket_base](#) &b) [ZMQ_NOTHROW](#)
- bool [operator<=](#) (const [detail::socket_base](#) &a, const [detail::socket_base](#) &b) [ZMQ_NOTHROW](#)
- bool [operator>=](#) (const [detail::socket_base](#) &a, const [detail::socket_base](#) &b) [ZMQ_NOTHROW](#)
- void [swap](#) ([socket_t](#) &a, [socket_t](#) &b) [ZMQ_NOTHROW](#)
- void [proxy](#) (void *frontend, void *backend, void *capture)
- void [proxy](#) ([socket_ref](#) frontend, [socket_ref](#) backend, [socket_ref](#) capture=[socket_ref](#)())
- void [proxy_steerable](#) (void *frontend, void *backend, void *capture, void *control)
- void [proxy_steerable](#) ([socket_ref](#) frontend, [socket_ref](#) backend, [socket_ref](#) capture, [socket_ref](#) control)
- std::ostream & [operator<<](#) (std::ostream &os, const [message_t](#) &msg)

Variables

- [ZMQ_CONSTEXPR_VAR from_handle_t from_handle](#)

5.4.1 Typedef Documentation

5.4.1.1 fd_t

```
typedef int zmq::fd_t
```

Definition at line 286 of file [zmq.hpp](#).

5.4.1.2 free_fn

```
typedef zmq_free_fn zmq::free_fn
```

Definition at line 275 of file [zmq.hpp](#).

5.4.1.3 pollitem_t

```
typedef zmq_pollitem_t zmq::pollitem_t
```

Definition at line 276 of file [zmq.hpp](#).

5.4.2 Function Documentation

5.4.2.1 `operator!=()`

```
bool zmq::operator!= (
    const detail::socket_base & a,
    const detail::socket_base & b ) [inline]
```

Definition at line 2143 of file [zmq.hpp](#).

5.4.2.2 `operator<()`

```
bool zmq::operator< (
    const detail::socket_base & a,
    const detail::socket_base & b ) [inline]
```

Definition at line 2147 of file [zmq.hpp](#).

5.4.2.3 `operator<<()`

```
std::ostream & zmq::operator<< (
    std::ostream & os,
    const message_t & msg ) [inline]
```

Definition at line 2755 of file [zmq.hpp](#).

5.4.2.4 `operator<=()`

```
bool zmq::operator<= (
    const detail::socket_base & a,
    const detail::socket_base & b ) [inline]
```

Definition at line 2155 of file [zmq.hpp](#).

5.4.2.5 `operator==()`

```
bool zmq::operator== (
    const detail::socket_base & a,
    const detail::socket_base & b ) [inline]
```

Definition at line 2139 of file [zmq.hpp](#).

5.4.2.6 `operator>()`

```
bool zmq::operator> (
    const detail::socket_base & a,
    const detail::socket_base & b ) [inline]
```

Definition at line 2151 of file [zmq.hpp](#).

5.4.2.7 operator>=()

```
bool zmq::operator>= (
    const detail::socket_base & a,
    const detail::socket_base & b ) [inline]
```

Definition at line 2159 of file [zmq.hpp](#).

5.4.2.8 poll() [1/2]

```
int zmq::poll (
    zmq_pollitem_t * items_,
    size_t nitems_,
    long timeout_ = -1 ) [inline]
```

Definition at line 318 of file [zmq.hpp](#).

5.4.2.9 poll() [2/2]

```
int zmq::poll (
    zmq_pollitem_t const * items_,
    size_t nitems_,
    long timeout_ = -1 ) [inline]
```

Definition at line 325 of file [zmq.hpp](#).

5.4.2.10 proxy() [1/2]

```
void zmq::proxy (
    socket_ref frontend,
    socket_ref backend,
    socket_ref capture = socket_ref() ) [inline]
```

Definition at line 2275 of file [zmq.hpp](#).

5.4.2.11 proxy() [2/2]

```
void zmq::proxy (
    void * frontend,
    void * backend,
    void * capture ) [inline]
```

Definition at line 2267 of file [zmq.hpp](#).

5.4.2.12 proxy_steerable() [1/2]

```
void zmq::proxy_steerable (
    socket_ref frontend,
    socket_ref backend,
    socket_ref capture,
    socket_ref control ) [inline]
```

Definition at line 2292 of file [zmq.hpp](#).

5.4.2.13 proxy_steerable() [2/2]

```
void zmq::proxy_steerable (
    void * frontend,
    void * backend,
    void * capture,
    void * control ) [inline]
```

Definition at line 2285 of file [zmq.hpp](#).

5.4.2.14 swap() [1/3]

```
void zmq::swap (
    context_t & a,
    context_t & b ) [inline]
```

Definition at line 912 of file [zmq.hpp](#).

5.4.2.15 swap() [2/3]

```
void zmq::swap (
    message_t & a,
    message_t & b ) [inline]
```

Definition at line 748 of file [zmq.hpp](#).

5.4.2.16 swap() [3/3]

```
void zmq::swap (
    socket_t & a,
    socket_t & b ) [inline]
```

Definition at line 2261 of file [zmq.hpp](#).

5.4.2.17 version()

```
void zmq::version (
    int * major_,
    int * minor_,
    int * patch_ ) [inline]
```

Definition at line 380 of file [zmq.hpp](#).

5.4.3 Variable Documentation**5.4.3.1 from_handle**

[ZMQ_CONSTEXPR_VAR](#) [from_handle_t](#) [zmq::from_handle](#)

Initial value:

```
=
    from_handle_t(from_handle_t::_private())
```

Definition at line 2102 of file [zmq.hpp](#).

5.5 zmq::detail Namespace Reference

Classes

- class [socket_base](#)

Functions

- int [poll](#) ([zmq_pollitem_t](#) *items_, size_t nitems_, long timeout_)

5.5.1 Function Documentation

5.5.1.1 poll()

```
int zmq::detail::poll (  
    zmq\_pollitem\_t * items_,  
    size_t nitems_,  
    long timeout_ ) [inline]
```

Definition at line 305 of file [zmq.hpp](#).

5.6 zmqutils Namespace Reference

Namespaces

- namespace [common](#)
- namespace [utils](#)

Classes

- class [CommandClientBase](#)
- class [CommandServerBase](#)

This class provides the base structure for a ZeroMQ based command server.

5.7 zmqutils::common Namespace Reference

Classes

- struct [CommandReply](#)
- struct [CommandRequest](#)
- struct [HostClient](#)
- struct [RequestData](#)

Typedefs

- using [CommandType](#) = std::uint32_t
Type used for the BaseServerCommand enumeration.
- using [ResultType](#) = std::uint32_t
Type used for the BaseServerResult enumeration.

Enumerations

- enum class [ServerCommand](#) : CommandType {
INVALID_COMMAND = 0 , REQ_CONNECT = 1 , REQ_DISCONNECT = 2 , REQ_ALIVE = 3 ,
RESERVED_COMMANDS = 4 , END_BASE_COMMANDS = 10 }
- enum class [ServerResult](#) : ResultType {
COMMAND_OK = 0 , INTERNAL_ZMQ_ERROR = 1 , EMPTY_MSG = 2 , EMPTY_CLIENT_IP = 3 ,
EMPTY_CLIENT_NAME = 4 , EMPTY_CLIENT_PID = 5 , EMPTY_PARAMS = 6 , TIMEOUT_REACHED = 7 ,
INVALID_PARTS = 8 , UNKNOWN_COMMAND = 9 , INVALID_MSG = 10 , CLIENT_NOT_CONNECTED = 11 ,
ALREADY_CONNECTED = 12 , BAD_PARAMETERS = 13 , COMMAND_FAILED = 14 , NOT_IMPLEMENTED = 15 ,
BAD_NO_PARAMETERS = 16 , END_BASE_ERRORS = 20 }
- enum class [ClientResult](#) : ResultType {
COMMAND_OK = 0 , INTERNAL_ZMQ_ERROR = 1 , EMPTY_MSG = 2 , EMPTY_PARAMS = 6 ,
TIMEOUT_REACHED = 7 , INVALID_PARTS = 8 , INVALID_MSG = 10 , CLIENT_STOPPED = 17 ,
END_BASE_ERRORS = 20 }

Variables

- constexpr int [kDefaultClientAliveTimeoutMsec](#) = 8000
Default timeout for consider a client dead.
- constexpr int [kDefaultServerAliveTimeoutMsec](#) = 3000
Default timeout for consider a server dead.
- constexpr unsigned [kServerReconnTimes](#) = 10
Server reconnection default number of attempts.
- constexpr unsigned [kClientAlivePeriodMsec](#) = 1000
Default period for sending alive commands.
- constexpr int [kZmqEFSMError](#) = 156384765
ZMQ EFSM error.
- constexpr int [kMinBaseCmdId](#) = static_cast<int>(ServerCommand::INVALID_COMMAND) + 1
- constexpr int [kMaxBaseCmdId](#) = static_cast<int>(ServerCommand::END_BASE_COMMANDS) - 1
- static constexpr std::array< const char *, 11 > [ServerCommandStr](#)
- static constexpr std::array< const char *, 21 > [ServerResultStr](#)

5.7.1 Typedef Documentation

5.7.1.1 CommandType

using [zmqutils::common::CommandType](#) = typedef std::uint32_t

Type used for the BaseServerCommand enumeration.

Definition at line 73 of file [common.h](#).

5.7.1.2 ResultType

```
using zmqutils::common::ResultType = typedef std::uint32_t
```

Type used for the BaseServerResult enumeration.

Definition at line 74 of file [common.h](#).

5.7.2 Enumeration Type Documentation

5.7.2.1 ClientResult

```
enum class zmqutils::common::ClientResult : ResultType [strong]
```

Enumerator

COMMAND_OK	
INTERNAL_ZMQ_ERROR	An internal ZeroMQ error occurred.
EMPTY_MSG	The message is empty.
EMPTY_PARAMS	The result parameters are missing or empty.
TIMEOUT_REACHED	The operation timed out, the server could be dead.
INVALID_PARTS	The command has invalid parts.
INVALID_MSG	The message is invalid.
CLIENT_STOPPED	The client is stopped.
END_BASE_ERRORS	Sentinel value indicating the end of the base errors (not is a valid error).

Definition at line 122 of file [common.h](#).

5.7.2.2 ServerCommand

```
enum class zmqutils::common::ServerCommand : CommandType [strong]
```

Enumerator

INVALID_COMMAND	Invalid command.
REQ_CONNECT	Request to connect to the server.
REQ_DISCONNECT	Request to disconnect from the server.
REQ_ALIVE	Request to check if the server is alive and for notify that the client is alive too.
RESERVED_COMMANDS	Sentinel value indicating the start of the reserved commands (not is as a valid msg).
END_BASE_COMMANDS	Sentinel value indicating the end of the base commands (not is as a valid msg).

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

5.7.2.3 ServerResult

```
enum class zmqutils::common::ServerResult : ResultType [strong]
```

Enumerator

COMMAND_OK	The command was executed successfully.
INTERNAL_ZMQ_ERROR	An internal ZeroMQ error occurred.
EMPTY_MSG	The message is empty.
EMPTY_CLIENT_IP	The client IP is missing or empty.
EMPTY_CLIENT_NAME	The client name is missing or empty.
EMPTY_CLIENT_PID	The client pid is missing or empty.
EMPTY_PARAMS	The command parameters are missing or empty.
TIMEOUT_REACHED	The operation timed out, the client could be dead.
INVALID_PARTS	The message has invalid parts.
UNKNOWN_COMMAND	The command is not recognized.
INVALID_MSG	The message is invalid.
CLIENT_NOT_CONNECTED	Not connected to the target.
ALREADY_CONNECTED	Already connected to the target.
BAD_PARAMETERS	The provided parameters are invalid.
COMMAND_FAILED	The command execution failed.
NOT_IMPLEMENTED	The command is not implemented.
BAD_NO_PARAMETERS	The provided number of parameters are invalid.
END_BASE_ERRORS	Sentinel value indicating the end of the base errors (not is a valid error).

Definition at line 98 of file [common.h](#).

5.7.3 Variable Documentation

5.7.3.1 kClientAlivePeriodMsec

```
constexpr unsigned zmqutils::common::kClientAlivePeriodMsec = 1000 [constexpr]
```

Default period for sending alive commands.

Definition at line 66 of file [common.h](#).

5.7.3.2 kDefaultClientAliveTimeoutMsec

```
constexpr int zmqutils::common::kDefaultClientAliveTimeoutMsec = 8000 [constexpr]
```

Default timeout for consider a client dead.

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

5.7.3.3 kDefaultServerAliveTimeoutMsec

```
constexpr int zmqutils::common::kDefaultServerAliveTimeoutMsec = 3000 [constexpr]
```

Default timeout for consider a server dead.

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

5.7.3.4 kMaxBaseCmdId

```
constexpr int zmqutils::common::kMaxBaseCmdId = static_cast<int>(ServerCommand::END_BASE_COMMANDS)
- 1 [constexpr]
```

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

5.7.3.5 kMinBaseCmdId

```
constexpr int zmqutils::common::kMinBaseCmdId = static_cast<int>(ServerCommand::INVALID_COMMAND)
+ 1 [constexpr]
```

Definition at line 138 of file [common.h](#).

5.7.3.6 kServerReconnTimes

```
constexpr unsigned zmqutils::common::kServerReconnTimes = 10 [constexpr]
```

Server reconnection default number of attempts.

Definition at line 65 of file [common.h](#).

5.7.3.7 kZmqEFSMError

```
constexpr int zmqutils::common::kZmqEFSMError = 156384765 [constexpr]
```

ZMQ EFSM error.

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

5.7.3.8 ServerCommandStr

```
constexpr std::array<const char*, 11> zmqutils::common::ServerCommandStr [static], [constexpr]
```

Initial value:

```
{
    "INVALID_COMMAND",
    "REQ_CONNECT",
    "REQ_DISCONNECT",
    "REQ_ALIVE",
    "RESERVED_BASE_COMMAND",
    "RESERVED_BASE_COMMAND",
    "RESERVED_BASE_COMMAND",
    "RESERVED_BASE_COMMAND",
    "RESERVED_BASE_COMMAND",
    "RESERVED_BASE_COMMAND",
    "END_BASE_COMMANDS"
}
```

Definition at line 141 of file [common.h](#).

5.7.3.9 ServerResultStr

```
constexpr std::array<const char*, 21> zmqutils::common::ServerResultStr [static], [constexpr]
```

Initial value:

```
{
    "COMMAND_OK - Command executed.",
    "INTERNAL_ZMQ_ERROR - Internal ZeroMQ error.",
    "EMPTY_MSG - Message is empty.",
    "EMPTY_CLIENT_IP - Client IP missing or empty.",
    "EMPTY_CLIENT_NAME - Client name missing or empty.",
    "EMPTY_CLIENT_PID - Client pid missing or empty.",
    "EMPTY_PARAMS - Command parameters missing or empty.",
    "TIMEOUT_REACHED - Operation timed out.",
    "INVALID_PARTS - Command has invalid parts.",
    "UNKNOWN_COMMAND - Command is not recognized.",
    "INVALID_COMMAND - Command is invalid.",
    "NOT_CONNECTED - Not connected to the server.",
    "ALREADY_CONNECTED - Already connected to the server.",
    "BAD_PARAMETERS - Provided parameters are invalid.",
    "COMMAND_FAILED - Command execution failed.",
    "NOT_IMPLEMENTED - Command is not implemented.",
    "RESERVED_BASE_ERROR",
    "RESERVED_BASE_ERROR",
    "RESERVED_BASE_ERROR",
    "RESERVED_BASE_ERROR",
    "RESERVED_BASE_ERROR"
}
```

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

5.8 zmqutils::utils Namespace Reference

Namespaces

- namespace [internal](#)

Classes

- struct [NetworkAdapterInfo](#)

Typedefs

- using [HRTIMEPOINTSTD](#) = std::chrono::time_point< std::chrono::high_resolution_clock >
High resolution time point to store datetimes (uses Unix Time).
- using [SCTIMEPOINTSTD](#) = std::chrono::steady_clock::time_point
Steady clock time point for measuring intervals.

Functions

- [LIBZMQUTILS_EXPORT](#) void [binarySerializeDeserialize](#) (const void *data, size_t data_size_bytes, void *dest)
Binary serialization and deserialization.
- [LIBZMQUTILS_EXPORT](#) std::vector< [NetworkAdapterInfo](#) > [getHostIPsWithInterfaces](#) ()
- [LIBZMQUTILS_EXPORT](#) std::string [getHostname](#) ()
- [LIBZMQUTILS_EXPORT](#) unsigned [getCurrentPID](#) ()
- [LIBZMQUTILS_EXPORT](#) std::string [timePointToString](#) (const [HRTIMEPOINTSTD](#) &tp, const std::string &format=“%Y-%m-%dT%H:%M:%S”, bool add_ms=true, bool add_ns=false, bool utc=true)
- [LIBZMQUTILS_EXPORT](#) std::string [timePointToIso8601](#) (const [HRTIMEPOINTSTD](#) &tp, bool add_ms=true, bool add_ns=false)
- [LIBZMQUTILS_EXPORT](#) std::string [currentISO8601Date](#) (bool add_ms=true)
- template<typename T, std::size_t N1, std::size_t N2>
constexpr std::array< T, N1+N2 > [joinArraysConstexpr](#) (const std::array< T, N1 > &a1, const std::array< T, N2 > &a2)

5.8.1 Typedef Documentation

5.8.1.1 HRTimePointStd

```
using zmqutils::utils::HRTimePointStd = typedef std::chrono::time_point<std::chrono::high_resolution_clock>
```

High resolution time point to store datetimes (uses Unix Time).

Definition at line 73 of file [utils.h](#).

5.8.1.2 SCTimePointStd

```
using zmqutils::utils::SCTimePointStd = typedef std::chrono::steady_clock::time_point
```

Steady clock time point for measuring intervals.

Definition at line 75 of file [utils.h](#).

5.8.2 Function Documentation

5.8.2.1 binarySerializeDeserialize()

```
void zmqutils::utils::binarySerializeDeserialize (
    const void * data,
    size_t data_size_bytes,
    void * dest )
```

Binary serialization and deserialization.

This function is responsible for binary serialization and deserialization by reversing the byte order of the data in a binary safe manner. This can be used for transforming data from little-endian to big-endian and vice versa.

Parameters

in	<i>data</i>	Pointer to the input data that needs to be serialized/deserialized.
in	<i>data_size_bytes</i>	Size of the input data in bytes.
out	<i>dest</i>	Pointer to the destination where the output (reversed bytes) is to be stored.

Definition at line 156 of file [utils.cpp](#).

5.8.2.2 currentISO8601Date()

```
std::string zmqutils::utils::currentISO8601Date (
    bool add_ms = true )
```

Definition at line 198 of file [utils.cpp](#).

5.8.2.3 getCurrentPID()

```
unsigned zmqutils::utils::getCurrentPID ( )
```

Definition at line 204 of file [utils.cpp](#).

5.8.2.4 getHostIPsWithInterfaces()

```
std::vector< NetworkAdapterInfo > zmqutils::utils::getHostIPsWithInterfaces ( )
```

Definition at line 65 of file [utils.cpp](#).

5.8.2.5 getHostname()

```
std::string zmqutils::utils::getHostname ( )
```

Definition at line 127 of file [utils.cpp](#).

5.8.2.6 joinArraysConstexpr()

```
template<typename T , std::size_t N1, std::size_t N2>  
constexpr std::array< T, N1+N2 > zmqutils::utils::joinArraysConstexpr (   
    const std::array< T, N1 > & a1,  
    const std::array< T, N2 > & a2 ) [constexpr]
```

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

5.8.2.7 timePointToIso8601()

```
std::string zmqutils::utils::timePointToIso8601 (   
    const HRTimestamp & tp,  
    bool add_ms = true,  
    bool add_ns = false )
```

Definition at line 192 of file [utils.cpp](#).

5.8.2.8 timePointToString()

```
std::string zmqutils::utils::timePointToString (   
    const HRTimestamp & tp,  
    const std::string & format = "%Y-%m-%dT%H:%M:%S",  
    bool add_ms = true,  
    bool add_ns = false,  
    bool utc = true )
```

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

5.9 zmqutils::utils::internal Namespace Reference

Functions

- `template<typename T, std::size_t... Is1, std::size_t... Is2>`
`constexpr std::array< T, sizeof...(Is1)+sizeof...(Is2)> joinArrays (const std::array< T, sizeof...(Is1)> &a1,`
`const std::array< T, sizeof...(Is2)> &a2, std::index_sequence< Is1... >, std::index_sequence< Is2... >)`

5.9.1 Function Documentation

5.9.1.1 `joinArrays()`

```
template<typename T, std::size_t... Is1, std::size_t... Is2>
constexpr std::array< T, sizeof...(Is1)+sizeof...(Is2)> zmqutils::utils::internal::joinArrays
(
    const std::array< T, sizeof...(Is1)> & a1,
    const std::array< T, sizeof...(Is2)> & a2,
    std::index_sequence< Is1... >,
    std::index_sequence< Is2... > ) [constexpr]
```

Definition at line 117 of file [utils.h](#).

Chapter 6

Class Documentation

6.1 zmq::from_handle_t::_private Struct Reference

```
#include <zmq.hpp>
```

6.1.1 Detailed Description

Definition at line 2096 of file [zmq.hpp](#).

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

- [external/zmq/includes/zmq/zmq.hpp](#)

6.2 amelas::common::AltAzPos Struct Reference

```
#include <common.h>
```

Public Member Functions

- [AltAzPos](#) (double [az](#), double [el](#))
- [AltAzPos](#) ()

Public Attributes

- double [az](#)
- double [el](#)

6.2.1 Detailed Description

Definition at line 36 of file [common.h](#).

6.2.2 Constructor & Destructor Documentation

6.2.2.1 AltAzPos() [1/2]

```
amelas::common::AltAzPos::AltAzPos (
    double az,
    double el ) [inline]
```

Definition at line 38 of file [common.h](#).

6.2.2.2 AltAzPos() [2/2]

```
amelas::common::AltAzPos::AltAzPos ( ) [inline]
```

Definition at line 41 of file [common.h](#).

6.2.3 Member Data Documentation

6.2.3.1 az

```
double amelas::common::AltAzPos::az
```

Definition at line 43 of file [common.h](#).

6.2.3.2 el

```
double amelas::common::AltAzPos::el
```

Definition at line 44 of file [common.h](#).

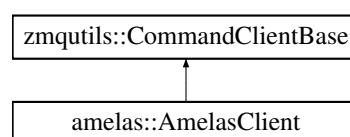
The documentation for this struct was generated from the following file:

- [examples/ExampleZMQCommanServerAmelas/AmelasExampleController/common.h](#)

6.3 amelas::AmelasClient Class Reference

```
#include <amelas_client.h>
```

Inheritance diagram for amelas::AmelasClient:



Public Member Functions

- [AmelasClient](#) (const std::string &server_endpoint)
- bool [startClient](#) (const std::string &interface_name)
- void [stopClient](#) ()
- void [resetClient](#) ()
- void [startAutoAlive](#) ()
- void [stopAutoAlive](#) ()
- void [setClientHostIP](#) (const std::string &interf)
- void [setClientId](#) (const std::string &id)
- ClientResult [sendCommand](#) (const RequestData &, [CommandReply](#) &)

Protected Member Functions

- virtual void [onSendCommand](#) (const RequestData &, const zmq::multipart_t &)=0

Private Member Functions

- void [onSendCommand](#) (const RequestData &req, const zmq::multipart_t &msg) override

6.3.1 Detailed Description

Definition at line 36 of file [amelas_client.h](#).

6.3.2 Constructor & Destructor Documentation

6.3.2.1 AmelasClient()

```
amelas::AmelasClient::AmelasClient (
    const std::string & server_endpoint )
```

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

6.3.3 Member Function Documentation

6.3.3.1 onSendCommand() [1/2]

```
virtual void zmqutils::CommandClientBase::onSendCommand (
    const RequestData & ,
    const zmq::multipart_t & ) [protected], [pure virtual], [inherited]
```

6.3.3.2 onSendCommand() [2/2]

```
void amelas::AmelasClient::onSendCommand (
    const RequestData & req,
    const zmq::multipart_t & msg ) [override], [private]
```

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

6.3.3.3 resetClient()

```
void zmqutils::CommandClientBase::resetClient ( ) [inherited]
```

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

6.3.3.4 sendCommand()

```
ClientResult zmqutils::CommandClientBase::sendCommand (
    const RequestData & msg,
    CommandReply & reply ) [inherited]
```

Definition at line 164 of file [command_client.cpp](#).

6.3.3.5 setClientHostIP()

```
void zmqutils::CommandClientBase::setClientHostIP (
    const std::string & interf ) [inherited]
```

Definition at line 160 of file [command_client.cpp](#).

6.3.3.6 setClientId()

```
void zmqutils::CommandClientBase::setClientId (
    const std::string & id ) [inherited]
```

Definition at line 162 of file [command_client.cpp](#).

6.3.3.7 startAutoAlive()

```
void zmqutils::CommandClientBase::startAutoAlive ( ) [inherited]
```

Definition at line 144 of file [command_client.cpp](#).

6.3.3.8 startClient()

```
bool zmqutils::CommandClientBase::startClient (
    const std::string & interface_name ) [inherited]
```

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

6.3.3.9 stopAutoAlive()

```
void zmqutils::CommandClientBase::stopAutoAlive ( ) [inherited]
```

Definition at line 150 of file [command_client.cpp](#).

6.3.3.10 stopClient()

```
void zmqutils::CommandClientBase::stopClient ( ) [inherited]
```

Definition at line 94 of file [command_client.cpp](#).

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

- examples/ExampleZMQCommandClientAmelas/AmelasExampleClient/[amelas_client.h](#)
- examples/ExampleZMQCommandClientAmelas/AmelasExampleClient/[amelas_client.cpp](#)

6.4 amelas::AmelasController Class Reference

```
#include <amelas_controller.h>
```

Public Member Functions

- [AmelasController](#) ()
- ControllerError [setHomePosition](#) (const [AltAzPos](#) &pos)
- ControllerError [getHomePosition](#) ([AltAzPos](#) &pos)
- ControllerError [getDatetime](#) (std::string &)

6.4.1 Detailed Description

Definition at line 52 of file [amelas_controller.h](#).

6.4.2 Constructor & Destructor Documentation

6.4.2.1 AmelasController()

```
amelas::AmelasController::AmelasController ( ) [inline]
```

Definition at line 57 of file [amelas_controller.h](#).

6.4.3 Member Function Documentation

6.4.3.1 getDatetime()

```
ControllerError amelas::AmelasController::getDatetime (
    std::string & ) [inline]
```

Definition at line 100 of file [amelas_controller.h](#).

6.4.3.2 getHomePosition()

```
ControllerError amelas::AmelasController::getHomePosition (
    AltAzPos & pos ) [inline]
```

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

6.4.3.3 setHomePosition()

```
ControllerError amelas::AmelasController::setHomePosition (
    const AltAzPos & pos ) [inline]
```

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

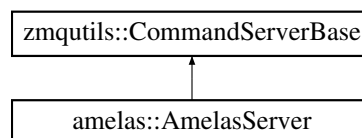
The documentation for this class was generated from the following file:

- [examples/ExampleZMQCommanServerAmelas/AmelasExampleController/amelas_controller.h](#)

6.5 amelas::AmelasServer Class Reference

```
#include <amelas_server.h>
```

Inheritance diagram for amelas::AmelasServer:



Public Member Functions

- [AmelasServer](#) (unsigned port, const std::string &local_addr="*")
- const std::map< [common::AmelasServerCommand](#), [common::ControllerCallback](#) > & [getCallbackMap](#) () const
- void [setCallback](#) ([common::AmelasServerCommand](#) command, [common::ControllerCallback](#) callback)
- template<typename ClassT = void, typename ReturnT = void, typename... Args>
void [setCallback](#) ([common::AmelasServerCommand](#) command, ClassT *object, ReturnT(ClassT &&...)*callback)(Args...)
- void [removeCallback](#) ([common::AmelasServerCommand](#))
- void [clearCallbacks](#) ()
- bool [isCallbackSet](#) ([common::AmelasServerCommand](#)) const
- const unsigned & [getServerPort](#) () const
Get the port number used by the server for incoming connections.
- const std::vector< [NetworkAdapterInfo](#) > & [getServerAddresses](#) () const
Get the network adapter addresses used by the server.
- const std::string & [getServerEndpoint](#) () const
Get the endpoint of the server.
- const std::future< void > & [getServerWorkerFuture](#) () const

- Get the future associated with the server's worker thread.*

 - const std::map< std::string, HostClient > & [getConnectedClients](#) () const

Get a const reference to the map of connected clients.
- bool [isWorking](#) () const

Check if the server is currently working.
- void [setClientStatusCheck](#) (bool)

Enables or disables the client's alive status checking.
- void [startServer](#) ()

Starts the command server.
- void [stopServer](#) ()

Stops the command server.

Protected Member Functions

- virtual void [onConnected](#) (const HostClient &)=0

Base connected callback. Subclasses must override this function.
- virtual void [onDisconnected](#) (const HostClient &)=0

Base disconnected callback. Subclasses must override this function.
- virtual void [onDeadClient](#) (const HostClient &)=0

Base dead client callback. Subclasses must override this function.
- virtual void [onInvalidMsgReceived](#) (const CommandRequest &)=0

Base invalid message received callback. Subclasses must override this function.
- virtual void [onCommandReceived](#) (const CommandRequest &)=0

Base command received callback. Subclasses must override this function.
- virtual void [onCustomCommandReceived](#) (const CommandRequest &, CommandReply &)

Base custom command received callback. Subclasses must override this function.
- virtual void [onSendingResponse](#) (const CommandReply &)=0

Base sending response callback. Subclasses must override this function.

Private Member Functions

- virtual void [onCustomCommandReceived](#) (const CommandRequest &, CommandReply &) final
- virtual void [onServerStart](#) () final

Base server start callback. Subclasses must override this function.
- virtual void [onServerStop](#) () final

Base server stop callback. Subclasses must override this function.
- virtual void [onWaitingCommand](#) () final

Base waiting command callback. Subclasses must override this function.
- virtual void [onDeadClient](#) (const HostClient &) final
- virtual void [onConnected](#) (const HostClient &) final
- virtual void [onDisconnected](#) (const HostClient &) final
- virtual void [onCommandReceived](#) (const CommandRequest &) final
- virtual void [onInvalidMsgReceived](#) (const CommandRequest &) final
- virtual void [onSendingResponse](#) (const CommandReply &) final
- virtual void [onServerError](#) (const zmq::error_t &, const std::string &ext_info) final

Base server error callback. Subclasses must override this function.

6.5.1 Detailed Description

Definition at line 34 of file [amelas_server.h](#).

6.5.2 Constructor & Destructor Documentation

6.5.2.1 AmelasServer()

```
amelas::AmelasServer::AmelasServer (
    unsigned port,
    const std::string & local_addr = "" )
```

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

6.5.3 Member Function Documentation

6.5.3.1 clearCallbacks()

```
void amelas::AmelasServer::clearCallbacks ( )
```

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

6.5.3.2 getCallbackMap()

```
const std::map< AmelasServerCommand, common::ControllerCallback > & amelas::AmelasServer↵
::getCallbackMap ( ) const
```

Definition at line 22 of file [amelas_server.cpp](#).

6.5.3.3 getConnectedClients()

```
const std::map< std::string, HostClient > & zmqutils::CommandServerBase::getConnectedClients (
) const [inherited]
```

Get a const reference to the map of connected clients.

This function returns a const reference to a `std::map<std::string, HostClient>` representing the list of connected clients. Each entry in the map consists of a string key (client identifier) and a `HostClient` object containing information about the connected client.

Returns

A const reference to the map of connected clients.

Definition at line 77 of file [command_server.cpp](#).

6.5.3.4 getServerAddresses()

```
const std::vector< utils::NetworkAdapterInfo > & zmqutils::CommandServerBase::getServerAddresses ( ) const [inherited]
```

Get the network adapter addresses used by the server.

This function returns a const reference to a vector of NetworkAdapterInfo objects. Each NetworkAdapterInfo object contains information about a network adapter used by the server for communication.

Returns

A const reference to a vector of NetworkAdapterInfo objects.

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

6.5.3.5 getServerEndpoint()

```
const std::string & zmqutils::CommandServerBase::getServerEndpoint ( ) const [inherited]
```

Get the endpoint of the server.

This function returns a const reference to a string representing the server's endpoint. The endpoint typically includes the IP address and port number.

Returns

A const reference to the server's endpoint.

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

6.5.3.6 getServerPort()

```
const unsigned & zmqutils::CommandServerBase::getServerPort ( ) const [inherited]
```

Get the port number used by the server for incoming connections.

Returns

A const reference to the port number of the server.

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

6.5.3.7 `getServerWorkerFuture()`

```
const std::future< void > & zmqutils::CommandServerBase::getServerWorkerFuture ( ) const  
[inherited]
```

Get the future associated with the server's worker thread.

This function returns a const reference to a `std::future<void>` object representing the asynchronous worker thread that is running the server. The `std::future` object can be used to check the status of the worker thread or wait for it to complete.

Returns

A const reference to the server's worker thread future.

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

6.5.3.8 `isCallbackSet()`

```
bool amelas::AmelasServer::isCallbackSet (   
    common::AmelasServerCommand command ) const
```

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

6.5.3.9 `isWorking()`

```
bool zmqutils::CommandServerBase::isWorking ( ) const [inline], [inherited]
```

Check if the server is currently working.

This function returns a boolean value indicating whether the server is currently active and working. If the server is working, it means it is processing incoming connections or performing its intended tasks.

Returns

True if the server is working, false otherwise.

Definition at line 226 of file [command_server.h](#).

6.5.3.10 `onCommandReceived()` [1/2]

```
void amelas::AmelasServer::onCommandReceived (   
    const CommandRequest & cmd_req ) [final], [private], [virtual]
```

Definition at line 286 of file [amelas_server.cpp](#).

6.5.3.11 `onCommandReceived()` [2/2]

```
virtual void zmqutils::CommandServerBase::onCommandReceived (   
    const CommandRequest & ) [protected], [pure virtual], [inherited]
```

Base command received callback. Subclasses must override this function.

Parameters

<i>The</i>	CommandRequest object representing the command execution request.
------------	---

Warning

This internal callback must be used for log or similar purposes. For specific custom command functionalities use the internal "onCustomCommandReceived".

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

6.5.3.12 onConnected() [1/2]

```
void amelas::AmelasServer::onConnected (
    const HostClient & client ) [final], [private], [virtual]
```

Definition at line 243 of file [amelas_server.cpp](#).

6.5.3.13 onConnected() [2/2]

```
virtual void zmqutils::CommandServerBase::onConnected (
    const HostClient & ) [protected], [pure virtual], [inherited]
```

Base connected callback. Subclasses must override this function.

Parameters

<i>The</i>	HostClient object representing the connected client.
------------	--

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

6.5.3.14 onCustomCommandReceived() [1/2]

```
void zmqutils::CommandServerBase::onCustomCommandReceived (
    const CommandRequest & ,
    CommandReply & rep ) [protected], [virtual], [inherited]
```

Base custom command received callback. Subclasses must override this function.

Parameters

in	<i>The</i>	CommandRequest object representing the command execution request.
out	<i>The</i>	CommandReply object representing the command execution reply.

Note

This function must process the CommandRequest (function parameter input) and update the CommandReply (function parameter output), especially the result code.

Warning

All internal callbacks, including this one, must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

Definition at line 630 of file [command_server.cpp](#).

6.5.3.15 onCustomCommandReceived() [2/2]

```
void amelas::AmelasServer::onCustomCommandReceived (
    const CommandRequest & request,
    CommandReply & reply ) [final], [private], [virtual]
```

Definition at line 149 of file [amelas_server.cpp](#).

6.5.3.16 onDeadClient() [1/2]

```
void amelas::AmelasServer::onDeadClient (
    const HostClient & client ) [final], [private], [virtual]
```

Definition at line 228 of file [amelas_server.cpp](#).

6.5.3.17 onDeadClient() [2/2]

```
virtual void zmqutils::CommandServerBase::onDeadClient (
    const HostClient & ) [protected], [pure virtual], [inherited]
```

Base dead client callback. Subclasses must override this function.

Parameters

<i>The</i>	HostClient object representing the dead client.
------------	---

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

6.5.3.18 onDisconnected() [1/2]

```
void amelas::AmelasServer::onDisconnected (
    const HostClient & client ) [final], [private], [virtual]
```

Definition at line 258 of file [amelas_server.cpp](#).

6.5.3.19 onDisconnected() [2/2]

```
virtual void zmqutils::CommandServerBase::onDisconnected (
    const HostClient & ) [protected], [pure virtual], [inherited]
```

Base disconnected callback. Subclasses must override this function.

Parameters

<i>The</i>	HostClient object representing the disconnected client.
------------	---

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

6.5.3.20 onInvalidMsgReceived() [1/2]

```
void amelas::AmelasServer::onInvalidMsgReceived (
    const CommandRequest & cmd_req ) [final], [private], [virtual]
```

Definition at line 302 of file [amelas_server.cpp](#).

6.5.3.21 onInvalidMsgReceived() [2/2]

```
virtual void zmqutils::CommandServerBase::onInvalidMsgReceived (
    const CommandRequest & ) [protected], [pure virtual], [inherited]
```

Base invalid message received callback. Subclasses must override this function.

Parameters

<i>The</i>	CommandRequest object representing the invalid command request.
------------	---

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

6.5.3.22 onSendingResponse() [1/2]

```
void amelas::AmelasServer::onSendingResponse (
    const CommandReply & cmd_rep ) [final], [private], [virtual]
```

Definition at line 319 of file [amelas_server.cpp](#).

6.5.3.23 onSendingResponse() [2/2]

```
virtual void zmqutils::CommandServerBase::onSendingResponse (
    const CommandReply & ) [protected], [pure virtual], [inherited]
```

Base sending response callback. Subclasses must override this function.

Parameters

<i>The</i>	CommandReply object representing the command reply being sent.
------------	--

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

6.5.3.24 onServerError()

```
void amelas::AmelasServer::onServerError (
    const zmq::error_t & error,
    const std::string & ext_info ) [final], [private], [virtual]
```

Base server error callback. Subclasses must override this function.

Parameters

<i>The</i>	zmq::error_t object representing the error that occurred.
<i>Optional</i>	additional information or context related to the error. It is an empty string by default.

Note

The [zmq::error_t](#) class provides information about ZeroMQ errors. You can access the error code, description, and other details using the methods provided by [zmq::error_t](#).

Warning

If this function is not overridden in subclasses, it will not handle server errors, and errors may not be handled properly.

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

Implements [zmquils::CommandServerBase](#).

Definition at line 273 of file [amelas_server.cpp](#).

6.5.3.25 onServerStart()

```
void amelas::AmelasServer::onServerStart ( ) [final], [private], [virtual]
```

Base server start callback. Subclasses must override this function.

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

Implements [zmquils::CommandServerBase](#).

Definition at line 184 of file [amelas_server.cpp](#).

6.5.3.26 onServerStop()

```
void amelas::AmelasServer::onServerStop ( ) [final], [private], [virtual]
```

Base server stop callback. Subclasses must override this function.

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

Implements [zmquils::CommandServerBase](#).

Definition at line 208 of file [amelas_server.cpp](#).

6.5.3.27 onWaitingCommand()

```
void amelas::AmelasServer::onWaitingCommand ( ) [final], [private], [virtual]
```

Base waiting command callback. Subclasses must override this function.

Note

This function is intended to be called during the server's main loop when there are no incoming requests to process. Subclasses may implement this function to perform periodic checks, cleanup tasks, or other non-blocking activities while waiting for requests.

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

Implements [zmquils::CommandServerBase](#).

Definition at line 218 of file [amelas_server.cpp](#).

6.5.3.28 removeCallback()

```
void amelas::AmelasServer::removeCallback (
    common::AmelasServerCommand command )
```

Definition at line 32 of file [amelas_server.cpp](#).

6.5.3.29 setCallback() [1/2]

```
template<typename ClassT = void, typename ReturnT = void, typename... Args>
void amelas::AmelasServer::setCallback (
    common::AmelasServerCommand command,
    ClassT * object,
    ReturnT(ClassT::*)(Args...) callback ) [inline]
```

Definition at line 45 of file [amelas_server.h](#).

6.5.3.30 setCallback() [2/2]

```
void amelas::AmelasServer::setCallback (
    common::AmelasServerCommand command,
    common::ControllerCallback callback )
```

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

6.5.3.31 setClientStatusCheck()

```
void zmquils::CommandServerBase::setClientStatusCheck (
    bool ) [inherited]
```

Enables or disables the client's alive status checking.

Enables or disables the checking of the client's alive status. This is a very important functionality in the context of critical systems that often use these types of servers.

Parameters

<i>The</i>	desired status of the client's alive status checking (true to enable, false to disable).
------------	--

Warning

It is strongly recommended to keep this check active, due to the critical nature of the systems that usually use this kind of servers. Disabling the client alive status check could result in unexpected behavior or system instability in case of sudden client disconnections or failures.

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

6.5.3.32 startServer()

```
void zmqutils::CommandServerBase::startServer ( ) [inherited]
```

Starts the command server.

If the server is already running, the function does nothing. Otherwise, it creates the ZMQ context if it doesn't exist and launches the server worker in a separate thread.

Definition at line 97 of file [command_server.cpp](#).

6.5.3.33 stopServer()

```
void zmqutils::CommandServerBase::stopServer ( ) [inherited]
```

Stops the command server.

If the server is already stopped, the function does nothing. Otherwise deletes the ZMQ context and cleans up the connected clients.

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

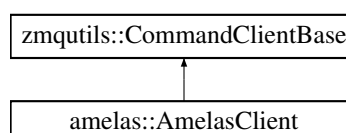
The documentation for this class was generated from the following files:

- [examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/amelas_server.h](#)
- [examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/amelas_server.cpp](#)

6.6 zmqutils::CommandClientBase Class Reference

```
#include <command_client.h>
```

Inheritance diagram for zmqutils::CommandClientBase:



Public Member Functions

- [CommandClientBase](#) (const std::string &server_endpoint)
- virtual [~CommandClientBase](#) ()
- bool [startClient](#) (const std::string &interface_name)
- void [stopClient](#) ()
- void [resetClient](#) ()
- void [startAutoAlive](#) ()
- void [stopAutoAlive](#) ()
- void [setClientHostIP](#) (const std::string &interf)
- void [setClientId](#) (const std::string &id)
- ClientResult [sendCommand](#) (const [RequestData](#) &, [CommandReply](#) &)

Protected Member Functions

- virtual void [onSendCommand](#) (const [RequestData](#) &, const zmq::multipart_t &)=0

6.6.1 Detailed Description

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

6.6.2 Constructor & Destructor Documentation

6.6.2.1 CommandClientBase()

```
zmqutils::CommandClientBase::CommandClientBase (
    const std::string & server_endpoint )
```

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

6.6.2.2 ~CommandClientBase()

```
zmqutils::CommandClientBase::~~CommandClientBase ( ) [virtual]
```

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

6.6.3 Member Function Documentation

6.6.3.1 onSendCommand()

```
virtual void zmqutils::CommandClientBase::onSendCommand (
    const RequestData & ,
    const zmq::multipart_t & ) [protected], [pure virtual]
```

6.6.3.2 resetClient()

```
void zmqutils::CommandClientBase::resetClient ( )
```

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

6.6.3.3 sendCommand()

```
ClientResult zmqutils::CommandClientBase::sendCommand (
    const RequestData & msg,
    CommandReply & reply )
```

Definition at line 164 of file [command_client.cpp](#).

6.6.3.4 setClientHostIP()

```
void zmqutils::CommandClientBase::setClientHostIP (
    const std::string & interf )
```

Definition at line 160 of file [command_client.cpp](#).

6.6.3.5 setClientId()

```
void zmqutils::CommandClientBase::setClientId (
    const std::string & id )
```

Definition at line 162 of file [command_client.cpp](#).

6.6.3.6 startAutoAlive()

```
void zmqutils::CommandClientBase::startAutoAlive ( )
```

Definition at line 144 of file [command_client.cpp](#).

6.6.3.7 startClient()

```
bool zmqutils::CommandClientBase::startClient (
    const std::string & interface_name )
```

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

6.6.3.8 stopAutoAlive()

```
void zmqutils::CommandClientBase::stopAutoAlive ( )
```

Definition at line 150 of file [command_client.cpp](#).

6.6.3.9 stopClient()

```
void zmqutils::CommandClientBase::stopClient ( )
```

Definition at line 94 of file [command_client.cpp](#).

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

- includes/LibZMQUtils/CommandServerClient/[command_client.h](#)
- sources/CommandServerClient/[command_client.cpp](#)

6.7 zmqutils::common::CommandReply Struct Reference

```
#include <common.h>
```

Public Member Functions

- [CommandReply](#) ()

Public Attributes

- `std::unique_ptr< std::uint8_t >` [params](#)
- `zmq::multipart_t` [raw_msg](#)
- `size_t` [params_size](#)
- `ServerResult` [result](#)

6.7.1 Detailed Description

Definition at line 225 of file [common.h](#).

6.7.2 Constructor & Destructor Documentation

6.7.2.1 CommandReply()

```
zmqutils::common::CommandReply::CommandReply ( ) [inline]
```

Definition at line 227 of file [common.h](#).

6.7.3 Member Data Documentation

6.7.3.1 params

```
std::unique_ptr<std::uint8_t> zmqutils::common::CommandReply::params
```

Definition at line 233 of file [common.h](#).

6.7.3.2 params_size

size_t zmqutils::common::CommandReply::params_size

Definition at line 235 of file [common.h](#).

6.7.3.3 raw_msg

zmq::multipart_t zmqutils::common::CommandReply::raw_msg

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

6.7.3.4 result

ServerResult zmqutils::common::CommandReply::result

Definition at line 236 of file [common.h](#).

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

- [includes/LibZMQUtils/CommandServerClient/common.h](#)

6.8 zmqutils::common::CommandRequest Struct Reference

```
#include <common.h>
```

Public Member Functions

- [CommandRequest](#) ()

Public Attributes

- [HostClient](#) client
- [ServerCommand](#) command
- std::unique_ptr< std::uint8_t > [params](#)
- zmq::multipart_t [raw_msg](#)
- size_t [params_size](#)

6.8.1 Detailed Description

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

6.8.2 Constructor & Destructor Documentation

6.8.2.1 CommandRequest()

```
zmqutils::common::CommandRequest::CommandRequest ( ) [inline]
```

Definition at line 212 of file [common.h](#).

6.8.3 Member Data Documentation

6.8.3.1 client

```
HostClient zmqutils::common::CommandRequest::client
```

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

6.8.3.2 command

```
ServerCommand zmqutils::common::CommandRequest::command
```

Definition at line 219 of file [common.h](#).

6.8.3.3 params

```
std::unique_ptr<std::uint8_t> zmqutils::common::CommandRequest::params
```

Definition at line 220 of file [common.h](#).

6.8.3.4 params_size

```
size_t zmqutils::common::CommandRequest::params_size
```

Definition at line 222 of file [common.h](#).

6.8.3.5 raw_msg

```
zmq::multipart_t zmqutils::common::CommandRequest::raw_msg
```

Definition at line 221 of file [common.h](#).

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

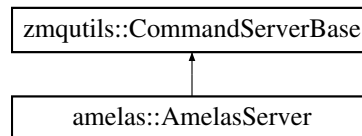
- [includes/LibZMQUtils/CommandServerClient/common.h](#)

6.9 zmqutils::CommandServerBase Class Reference

This class provides the base structure for a ZeroMQ based command server.

```
#include <command_server.h>
```

Inheritance diagram for zmqutils::CommandServerBase:



Public Member Functions

- [CommandServerBase](#) (unsigned port, const std::string &local_addr="*")
Base constructor for a ZeroMQ command server.
- const unsigned & [getServerPort](#) () const
Get the port number used by the server for incoming connections.
- const std::vector< [NetworkAdapterInfo](#) > & [getServerAddresses](#) () const
Get the network adapter addresses used by the server.
- const std::string & [getServerEndpoint](#) () const
Get the endpoint of the server.
- const std::future< void > & [getServerWorkerFuture](#) () const
Get the future associated with the server's worker thread.
- const std::map< std::string, [HostClient](#) > & [getConnectedClients](#) () const
Get a const reference to the map of connected clients.
- bool [isWorking](#) () const
Check if the server is currently working.
- void [setClientStatusCheck](#) (bool)
Enables or disables the client's alive status checking.
- void [startServer](#) ()
Starts the command server.
- void [stopServer](#) ()
Stops the command server.
- virtual [~CommandServerBase](#) ()
Virtual destructor.

Protected Member Functions

- virtual void [onServerStop](#) ()=0
Base server stop callback. Subclasses must override this function.
- virtual void [onServerStart](#) ()=0
Base server start callback. Subclasses must override this function.
- virtual void [onWaitingCommand](#) ()=0
Base waiting command callback. Subclasses must override this function.
- virtual void [onConnected](#) (const [HostClient](#) &)=0
Base connected callback. Subclasses must override this function.
- virtual void [onDisconnected](#) (const [HostClient](#) &)=0

- Base disconnected callback. Subclasses must override this function.*
- virtual void `onDeadClient` (const `HostClient` &)=0
- Base dead client callback. Subclasses must override this function.*
- virtual void `onInvalidMsgReceived` (const `CommandRequest` &)=0
- Base invalid message received callback. Subclasses must override this function.*
- virtual void `onCommandReceived` (const `CommandRequest` &)=0
- Base command received callback. Subclasses must override this function.*
- virtual void `onCustomCommandReceived` (const `CommandRequest` &, `CommandReply` &)
- Base custom command received callback. Subclasses must override this function.*
- virtual void `onServerError` (const `zmq::error_t` &error, const std::string &ext_info="")=0
- Base server error callback. Subclasses must override this function.*
- virtual void `onSendingResponse` (const `CommandReply` &)=0
- Base sending response callback. Subclasses must override this function.*

6.9.1 Detailed Description

This class provides the base structure for a ZeroMQ based command server.

The `CommandServerBase` class encapsulates the common logic and functionality for a server that communicates over the ZeroMQ messaging infrastructure. It provides the basic mechanics for starting, stopping, and managing a server, and for handling client connections, commands, and responses.

This base class is designed to be inherited by subclasses that provide specific implementations for various callback functions to handle server events such as the start/stop of the server, client connections/disconnections, receiving invalid or custom commands, and server errors. This design allows the creation of specialized servers for different use cases while keeping the core logic generic and reusable.

The server created with this class operates asynchronously, with the main server tasks running in a separate thread. It is capable of managing multiple client connections, processing command requests, and sending responses. The server also provides optional functionalities such as checking the alive status of connected clients.

Note

This class is not directly useful on its own. Instead, it is intended to be subclassed and its callback methods overridden to implement the desired server behavior.

Warning

When creating a subclass, make sure to avoid blocking or computationally intensive operations within the overridden callbacks. Blocking the server thread can affect the server's performance and responsiveness. If complex tasks are needed, consider performing them asynchronously or using separate threads.

6.9.2 Usage

To use this class, create a subclass and override the callback functions according to your needs. Also you can define the custom commands and the custom errors related with the subclass, as well as extend the containers that contains the string representation of the commands and errors.

Then, create an instance of your subclass, and use the `startServer` and `stopServer` methods to control the server's operation. You can query the server's state and information using the various getters (`getServerPort`, `getServerAddresses`, `getServerEndpoint`, `getServerWorkerFuture`, `getConnectedClients`, and `isWorking`). You can also use `setClientStatusCheck(bool)` to control the checking of clients' alive status.

A similar usage pattern applies to the `CommandClientBase` class, which is meant to interact with a `CommandServerBase` instance. `CommandClientBase` is also designed to be subclassed with callback methods to be overridden for specific client behaviors. Therefore, a typical usage scenario involves creating subclassed instances of both classes `CommandServerBase` and `CommandClientBase`, where the server handles commands sent by the client.

6.9.3 Callbacks

The following callbacks need to be overridden in your subclass:

- onServerStop
- onServerStart
- onWaitingCommand
- [onConnected\(const HostClient&\)](#)
- [onDisconnected\(const HostClient&\)](#)
- [onDeadClient\(const HostClient&\)](#)
- [onInvalidMsgReceived\(const CommandRequest&\)](#)
- [onCommandReceived\(const CommandRequest&\)](#)
- [onCustomCommandReceived\(const CommandRequest&, CommandReply&\)](#)
- onServerError(const [zmq::error_t](#) &error, const std::string& ext_info = "")
- [onSendingResponse\(const CommandReply&\)](#)

Among these callbacks, `onCustomCommandReceived` is crucial because it handles all custom commands. This function receives a command request and a command reply as parameters. It is intended to deserialize the input parameters from the command request, execute the custom command, and then serialize the output parameters into the command reply. Therefore, it is necessary to override this function to add handling for the custom commands specific to your server's application.

See also

`CommandRequest`, `CommandReply`, `HostClient`, [CommandClientBase](#), [onCustomCommandReceived](#)

Definition at line 141 of file [command_server.h](#).

6.9.4 Constructor & Destructor Documentation

6.9.4.1 CommandServerBase()

```
zmqutils::CommandServerBase::CommandServerBase (
    unsigned port,
    const std::string & local_addr = "*" )
```

Base constructor for a ZeroMQ command server.

This constructor initializes a ZeroMQ based command server with the specified port for listening to incoming requests. Additionally, it allows specifying local addresses on which the server will accept connections. By default, the server will accept connections on all available local addresses.

Parameters

<i>port</i>	The port number on which the server will listen for incoming requests.
<i>local_addr</i>	Optional parameter to specify the local addresses on which the server will accept connections. By default, it is set to "*", which means the server will accept connections on all available local
Generated by Doxygen	

Note

The server created with this constructor will be a base server and it doesn't have the complete implementation of specific request-response logic. It is intended to be subclassed to provide custom request handling. You can implement the "onCustomCommandReceived" function as an internal callback in the subclass to handle incoming requests and provide the desired response logic.

Warning

When specifying the `local_addr`, ensure it is a valid IP address present on the system. Incorrect or unavailable addresses may result in connection failures.

Definition at line 52 of file [command_server.cpp](#).

6.9.4.2 ~CommandServerBase()

```
zmqutils::CommandServerBase::~CommandServerBase ( ) [virtual]
```

Virtual destructor.

This destructor is virtual to ensure proper cleanup when the derived class is destroyed.

Definition at line 137 of file [command_server.cpp](#).

6.9.5 Member Function Documentation**6.9.5.1 getConnectedClients()**

```
const std::map< std::string, HostClient > & zmqutils::CommandServerBase::getConnectedClients (
) const
```

Get a const reference to the map of connected clients.

This function returns a const reference to a `std::map<std::string, HostClient>` representing the list of connected clients. Each entry in the map consists of a string key (client identifier) and a `HostClient` object containing information about the connected client.

Returns

A const reference to the map of connected clients.

Definition at line 77 of file [command_server.cpp](#).

6.9.5.2 getServerAddresses()

```
const std::vector< utils::NetworkAdapterInfo > & zmqutils::CommandServerBase::getServerAddresses ( ) const
```

Get the network adapter addresses used by the server.

This function returns a const reference to a vector of `NetworkAdapterInfo` objects. Each `NetworkAdapterInfo` object contains information about a network adapter used by the server for communication.

Returns

A const reference to a vector of `NetworkAdapterInfo` objects.

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

6.9.5.3 getServerEndpoint()

```
const std::string & zmqutils::CommandServerBase::getServerEndpoint ( ) const
```

Get the endpoint of the server.

This function returns a const reference to a string representing the server's endpoint. The endpoint typically includes the IP address and port number.

Returns

A const reference to the server's endpoint.

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

6.9.5.4 getServerPort()

```
const unsigned & zmqutils::CommandServerBase::getServerPort ( ) const
```

Get the port number used by the server for incoming connections.

Returns

A const reference to the port number of the server.

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

6.9.5.5 getServerWorkerFuture()

```
const std::future< void > & zmqutils::CommandServerBase::getServerWorkerFuture ( ) const
```

Get the future associated with the server's worker thread.

This function returns a const reference to a `std::future<void>` object representing the asynchronous worker thread that is running the server. The `std::future` object can be used to check the status of the worker thread or wait for it to complete.

Returns

A const reference to the server's worker thread future.

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

6.9.5.6 isWorking()

```
bool zmqutils::CommandServerBase::isWorking ( ) const [inline]
```

Check if the server is currently working.

This function returns a boolean value indicating whether the server is currently active and working. If the server is working, it means it is processing incoming connections or performing its intended tasks.

Returns

True if the server is working, false otherwise.

Definition at line 226 of file [command_server.h](#).

6.9.5.7 onCommandReceived()

```
virtual void zmqutils::CommandServerBase::onCommandReceived (
    const CommandRequest & ) [protected], [pure virtual]
```

Base command received callback. Subclasses must override this function.

Parameters

<i>The</i>	CommandRequest object representing the command execution request.
------------	---

Warning

This internal callback must be used for log or similar purposes. For specific custom command functionalities use the internal "onCustomCommandReceived".

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

6.9.5.8 onConnected()

```
virtual void zmqutils::CommandServerBase::onConnected (
    const HostClient & ) [protected], [pure virtual]
```

Base connected callback. Subclasses must override this function.

Parameters

<i>The</i>	HostClient object representing the connected client.
------------	--

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

6.9.5.9 onCustomCommandReceived()

```
void zmqutils::CommandServerBase::onCustomCommandReceived (
    const CommandRequest & ,
    CommandReply & rep ) [protected], [virtual]
```

Base custom command received callback. Subclasses must override this function.

Parameters

in	<i>The</i>	CommandRequest object representing the command execution request.
out	<i>The</i>	CommandReply object representing the command execution reply.

Note

This function must process the `CommandRequest` (function parameter input) and update the `CommandReply` (function parameter output), especially the result code.

Warning

All internal callbacks, including this one, must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

Definition at line 630 of file `command_server.cpp`.

6.9.5.10 onDeadClient()

```
virtual void zmqutils::CommandServerBase::onDeadClient (
    const HostClient & ) [protected], [pure virtual]
```

Base dead client callback. Subclasses must override this function.

Parameters

<i>The</i>	HostClient object representing the dead client.
------------	---

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

6.9.5.11 onDisconnected()

```
virtual void zmqutils::CommandServerBase::onDisconnected (
    const HostClient & ) [protected], [pure virtual]
```

Base disconnected callback. Subclasses must override this function.

Parameters

<i>The</i>	HostClient object representing the disconnected client.
------------	---

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking

the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

6.9.5.12 onInvalidMsgReceived()

```
virtual void zmqutils::CommandServerBase::onInvalidMsgReceived (
    const CommandRequest & ) [protected], [pure virtual]
```

Base invalid message received callback. Subclasses must override this function.

Parameters

<i>The</i>	CommandRequest object representing the invalid command request.
------------	---

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

6.9.5.13 onSendingResponse()

```
virtual void zmqutils::CommandServerBase::onSendingResponse (
    const CommandReply & ) [protected], [pure virtual]
```

Base sending response callback. Subclasses must override this function.

Parameters

<i>The</i>	CommandReply object representing the command reply being sent.
------------	--

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

6.9.5.14 onServerError()

```
virtual void zmqutils::CommandServerBase::onServerError (
    const zmq::error\_t & error,
    const std::string & ext_info = "" ) [protected], [pure virtual]
```

Base server error callback. Subclasses must override this function.

Parameters

<i>The</i>	<code>zmq::error_t</code> object representing the error that occurred.
<i>Optional</i>	additional information or context related to the error. It is an empty string by default.

Note

The `zmq::error_t` class provides information about ZeroMQ errors. You can access the error code, description, and other details using the methods provided by `zmq::error_t`.

Warning

If this function is not overridden in subclasses, it will not handle server errors, and errors may not be handled properly.

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

Implemented in `amelas::AmelasServer`.

6.9.5.15 onServerStart()

```
virtual void zmqutils::CommandServerBase::onServerStart ( ) [protected], [pure virtual]
```

Base server start callback. Subclasses must override this function.

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

Implemented in `amelas::AmelasServer`.

6.9.5.16 onServerStop()

```
virtual void zmqutils::CommandServerBase::onServerStop ( ) [protected], [pure virtual]
```

Base server stop callback. Subclasses must override this function.

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

Implemented in `amelas::AmelasServer`.

6.9.5.17 onWaitingCommand()

```
virtual void zmqutils::CommandServerBase::onWaitingCommand ( ) [protected], [pure virtual]
```

Base waiting command callback. Subclasses must override this function.

Note

This function is intended to be called during the server's main loop when there are no incoming requests to process. Subclasses may implement this function to perform periodic checks, cleanup tasks, or other non-blocking activities while waiting for requests.

Warning

The overridden callback must be non-blocking and have minimal computation time. Blocking or computationally intensive operations within internal callbacks can significantly affect the server's performance and responsiveness. If complex tasks are required, it is recommended to perform them asynchronously to avoid blocking the server's main thread. Consider using separate threads or asynchronous mechanisms to handle time-consuming tasks.

Implemented in [amelas::AmelasServer](#).

6.9.5.18 setClientStatusCheck()

```
void zmqutils::CommandServerBase::setClientStatusCheck (
    bool )
```

Enables or disables the client's alive status checking.

Enables or disables the checking of the client's alive status. This is a very important functionality in the context of critical systems that often use these types of servers.

Parameters

<i>The</i>	desired status of the client's alive status checking (true to enable, false to disable).
------------	--

Warning

It is strongly recommended to keep this check active, due to the critical nature of the systems that usually use this kind of servers. Disabling the client alive status check could result in unexpected behavior or system instability in case of sudden client disconnections or failures.

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

6.9.5.19 startServer()

```
void zmqutils::CommandServerBase::startServer ( )
```

Starts the command server.

If the server is already running, the function does nothing. Otherwise, it creates the ZMQ context if it doesn't exist and launches the server worker in a separate thread.

Definition at line 97 of file [command_server.cpp](#).

6.9.5.20 stopServer()

```
void zmqutils::CommandServerBase::stopServer ( )
```

Stops the command server.

If the server is already stopped, the function does nothing. Otherwise deletes the ZMQ context and cleans up the connected clients.

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

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

- includes/LibZMQUtils/CommandServerClient/[command_server.h](#)
- sources/CommandServerClient/[command_server.cpp](#)

6.10 zmq::context_t Class Reference

```
#include <zmq.hpp>
```

Public Member Functions

- [context_t](#) ()
- [context_t](#) (int io_threads_, int max_sockets_=ZMQ_MAX_SOCKETS_DFLT)
- [~context_t](#) () ZMQ_NOTHROW
- int [setctxopt](#) (int option_, int optval_)
- int [getctxopt](#) (int option_)
- void [close](#) () ZMQ_NOTHROW
- void [shutdown](#) () ZMQ_NOTHROW
- ZMQ_EXPLICIT operator void * () ZMQ_NOTHROW
- ZMQ_EXPLICIT operator void const * () const ZMQ_NOTHROW
- ZMQ_NODISCARD void * [handle](#) () ZMQ_NOTHROW
- operator bool () const ZMQ_NOTHROW
- void [swap](#) (context_t &other) ZMQ_NOTHROW

6.10.1 Detailed Description

Definition at line 798 of file [zmq.hpp](#).

6.10.2 Constructor & Destructor Documentation

6.10.2.1 context_t() [1/2]

```
zmq::context_t::context_t ( ) [inline]
```

Definition at line 801 of file [zmq.hpp](#).

6.10.2.2 context_t() [2/2]

```
zmq::context_t::context_t (
    int io_threads_,
    int max_sockets_ = ZMQ_MAX_SOCKETS_DFLT ) [inline], [explicit]
```

Definition at line 809 of file [zmq.hpp](#).

6.10.2.3 ~context_t()

```
zmq::context_t::~~context_t ( ) [inline]
```

Definition at line 832 of file [zmq.hpp](#).

6.10.3 Member Function Documentation

6.10.3.1 close()

```
void zmq::context_t::close ( ) [inline]
```

Definition at line 866 of file [zmq.hpp](#).

6.10.3.2 getctxopt()

```
int zmq::context_t::getctxopt (
    int option_ ) [inline]
```

Definition at line 843 of file [zmq.hpp](#).

6.10.3.3 handle()

```
ZMQ_NODISCARD void * zmq::context_t::handle ( ) [inline]
```

Definition at line 898 of file [zmq.hpp](#).

6.10.3.4 operator bool()

```
zmq::context_t::operator bool ( ) const [inline]
```

Definition at line 901 of file [zmq.hpp](#).

6.10.3.5 operator void *()

```
ZMQ_EXPLICIT zmq::context_t::operator void * ( ) [inline]
```

Definition at line 894 of file [zmq.hpp](#).

6.10.3.6 operator void const *()

```
ZMQ_EXPLICIT zmq::context_t::operator void const * ( ) const [inline]
```

Definition at line 896 of file [zmq.hpp](#).

6.10.3.7 setctxopt()

```
int zmq::context_t::setctxopt (
    int option_,
    int optval_ ) [inline]
```

Definition at line 835 of file [zmq.hpp](#).

6.10.3.8 shutdown()

```
void zmq::context_t::shutdown ( ) [inline]
```

Definition at line 883 of file [zmq.hpp](#).

6.10.3.9 swap()

```
void zmq::context_t::swap (
    context_t & other ) [inline]
```

Definition at line 903 of file [zmq.hpp](#).

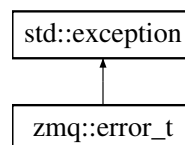
The documentation for this class was generated from the following file:

- [external/zmq/includes/zmq/zmq.hpp](#)

6.11 zmq::error_t Class Reference

```
#include <zmq.hpp>
```

Inheritance diagram for `zmq::error_t`:



Public Member Functions

- [error_t \(\) ZMQ_NOTHROW](#)
- [error_t \(int err\) ZMQ_NOTHROW](#)
- [virtual const char * what \(\) const ZMQ_NOTHROW ZMQ_OVERRIDE](#)
- [int num \(\) const ZMQ_NOTHROW](#)

6.11.1 Detailed Description

Definition at line [289](#) of file [zmq.hpp](#).

6.11.2 Constructor & Destructor Documentation

6.11.2.1 `error_t()` [1/2]

```
zmq::error_t::error_t ( ) [inline]
```

Definition at line [292](#) of file [zmq.hpp](#).

6.11.2.2 `error_t()` [2/2]

```
zmq::error_t::error_t (
    int err ) [inline], [explicit]
```

Definition at line [293](#) of file [zmq.hpp](#).

6.11.3 Member Function Documentation

6.11.3.1 `num()`

```
int zmq::error_t::num ( ) const [inline]
```

Definition at line [298](#) of file [zmq.hpp](#).

6.11.3.2 `what()`

```
virtual const char * zmq::error_t::what ( ) const [inline], [virtual]
```

Definition at line [294](#) of file [zmq.hpp](#).

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

- [external/zmq/includes/zmq/zmq.hpp](#)

6.12 `zmq::from_handle_t` Struct Reference

```
#include <zmq.hpp>
```

Classes

- [struct `_private`](#)

Public Member Functions

- [ZMQ_CONSTEXPR_FN ZMQ_EXPLICIT from_handle_t \(_private\) ZMQ_NOTHROW](#)

6.12.1 Detailed Description

Definition at line 2094 of file [zmq.hpp](#).

6.12.2 Constructor & Destructor Documentation

6.12.2.1 from_handle_t()

```
ZMQ_CONSTEXPR_FN ZMQ_EXPLICIT zmq::from_handle_t::from_handle_t (
    _private ) [inline]
```

Definition at line 2099 of file [zmq.hpp](#).

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

- [external/zmq/includes/zmq/zmq.hpp](#)

6.13 zmqutils::common::HostClient Struct Reference

```
#include <common.h>
```

Public Member Functions

- [HostClient \(\)](#)=default
- [HostClient \(const HostClient &\)](#)=default
- [HostClient \(HostClient &&\)](#)=default
- [HostClient & operator= \(const HostClient &\)](#)=default
- [HostClient & operator= \(HostClient &&\)](#)=default
- [HostClient \(const std::string &ip, const std::string &name, const std::string &pid, const std::string &info= ""\)](#)

Public Attributes

- [std::string id](#)
Dynamic host client identification -> [ip//name//pid].
- [std::string ip](#)
Host client ip.
- [std::string hostname](#)
Host client name.
- [std::string pid](#)
PID of the host client process.
- [std::string info](#)
Host client information.
- [utils::SCTimePointStd last_connection](#)
Host client last connection time.

6.13.1 Detailed Description

Definition at line 186 of file [common.h](#).

6.13.2 Constructor & Destructor Documentation

6.13.2.1 HostClient() [1/4]

```
zmqutils::common::HostClient::HostClient ( ) [default]
```

6.13.2.2 HostClient() [2/4]

```
zmqutils::common::HostClient::HostClient (
    const HostClient & ) [default]
```

6.13.2.3 HostClient() [3/4]

```
zmqutils::common::HostClient::HostClient (
    HostClient && ) [default]
```

6.13.2.4 HostClient() [4/4]

```
zmqutils::common::HostClient::HostClient (
    const std::string & ip,
    const std::string & name,
    const std::string & pid,
    const std::string & info = "" )
```

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

6.13.3 Member Function Documentation

6.13.3.1 operator=() [1/2]

```
HostClient & zmqutils::common::HostClient::operator= (
    const HostClient & ) [default]
```

6.13.3.2 operator=() [2/2]

```
HostClient & zmqutils::common::HostClient::operator= (
    HostClient && ) [default]
```

6.13.4 Member Data Documentation

6.13.4.1 hostname

`std::string zmqutils::common::HostClient::hostname`

Host client name.

Definition at line 204 of file [common.h](#).

6.13.4.2 id

`std::string zmqutils::common::HostClient::id`

Dynamic host client identification -> [ip//name//pid].

Definition at line 202 of file [common.h](#).

6.13.4.3 info

`std::string zmqutils::common::HostClient::info`

Host client information.

Definition at line 206 of file [common.h](#).

6.13.4.4 ip

`std::string zmqutils::common::HostClient::ip`

Host client ip.

Definition at line 203 of file [common.h](#).

6.13.4.5 last_connection

`utils::SCTimePointStd zmqutils::common::HostClient::last_connection`

Host client last connection time.

Definition at line 207 of file [common.h](#).

6.13.4.6 pid

`std::string zmqutils::common::HostClient::pid`

PID of the host client process.

Definition at line 205 of file [common.h](#).

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

- [includes/LibZMQUtils/CommandServerClient/common.h](#)
- [sources/CommandServerClient/common.cpp](#)

6.14 zmq::message_t Class Reference

```
#include <zmq.hpp>
```

Public Member Functions

- [message_t \(\) ZMQ_NOTHROW](#)
- [message_t \(size_t size_\)](#)
- [template<class ForwardIter >
message_t \(ForwardIter first, ForwardIter last\)](#)
- [message_t \(const void *data_, size_t size_\)](#)
- [message_t \(void *data_, size_t size_, free_fn *ffn_, void *hint_=ZMQ_NULLPTR\)](#)
- [~message_t \(\) ZMQ_NOTHROW](#)
- [void rebuild \(\)](#)
- [void rebuild \(size_t size_\)](#)
- [void rebuild \(const void *data_, size_t size_\)](#)
- [void rebuild \(const std::string &str\)](#)
- [void rebuild \(void *data_, size_t size_, free_fn *ffn_, void *hint_=ZMQ_NULLPTR\)](#)
- [void move \(message_t const &msg_\)](#)
- [void move \(message_t &msg_\)](#)
- [void copy \(message_t const &msg_\)](#)
- [void copy \(message_t &msg_\)](#)
- [bool more \(\) const ZMQ_NOTHROW](#)
- [void * data \(\) ZMQ_NOTHROW](#)
- [const void * data \(\) const ZMQ_NOTHROW](#)
- [size_t size \(\) const ZMQ_NOTHROW](#)
- [ZMQ_NODISCARD bool empty \(\) const ZMQ_NOTHROW](#)
- [template<typename T >
T * data \(\) ZMQ_NOTHROW](#)
- [template<typename T >
T const * data \(\) const ZMQ_NOTHROW](#)
- [bool equal \(const message_t *other\) const ZMQ_NOTHROW](#)
- [bool operator== \(const message_t &other\) const ZMQ_NOTHROW](#)
- [bool operator!= \(const message_t &other\) const ZMQ_NOTHROW](#)
- [int get \(int property_\)](#)
- [const char * gets \(const char *property_\)](#)
- [std::string to_string \(\) const](#)
- [std::string str \(\) const](#)
- [void swap \(message_t &other\) ZMQ_NOTHROW](#)
- [ZMQ_NODISCARD zmq_msg_t * handle \(\) ZMQ_NOTHROW](#)
- [ZMQ_NODISCARD const zmq_msg_t * handle \(\) const ZMQ_NOTHROW](#)

6.14.1 Detailed Description

Definition at line 408 of file [zmq.hpp](#).

6.14.2 Constructor & Destructor Documentation

6.14.2.1 message_t() [1/5]

```
zmq::message_t::message_t ( ) [inline]
```

Definition at line 411 of file [zmq.hpp](#).

6.14.2.2 message_t() [2/5]

```
zmq::message_t::message_t (  
    size_t size_ ) [inline], [explicit]
```

Definition at line 417 of file [zmq.hpp](#).

6.14.2.3 message_t() [3/5]

```
template<class ForwardIter >  
zmq::message_t::message_t (  
    ForwardIter first,  
    ForwardIter last ) [inline]
```

Definition at line 424 of file [zmq.hpp](#).

6.14.2.4 message_t() [4/5]

```
zmq::message_t::message_t (  
    const void * data_,  
    size_t size_ ) [inline]
```

Definition at line 437 of file [zmq.hpp](#).

6.14.2.5 message_t() [5/5]

```
zmq::message_t::message_t (  
    void * data_,  
    size_t size_,  
    free_fn * ffn_,  
    void * hint_ = ZMQ_NULLPTR ) [inline]
```

Definition at line 449 of file [zmq.hpp](#).

6.14.2.6 ~message_t()

```
zmq::message_t::~~message_t ( ) [inline]
```

Definition at line 506 of file [zmq.hpp](#).

6.14.3 Member Function Documentation

6.14.3.1 copy() [1/2]

```
void zmq::message_t::copy (  
    message_t & msg_ ) [inline]
```

Definition at line 580 of file [zmq.hpp](#).

6.14.3.2 copy() [2/2]

```
void zmq::message_t::copy (
    message_t const * msg_ ) [inline]
```

Definition at line 573 of file [zmq.hpp](#).

6.14.3.3 data() [1/4]

```
const void * zmq::message_t::data ( ) const [inline]
```

Definition at line 595 of file [zmq.hpp](#).

6.14.3.4 data() [2/4]

```
template<typename T >
T const * zmq::message_t::data ( ) const [inline]
```

Definition at line 609 of file [zmq.hpp](#).

6.14.3.5 data() [3/4]

```
void * zmq::message_t::data ( ) [inline]
```

Definition at line 593 of file [zmq.hpp](#).

6.14.3.6 data() [4/4]

```
template<typename T >
T * zmq::message_t::data ( ) [inline]
```

Definition at line 607 of file [zmq.hpp](#).

6.14.3.7 empty()

```
ZMQ_NODISCARD bool zmq::message_t::empty ( ) const [inline]
```

Definition at line 605 of file [zmq.hpp](#).

6.14.3.8 equal()

```
bool zmq::message_t::equal (
    const message_t * other ) const [inline]
```

Definition at line 615 of file [zmq.hpp](#).

6.14.3.9 get()

```
int zmq::message_t::get (
    int property_ ) [inline]
```

Definition at line 629 of file [zmq.hpp](#).

6.14.3.10 gets()

```
const char * zmq::message_t::gets (
    const char * property_ ) [inline]
```

Definition at line 639 of file [zmq.hpp](#).

6.14.3.11 handle() [1/2]

```
ZMQ_NODISCARD const zmq_msg_t * zmq::message_t::handle ( ) const [inline]
```

Definition at line 736 of file [zmq.hpp](#).

6.14.3.12 handle() [2/2]

```
ZMQ_NODISCARD zmq_msg_t * zmq::message_t::handle ( ) [inline]
```

Definition at line 735 of file [zmq.hpp](#).

6.14.3.13 more()

```
bool zmq::message_t::more ( ) const [inline]
```

Definition at line 587 of file [zmq.hpp](#).

6.14.3.14 move() [1/2]

```
void zmq::message_t::move (
    message_t & msg_ ) [inline]
```

Definition at line 565 of file [zmq.hpp](#).

6.14.3.15 move() [2/2]

```
void zmq::message_t::move (
    message_t const * msg_ ) [inline]
```

Definition at line 558 of file [zmq.hpp](#).

6.14.3.16 operator"!=()

```
bool zmq::message_t::operator!= (
    const message\_t & other ) const [inline]
```

Definition at line 623 of file [zmq.hpp](#).

6.14.3.17 operator==(

```
bool zmq::message_t::operator== (
    const message\_t & other ) const [inline]
```

Definition at line 617 of file [zmq.hpp](#).

6.14.3.18 rebuild() [1/5]

```
void zmq::message_t::rebuild ( ) [inline]
```

Definition at line 512 of file [zmq.hpp](#).

6.14.3.19 rebuild() [2/5]

```
void zmq::message_t::rebuild (
    const std::string & str ) [inline]
```

Definition at line 542 of file [zmq.hpp](#).

6.14.3.20 rebuild() [3/5]

```
void zmq::message_t::rebuild (
    const void * data_,
    size_t size_ ) [inline]
```

Definition at line 531 of file [zmq.hpp](#).

6.14.3.21 rebuild() [4/5]

```
void zmq::message_t::rebuild (
    size_t size_ ) [inline]
```

Definition at line 521 of file [zmq.hpp](#).

6.14.3.22 rebuild() [5/5]

```
void zmq::message_t::rebuild (
    void * data_,
    size_t size_,
    free\_fn * ffn_,
    void * hint_ = ZMQ\_NULLPTR ) [inline]
```

Definition at line 547 of file [zmq.hpp](#).

6.14.3.23 size()

```
size_t zmq::message_t::size ( ) const [inline]
```

Definition at line 600 of file [zmq.hpp](#).

6.14.3.24 str()

```
std::string zmq::message_t::str ( ) const [inline]
```

Dump content to string for debugging. Ascii chars are readable, the rest is printed as hex. Probably ridiculously slow. Use [to_string\(\)](#) or [to_string_view\(\)](#) for interpreting the message as a string.

Definition at line 693 of file [zmq.hpp](#).

6.14.3.25 swap()

```
void zmq::message_t::swap (
    message_t & other ) [inline]
```

Definition at line 729 of file [zmq.hpp](#).

6.14.3.26 to_string()

```
std::string zmq::message_t::to_string ( ) const [inline]
```

Definition at line 675 of file [zmq.hpp](#).

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

- [external/zmq/includes/zmq/zmq.hpp](#)

6.15 zmq::monitor_t Class Reference

```
#include <zmq.hpp>
```

Public Member Functions

- [monitor_t](#) ()
- virtual [~monitor_t](#) ()
- void [monitor](#) ([socket_t](#) &socket, std::string const &addr, int events=[ZMQ_EVENT_ALL](#))
- void [monitor](#) ([socket_t](#) &socket, const char *addr_, int events=[ZMQ_EVENT_ALL](#))
- void [init](#) ([socket_t](#) &socket, std::string const &addr, int events=[ZMQ_EVENT_ALL](#))
- void [init](#) ([socket_t](#) &socket, const char *addr_, int events=[ZMQ_EVENT_ALL](#))
- bool [check_event](#) (int timeout=0)
- void [abort](#) ()
- virtual void [on_monitor_started](#) ()
- virtual void [on_event_connected](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_connect_delayed](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_connect_retried](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_listening](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_bind_failed](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_accepted](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_accept_failed](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_closed](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_close_failed](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_disconnected](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_handshake_failed_no_detail](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_handshake_failed_protocol](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_handshake_failed_auth](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_handshake_succeeded](#) (const [zmq_event_t](#) &event_, const char *addr_)
- virtual void [on_event_unknown](#) (const [zmq_event_t](#) &event_, const char *addr_)

6.15.1 Detailed Description

Definition at line [2304](#) of file [zmq.hpp](#).

6.15.2 Constructor & Destructor Documentation

6.15.2.1 [monitor_t](#)()

```
zmq::monitor_t::monitor_t ( ) [inline]
```

Definition at line [2307](#) of file [zmq.hpp](#).

6.15.2.2 [~monitor_t](#)()

```
virtual zmq::monitor_t::~~monitor_t ( ) [inline], [virtual]
```

Definition at line [2309](#) of file [zmq.hpp](#).

6.15.3 Member Function Documentation

6.15.3.1 [abort](#)()

```
void zmq::monitor_t::abort ( ) [inline]
```

Definition at line [2480](#) of file [zmq.hpp](#).

6.15.3.2 check_event()

```
bool zmq::monitor_t::check_event (
    int timeout = 0 ) [inline]
```

Definition at line 2361 of file [zmq.hpp](#).

6.15.3.3 init() [1/2]

```
void zmq::monitor_t::init (
    socket_t & socket,
    const char * addr_,
    int events = ZMQ_EVENT_ALL ) [inline]
```

Definition at line 2348 of file [zmq.hpp](#).

6.15.3.4 init() [2/2]

```
void zmq::monitor_t::init (
    socket_t & socket,
    std::string const & addr_,
    int events = ZMQ_EVENT_ALL ) [inline]
```

Definition at line 2343 of file [zmq.hpp](#).

6.15.3.5 monitor() [1/2]

```
void zmq::monitor_t::monitor (
    socket_t & socket,
    const char * addr_,
    int events = ZMQ_EVENT_ALL ) [inline]
```

Definition at line 2335 of file [zmq.hpp](#).

6.15.3.6 monitor() [2/2]

```
void zmq::monitor_t::monitor (
    socket_t & socket,
    std::string const & addr_,
    int events = ZMQ_EVENT_ALL ) [inline]
```

Definition at line 2330 of file [zmq.hpp](#).

6.15.3.7 on_event_accept_failed()

```
virtual void zmq::monitor_t::on_event_accept_failed (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2521 of file [zmq.hpp](#).

6.15.3.8 on_event_accepted()

```
virtual void zmq::monitor_t::on_event_accepted (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2516 of file [zmq.hpp](#).

6.15.3.9 on_event_bind_failed()

```
virtual void zmq::monitor_t::on_event_bind_failed (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2511 of file [zmq.hpp](#).

6.15.3.10 on_event_close_failed()

```
virtual void zmq::monitor_t::on_event_close_failed (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2531 of file [zmq.hpp](#).

6.15.3.11 on_event_closed()

```
virtual void zmq::monitor_t::on_event_closed (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2526 of file [zmq.hpp](#).

6.15.3.12 on_event_connect_delayed()

```
virtual void zmq::monitor_t::on_event_connect_delayed (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2494 of file [zmq.hpp](#).

6.15.3.13 on_event_connect_retried()

```
virtual void zmq::monitor_t::on_event_connect_retried (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2500 of file [zmq.hpp](#).

6.15.3.14 on_event_connected()

```
virtual void zmq::monitor_t::on_event_connected (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2489 of file [zmq.hpp](#).

6.15.3.15 on_event_disconnected()

```
virtual void zmq::monitor_t::on_event_disconnected (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2536 of file [zmq.hpp](#).

6.15.3.16 on_event_handshake_failed_auth()

```
virtual void zmq::monitor_t::on_event_handshake_failed_auth (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2554 of file [zmq.hpp](#).

6.15.3.17 on_event_handshake_failed_no_detail()

```
virtual void zmq::monitor_t::on_event_handshake_failed_no_detail (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2542 of file [zmq.hpp](#).

6.15.3.18 on_event_handshake_failed_protocol()

```
virtual void zmq::monitor_t::on_event_handshake_failed_protocol (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2548 of file [zmq.hpp](#).

6.15.3.19 on_event_handshake_succeeded()

```
virtual void zmq::monitor_t::on_event_handshake_succeeded (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2560 of file [zmq.hpp](#).

6.15.3.20 on_event_listening()

```
virtual void zmq::monitor_t::on_event_listening (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2506 of file [zmq.hpp](#).

6.15.3.21 on_event_unknown()

```
virtual void zmq::monitor_t::on_event_unknown (
    const zmq_event_t & event_,
    const char * addr_ ) [inline], [virtual]
```

Definition at line 2580 of file [zmq.hpp](#).

6.15.3.22 on_monitor_started()

```
virtual void zmq::monitor_t::on_monitor_started ( ) [inline], [virtual]
```

Definition at line 2488 of file [zmq.hpp](#).

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

- [external/zmq/includes/zmq/zmq.hpp](#)

6.16 zmqutils::utils::NetworkAdapterInfo Struct Reference

```
#include <utils.h>
```

Public Attributes

- `std::string` [id](#)
- `std::string` [name](#)
- `std::string` [descr](#)
- `std::string` [ip](#)

6.16.1 Detailed Description

Definition at line 78 of file [utils.h](#).

6.16.2 Member Data Documentation

6.16.2.1 descr

```
std::string zmqutils::utils::NetworkAdapterInfo::descr
```

Definition at line 82 of file [utils.h](#).

6.16.2.2 id

```
std::string zmqutils::utils::NetworkAdapterInfo::id
```

Definition at line 80 of file [utils.h](#).

6.16.2.3 ip

```
std::string zmqutils::utils::NetworkAdapterInfo::ip
```

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

6.16.2.4 name

```
std::string zmqutils::utils::NetworkAdapterInfo::name
```

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

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

- [includes/LibZMQUtils/utils.h](#)

6.17 zmqutils::common::RequestData Struct Reference

```
#include <common.h>
```

Public Member Functions

- [RequestData](#) ([CommandType](#) id)
- [RequestData](#) ()

Public Attributes

- [CommandType](#) [command](#)
- [std::unique_ptr< std::uint8_t >](#) [params](#)
- [size_t](#) [params_size](#)

6.17.1 Detailed Description

Definition at line 239 of file [common.h](#).

6.17.2 Constructor & Destructor Documentation

6.17.2.1 RequestData() [1/2]

```
zmqutils::common::RequestData::RequestData (
    CommandType id ) [inline]
```

Definition at line 241 of file [common.h](#).

6.17.2.2 RequestData() [2/2]

```
zmqutils::common::RequestData::RequestData ( ) [inline]
```

Definition at line 246 of file [common.h](#).

6.17.3 Member Data Documentation

6.17.3.1 command

```
CommandType zmqutils::common::RequestData::command
```

Definition at line 251 of file [common.h](#).

6.17.3.2 params

```
std::unique_ptr<std::uint8_t> zmqutils::common::RequestData::params
```

Definition at line 252 of file [common.h](#).

6.17.3.3 params_size

```
size_t zmqutils::common::RequestData::params_size
```

Definition at line 253 of file [common.h](#).

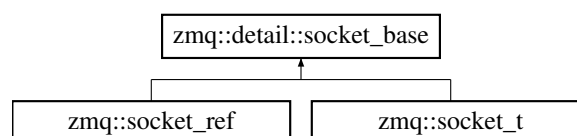
The documentation for this struct was generated from the following file:

- [includes/LibZMQUtils/CommandServerClient/common.h](#)

6.18 zmq::detail::socket_base Class Reference

```
#include <zmq.hpp>
```

Inheritance diagram for zmq::detail::socket_base:



Public Member Functions

- [socket_base\(\)](#) [ZMQ_NOTHROW](#)
- [ZMQ_EXPLICIT socket_base](#) (void *handle) [ZMQ_NOTHROW](#)
- [template<typename T > void setsockopt](#) (int option_, T const &optval)
- [void setsockopt](#) (int option_, const void *optval_, size_t optvallen_)
- [void getsockopt](#) (int option_, void *optval_, size_t *optvallen_) const
- [template<typename T > T getsockopt](#) (int option_) const
- [void bind](#) (std::string const &addr)
- [void bind](#) (const char *addr_)
- [void unbind](#) (std::string const &addr)
- [void unbind](#) (const char *addr_)
- [void connect](#) (std::string const &addr)
- [void connect](#) (const char *addr_)
- [void disconnect](#) (std::string const &addr)
- [void disconnect](#) (const char *addr_)
- [bool connected](#) () const [ZMQ_NOTHROW](#)
- [size_t send](#) (const void *buf_, size_t len_, int flags_=0)
- [bool send](#) ([message_t](#) &msg_, int flags_=0)
- [template<typename T > ZMQ_CPP11_DEPRECATED](#) ("from 4.4.1, use [send](#) taking [message_t](#) or buffer (for contiguous " "ranges), and send_flags") [bool send](#)(T first

Public Attributes

- [T last](#)
- [T int flags_](#)

6.18.1 Detailed Description

Definition at line 1727 of file [zmq.hpp](#).

6.18.2 Constructor & Destructor Documentation

6.18.2.1 socket_base() [1/2]

```
zmq::detail::socket_base::socket_base ( ) [inline]
```

Definition at line 1730 of file [zmq.hpp](#).

6.18.2.2 socket_base() [2/2]

```
ZMQ\_EXPLICIT zmq::detail::socket_base::socket_base (
    void * handle ) [inline]
```

Definition at line 1731 of file [zmq.hpp](#).

6.18.3 Member Function Documentation

6.18.3.1 bind() [1/2]

```
void zmq::detail::socket_base::bind (
    const char * addr_ ) [inline]
```

Definition at line 1878 of file [zmq.hpp](#).

6.18.3.2 bind() [2/2]

```
void zmq::detail::socket_base::bind (
    std::string const & addr ) [inline]
```

Definition at line 1876 of file [zmq.hpp](#).

6.18.3.3 connect() [1/2]

```
void zmq::detail::socket_base::connect (
    const char * addr_ ) [inline]
```

Definition at line 1896 of file [zmq.hpp](#).

6.18.3.4 connect() [2/2]

```
void zmq::detail::socket_base::connect (
    std::string const & addr ) [inline]
```

Definition at line 1894 of file [zmq.hpp](#).

6.18.3.5 connected()

```
bool zmq::detail::socket_base::connected ( ) const [inline]
```

Definition at line 1913 of file [zmq.hpp](#).

6.18.3.6 disconnect() [1/2]

```
void zmq::detail::socket_base::disconnect (
    const char * addr_ ) [inline]
```

Definition at line 1905 of file [zmq.hpp](#).

6.18.3.7 disconnect() [2/2]

```
void zmq::detail::socket_base::disconnect (
    std::string const & addr ) [inline]
```

Definition at line 1903 of file [zmq.hpp](#).

6.18.3.8 getsockopt() [1/2]

```
template<typename T >
T zmq::detail::socket_base::getsockopt (
    int option_ ) const [inline]
```

Definition at line 1758 of file [zmq.hpp](#).

6.18.3.9 getsockopt() [2/2]

```
void zmq::detail::socket_base::getsockopt (
    int option_,
    void * optval_,
    size_t * optvallen_ ) const [inline]
```

Definition at line 1749 of file [zmq.hpp](#).

6.18.3.10 send() [1/2]

```
size_t zmq::detail::socket_base::send (
    const void * buf_,
    size_t len_,
    int flags_ = 0 ) [inline]
```

Definition at line 1916 of file [zmq.hpp](#).

6.18.3.11 send() [2/2]

```
bool zmq::detail::socket_base::send (
    message_t & msg_,
    int flags_ = 0 ) [inline]
```

Definition at line 1927 of file [zmq.hpp](#).

6.18.3.12 setsockopt() [1/2]

```
void zmq::detail::socket_base::setsockopt (
    int option_,
    const void * optval_,
    size_t optvallen_ ) [inline]
```

Definition at line 1741 of file [zmq.hpp](#).

6.18.3.13 setsockopt() [2/2]

```
template<typename T >
void zmq::detail::socket_base::setsockopt (
    int option_,
    T const & optval ) [inline]
```

Definition at line 1735 of file [zmq.hpp](#).

6.18.3.14 unbind() [1/2]

```
void zmq::detail::socket_base::unbind (
    const char * addr_ ) [inline]
```

Definition at line 1887 of file [zmq.hpp](#).

6.18.3.15 unbind() [2/2]

```
void zmq::detail::socket_base::unbind (
    std::string const & addr ) [inline]
```

Definition at line 1885 of file [zmq.hpp](#).

6.18.3.16 ZMQ_CPP11_DEPRECATED()

```
template<typename T >
zmq::detail::socket_base::ZMQ_CPP11_DEPRECATED (
    "from 4.4. 1,
    use send taking message\_t or buffer for contiguous " "ranges,
    and send\_flags" )
```

6.18.4 Member Data Documentation**6.18.4.1 flags_**

```
T int zmq::detail::socket_base::flags_
```

Definition at line 1942 of file [zmq.hpp](#).

6.18.4.2 last

```
T zmq::detail::socket_base::last
```

Definition at line 1942 of file [zmq.hpp](#).

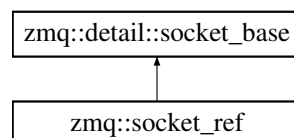
The documentation for this class was generated from the following file:

- [external/zmq/includes/zmq/zmq.hpp](#)

6.19 zmq::socket_ref Class Reference

```
#include <zmq.hpp>
```

Inheritance diagram for `zmq::socket_ref`:



Public Member Functions

- [socket_ref](#) () [ZMQ_NOTHROW](#)
- [socket_ref](#) ([from_handle_t](#), void *handle) [ZMQ_NOTHROW](#)
- `template<typename T >`
void [setsockopt](#) (int option_, T const &optval)
- void [setsockopt](#) (int option_, const void *optval_, size_t optvallen_)
- void [getsockopt](#) (int option_, void *optval_, size_t *optvallen_) const
- `template<typename T >`
T [getsockopt](#) (int option_) const
- void [bind](#) (std::string const &addr)
- void [bind](#) (const char *addr_)
- void [unbind](#) (std::string const &addr)
- void [unbind](#) (const char *addr_)
- void [connect](#) (std::string const &addr)
- void [connect](#) (const char *addr_)
- void [disconnect](#) (std::string const &addr)
- void [disconnect](#) (const char *addr_)
- bool [connected](#) () const [ZMQ_NOTHROW](#)
- size_t [send](#) (const void *buf_, size_t len_, int flags_=0)
- bool [send](#) ([message_t](#) &msg_, int flags_=0)
- `template<typename T >`
[ZMQ_CPP11_DEPRECATED](#) ("from 4.4.1, use [send](#) taking [message_t](#) or buffer (for contiguous " "ranges), and send_flags") bool [send](#)(T first

Public Attributes

- T [last](#)
- T int [flags_](#)

6.19.1 Detailed Description

Definition at line 2107 of file [zmq.hpp](#).

6.19.2 Constructor & Destructor Documentation

6.19.2.1 [socket_ref](#)() [1/2]

```
zmq::socket_ref::socket_ref ( ) [inline]
```

Definition at line 2110 of file [zmq.hpp](#).

6.19.2.2 [socket_ref](#)() [2/2]

```
zmq::socket_ref::socket_ref (
    from\_handle\_t ,
    void * handle ) [inline]
```

Definition at line 2114 of file [zmq.hpp](#).

6.19.3 Member Function Documentation

6.19.3.1 bind() [1/2]

```
void zmq::detail::socket_base::bind (
    const char * addr_ ) [inline], [inherited]
```

Definition at line 1878 of file [zmq.hpp](#).

6.19.3.2 bind() [2/2]

```
void zmq::detail::socket_base::bind (
    std::string const & addr ) [inline], [inherited]
```

Definition at line 1876 of file [zmq.hpp](#).

6.19.3.3 connect() [1/2]

```
void zmq::detail::socket_base::connect (
    const char * addr_ ) [inline], [inherited]
```

Definition at line 1896 of file [zmq.hpp](#).

6.19.3.4 connect() [2/2]

```
void zmq::detail::socket_base::connect (
    std::string const & addr ) [inline], [inherited]
```

Definition at line 1894 of file [zmq.hpp](#).

6.19.3.5 connected()

```
bool zmq::detail::socket_base::connected ( ) const [inline], [inherited]
```

Definition at line 1913 of file [zmq.hpp](#).

6.19.3.6 disconnect() [1/2]

```
void zmq::detail::socket_base::disconnect (
    const char * addr_ ) [inline], [inherited]
```

Definition at line 1905 of file [zmq.hpp](#).

6.19.3.7 disconnect() [2/2]

```
void zmq::detail::socket_base::disconnect (
    std::string const & addr ) [inline], [inherited]
```

Definition at line 1903 of file [zmq.hpp](#).

6.19.3.8 getsockopt() [1/2]

```
template<typename T >
T zmq::detail::socket_base::getsockopt (
    int option_ ) const [inline], [inherited]
```

Definition at line 1758 of file [zmq.hpp](#).

6.19.3.9 getsockopt() [2/2]

```
void zmq::detail::socket_base::getsockopt (
    int option_,
    void * optval_,
    size_t * optvallen_ ) const [inline], [inherited]
```

Definition at line 1749 of file [zmq.hpp](#).

6.19.3.10 send() [1/2]

```
size_t zmq::detail::socket_base::send (
    const void * buf_,
    size_t len_,
    int flags_ = 0 ) [inline], [inherited]
```

Definition at line 1916 of file [zmq.hpp](#).

6.19.3.11 send() [2/2]

```
bool zmq::detail::socket_base::send (
    message_t & msg_,
    int flags_ = 0 ) [inline], [inherited]
```

Definition at line 1927 of file [zmq.hpp](#).

6.19.3.12 setsockopt() [1/2]

```
void zmq::detail::socket_base::setsockopt (
    int option_,
    const void * optval_,
    size_t optvallen_ ) [inline], [inherited]
```

Definition at line 1741 of file [zmq.hpp](#).

6.19.3.13 setsockopt() [2/2]

```
template<typename T >
void zmq::detail::socket_base::setsockopt (
    int option_,
    T const & optval ) [inline], [inherited]
```

Definition at line 1735 of file [zmq.hpp](#).

6.19.3.14 unbind() [1/2]

```
void zmq::detail::socket_base::unbind (
    const char * addr_ ) [inline], [inherited]
```

Definition at line 1887 of file [zmq.hpp](#).

6.19.3.15 unbind() [2/2]

```
void zmq::detail::socket_base::unbind (
    std::string const & addr ) [inline], [inherited]
```

Definition at line 1885 of file [zmq.hpp](#).

6.19.3.16 ZMQ_CPP11_DEPRECATED()

```
template<typename T >
zmq::detail::socket_base::ZMQ_CPP11_DEPRECATED (
    "from 4.4. 1,
    use send taking message\_t or buffer for contiguous " "ranges,
    and send\_flags" ) [inherited]
```

6.19.4 Member Data Documentation**6.19.4.1 flags_**

```
T int zmq::detail::socket_base::flags_ [inherited]
```

Definition at line 1942 of file [zmq.hpp](#).

6.19.4.2 last

```
T zmq::detail::socket_base::last [inherited]
```

Definition at line 1942 of file [zmq.hpp](#).

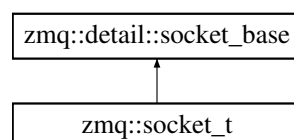
The documentation for this class was generated from the following file:

- [external/zmq/includes/zmq/zmq.hpp](#)

6.20 zmq::socket_t Class Reference

```
#include <zmq.hpp>
```

Inheritance diagram for `zmq::socket_t`:



Public Member Functions

- [socket_t\(\)](#) [ZMQ_NOTHROW](#)
- [socket_t\(context_t &context_, int type_\)](#)
- [~socket_t\(\)](#) [ZMQ_NOTHROW](#)
- [operator void *](#) [\(\)](#) [ZMQ_NOTHROW](#)
- [operator void const *](#) [\(\)](#) [const](#) [ZMQ_NOTHROW](#)
- [void close\(\)](#) [ZMQ_NOTHROW](#)
- [void swap\(socket_t &other\)](#) [ZMQ_NOTHROW](#)
- [operator socket_ref\(\)](#) [ZMQ_NOTHROW](#)
- [template<typename T > void setsockopt\(int option_, T const &optval\)](#)
- [void setsockopt\(int option_, const void *optval_, size_t optvallen_\)](#)
- [void getsockopt\(int option_, void *optval_, size_t *optvallen_\) const](#)
- [template<typename T > T getsockopt\(int option_\) const](#)
- [void bind\(std::string const &addr\)](#)
- [void bind\(const char *addr_\)](#)
- [void unbind\(std::string const &addr\)](#)
- [void unbind\(const char *addr_\)](#)
- [void connect\(std::string const &addr\)](#)
- [void connect\(const char *addr_\)](#)
- [void disconnect\(std::string const &addr\)](#)
- [void disconnect\(const char *addr_\)](#)
- [bool connected\(\)](#) [const](#) [ZMQ_NOTHROW](#)
- [size_t send\(const void *buf_, size_t len_, int flags_=0\)](#)
- [bool send\(message_t &msg_, int flags_=0\)](#)
- [template<typename T > ZMQ_CPP11_DEPRECATED](#) ("from 4.4.1, use [send](#) taking [message_t](#) or buffer (for contiguous " "ranges), and [send_flags](#)") [bool send\(T first](#)

Public Attributes

- [T last](#)
- [T int flags_](#)

Friends

- [class monitor_t](#)

6.20.1 Detailed Description

Definition at line [2181](#) of file [zmq.hpp](#).

6.20.2 Constructor & Destructor Documentation

6.20.2.1 socket_t() [1/2]

```
zmq::socket_t::socket_t ( ) [inline]
```

Definition at line [2186](#) of file [zmq.hpp](#).

6.20.2.2 `socket_t()` [2/2]

```
zmq::socket_t::socket_t (
    context_t & context_,
    int type_ ) [inline]
```

Definition at line 2188 of file [zmq.hpp](#).

6.20.2.3 `~socket_t()`

```
zmq::socket_t::~~socket_t ( ) [inline]
```

Definition at line 2219 of file [zmq.hpp](#).

6.20.3 Member Function Documentation

6.20.3.1 `bind()` [1/2]

```
void zmq::detail::socket_base::bind (
    const char * addr_ ) [inline], [inherited]
```

Definition at line 1878 of file [zmq.hpp](#).

6.20.3.2 `bind()` [2/2]

```
void zmq::detail::socket_base::bind (
    std::string const & addr ) [inline], [inherited]
```

Definition at line 1876 of file [zmq.hpp](#).

6.20.3.3 `close()`

```
void zmq::socket_t::close ( ) [inline]
```

Definition at line 2225 of file [zmq.hpp](#).

6.20.3.4 `connect()` [1/2]

```
void zmq::detail::socket_base::connect (
    const char * addr_ ) [inline], [inherited]
```

Definition at line 1896 of file [zmq.hpp](#).

6.20.3.5 `connect()` [2/2]

```
void zmq::detail::socket_base::connect (
    std::string const & addr ) [inline], [inherited]
```

Definition at line 1894 of file [zmq.hpp](#).

6.20.3.6 connected()

```
bool zmq::detail::socket_base::connected ( ) const [inline], [inherited]
```

Definition at line 1913 of file [zmq.hpp](#).

6.20.3.7 disconnect() [1/2]

```
void zmq::detail::socket_base::disconnect (
    const char * addr_ ) [inline], [inherited]
```

Definition at line 1905 of file [zmq.hpp](#).

6.20.3.8 disconnect() [2/2]

```
void zmq::detail::socket_base::disconnect (
    std::string const & addr ) [inline], [inherited]
```

Definition at line 1903 of file [zmq.hpp](#).

6.20.3.9 getsockopt() [1/2]

```
template<typename T >
T zmq::detail::socket_base::getsockopt (
    int option_ ) const [inline], [inherited]
```

Definition at line 1758 of file [zmq.hpp](#).

6.20.3.10 getsockopt() [2/2]

```
void zmq::detail::socket_base::getsockopt (
    int option_,
    void * optval_,
    size_t * optvallen_ ) const [inline], [inherited]
```

Definition at line 1749 of file [zmq.hpp](#).

6.20.3.11 operator socket_ref()

```
zmq::socket_t::operator socket\_ref ( ) [inline]
```

Definition at line 2242 of file [zmq.hpp](#).

6.20.3.12 operator void *()

```
zmq::socket_t::operator void * ( ) [inline]
```

Definition at line 2221 of file [zmq.hpp](#).

6.20.3.13 operator void const *()

```
zmq::socket_t::operator void const * ( ) const [inline]
```

Definition at line 2223 of file [zmq.hpp](#).

6.20.3.14 send() [1/2]

```
size_t zmq::detail::socket_base::send (
    const void * buf_,
    size_t len_,
    int flags_ = 0 ) [inline], [inherited]
```

Definition at line 1916 of file [zmq.hpp](#).

6.20.3.15 send() [2/2]

```
bool zmq::detail::socket_base::send (
    message_t & msg_,
    int flags_ = 0 ) [inline], [inherited]
```

Definition at line 1927 of file [zmq.hpp](#).

6.20.3.16 setsockopt() [1/2]

```
void zmq::detail::socket_base::setsockopt (
    int option_,
    const void * optval_,
    size_t optvallen_ ) [inline], [inherited]
```

Definition at line 1741 of file [zmq.hpp](#).

6.20.3.17 setsockopt() [2/2]

```
template<typename T >
void zmq::detail::socket_base::setsockopt (
    int option_,
    T const & optval ) [inline], [inherited]
```

Definition at line 1735 of file [zmq.hpp](#).

6.20.3.18 swap()

```
void zmq::socket_t::swap (
    socket_t & other ) [inline]
```

Definition at line 2236 of file [zmq.hpp](#).

6.20.3.19 unbind() [1/2]

```
void zmq::detail::socket_base::unbind (
    const char * addr_ ) [inline], [inherited]
```

Definition at line 1887 of file [zmq.hpp](#).

6.20.3.20 unbind() [2/2]

```
void zmq::detail::socket_base::unbind (
    std::string const & addr ) [inline], [inherited]
```

Definition at line 1885 of file [zmq.hpp](#).

6.20.3.21 ZMQ_CPP11_DEPRECATED()

```
template<typename T >
zmq::detail::socket_base::ZMQ_CPP11_DEPRECATED (
    "from 4.4. 1,
    use send taking message\_t or bufferfor contiguous " "ranges,
    and send_flags" ) [inherited]
```

6.20.4 Friends And Related Symbol Documentation

6.20.4.1 monitor_t

```
friend class monitor\_t [friend]
```

Definition at line 2183 of file [zmq.hpp](#).

6.20.5 Member Data Documentation

6.20.5.1 flags_

```
T int zmq::detail::socket_base::flags_ [inherited]
```

Definition at line 1942 of file [zmq.hpp](#).

6.20.5.2 last

```
T zmq::detail::socket_base::last [inherited]
```

Definition at line 1942 of file [zmq.hpp](#).

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

- [external/zmq/includes/zmq/zmq.hpp](#)

6.21 zmq_event_t Struct Reference

```
#include <zmq.hpp>
```

Public Attributes

- uint16_t [event](#)
- int32_t [value](#)

6.21.1 Detailed Description

Definition at line [207](#) of file [zmq.hpp](#).

6.21.2 Member Data Documentation

6.21.2.1 event

```
uint16_t zmq_event_t::event
```

Definition at line [209](#) of file [zmq.hpp](#).

6.21.2.2 value

```
int32_t zmq_event_t::value
```

Definition at line [210](#) of file [zmq.hpp](#).

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

- [external/zmq/includes/zmq/zmq.hpp](#)

6.22 zmq_msg_t Struct Reference

```
#include <zmq.h>
```

Public Attributes

- unsigned char [_](#)[64]

6.22.1 Detailed Description

Definition at line [251](#) of file [zmq.h](#).

6.22.2 Member Data Documentation

6.22.2.1 _

```
unsigned char zmq_msg_t::_[64]
```

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

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

- [external/zmq/includes/zmq/zmq.h](#)

6.23 zmq_pollitem_t Struct Reference

```
#include <zmq.h>
```

Public Attributes

- void * [socket](#)
- [zmq_fd_t](#) [fd](#)
- short [events](#)
- short [revents](#)

6.23.1 Detailed Description

Definition at line 520 of file [zmq.h](#).

6.23.2 Member Data Documentation

6.23.2.1 events

```
short zmq_pollitem_t::events
```

Definition at line 524 of file [zmq.h](#).

6.23.2.2 fd

```
zmq\_fd\_t zmq_pollitem_t::fd
```

Definition at line 523 of file [zmq.h](#).

6.23.2.3 revents

```
short zmq_pollitem_t::revents
```

Definition at line 525 of file [zmq.h](#).

6.23.2.4 socket

```
void* zmq_pollitem_t::socket
```

Definition at line 522 of file [zmq.h](#).

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

- [external/zmq/includes/zmq/zmq.h](#)

Chapter 7

File Documentation

7.1 examples/ExampleZMQCommandClientAmelas/AmelasExampleClient/amelas_client.cpp File Reference

```
#include "amelas_client.h"
```

Namespaces

- namespace [amelas](#)

7.2 amelas_client.cpp

[Go to the documentation of this file.](#)

```
00001 #include "amelas_client.h"
00002
00003 // AMELAS NAMESPACES
00004 //
=====
00005 namespace amelas{
00006 //
=====
00007
00008 using common::AmelasServerCommandStr;
00009 using common::AmelasServerResultStr;
00010 using common::ControllerError;
00011 using common::AmelasServerCommand;
00012 using common::AmelasServerResult;
00013 using zmqutils::common::ServerCommand;
00014 using zmqutils::common::ServerResult;
00015 using zmqutils::common::ResultType;
00016
00017 AmelasClient::AmelasClient(const std::string &server_endpoint) :
00018     zmqutils::CommandClientBase(server_endpoint)
00019 {}
00020
00021 void AmelasClient::onSendCommand(const RequestData &req, const zmq::multipart_t &msg)
00022 {
00023     // Get the command string.
00024     std::string cmd_str;
00025     cmd_str = (req.command <AmelasServerCommandStr.size()) ? AmelasServerCommandStr[req.command] :
"Unknown command";
00026     // Log.
00027     std::cout << std::string(80, '-') << std::endl;
00028     std::cout << "AMELAS CLIENT" << std::endl;
00029     std::cout << "-> ON SEND COMMAND: " << std::endl;
00030     std::cout << "Time: " << zmqutils::utils::currentISO8601Date() << std::endl;
```

```

00031     std::cout<<"Command: "<<static_cast<int>(req.command)<<std::endl;
00032     std::cout<<"Params size: "<<req.params_size<<std::endl;
00033     std::cout<<"Msg parts: "<<msg.size()<<std::endl;
00034     std::cout << std::string(80, '-') << std::endl;
00035 }
00036
00037
00038
00039 } // END NAMESPACES.
00040 //
=====
00041

```

7.3 examples/ExampleZMQCommandClientAmelas/AmelasExampleClient/amelas_client.h File Reference

```

#include <string>
#include <LibZMQUtils/CommandClient>
#include <LibZMQUtils/Utils>
#include <AmelasExampleServer/amelas_server.h>
#include <AmelasExampleServer/common.h>

```

Classes

- class [amelas::AmelasClient](#)

Namespaces

- namespace [amelas](#)

7.4 amelas_client.h

[Go to the documentation of this file.](#)

```

00001
00002 //
=====
00003 #pragma once
00004 //
=====
00005
00006 // C++ INCLUDES
00007 //
=====
00008 #include <string>
00009 //
=====
00010
00011 // ZMQUTILS INCLUDES
00012 //
=====
00013 #include <LibZMQUtils/CommandClient>
00014 #include <LibZMQUtils/Utils>
00015 //
=====
00016
00017 // AMELAS INCLUDES
00018 //
=====
00019 #include <AmelasExampleServer/amelas_server.h>
00020 #include <AmelasExampleServer/common.h>
00021 //
=====
00022

```

```

00023 // PROJECT INCLUDES
00024 //
=====
00025 //
=====
00026
00027 // AMELAS NAMESPACES
00028 //
=====
00029 namespace amelas{
00030 //
=====
00031
00032 using common::AmelasServerCommandStr;
00033
00034
00035
00036 class AmelasClient : public zmqutils::CommandClientBase
00037 {
00038 public:
00039
00040     AmelasClient(const std::string &server_endpoint);
00041
00042     // TODO
00043     //virtual void prepareRequest() = 0;
00044
00045 private:
00046
00047     void onSendCommand(const RequestData& req, const zmq::multipart_t& msg) override;
00048
00049     /* TODO
00050     void onConnected(const HostClient& client) override
00051     {
00052         // Log.
00053         std::cout << std::string(80, '-') << std::endl;
00054         std::cout<<"<AMELAS SERVER>"<<std::endl;
00055         std::cout<<"-> ON CONNECTED: "<<std::endl;
00056         std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00057         std::cout<<"Current Clients: "<<this->getConnectedClients().size()<<std::endl;
00058         std::cout<<"Client Id: "<<client.id<<std::endl;
00059         std::cout<<"Client Ip: "<<client.ip<<std::endl;
00060         std::cout<<"Client Host: "<<client.hostname<<std::endl;
00061         std::cout<<"Client Process: "<<client.pid<<std::endl;
00062         std::cout << std::string(80, '-') << std::endl;
00063     }
00064     */
00065 };
00066
00067 } // END NAMESPACES.
00068 //
=====

```

7.5 examples/ExampleZMQCommandClientAmelas/ExampleZMQClientAmelas.cpp File Reference

```

#include <iostream>
#include <cstring>
#include "AmelasExampleController/common.h"
#include "AmelasExampleServer/common.h"
#include "AmelasExampleClient/amelas_client.h"

```

Functions

- void [parseCommand](#) ([CommandClientBase](#) &client, const std::string &command)
- int [main](#) (int argc, char **argv)

7.5.1 Function Documentation

7.5.1.1 main()

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

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

7.5.1.2 parseCommand()

```
void parseCommand (
    CommandClientBase & client,
    const std::string & command )
```

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

7.6 ExampleZMQClientAmelas.cpp

[Go to the documentation of this file.](#)

```
00001
00002
00003 #include <iostream>
00004 #include <cstring>
00005
00006 #include "AmelasExampleController/common.h"
00007 #include "AmelasExampleServer/common.h"
00008 #include "AmelasExampleClient/amelas_client.h"
00009
00010
00011
00012 using namespace zmqutils;
00013 using namespace amelas;
00014
00015 using amelas::common::AmelasServerCommand;
00016 using amelas::common::AmelasServerResult;
00017 using zmqutils::common::CommandType;
00018
00019 void parseCommand(CommandClientBase &client, const std::string &command)
00020 {
00021     zmqutils::common::ClientResult send_result = ClientResult::COMMAND_OK;
00022
00023     char *command_str = new char[command.size()];
00024     std::copy(command.begin(), command.end(), command_str);
00025
00026     char *token = std::strtok(command_str, " ");
00027
00028     if (token)
00029     {
00030         CommandType command_id;
00031
00032         try
00033         {
00034             command_id = static_cast<CommandType>(std::stoi(token));
00035         }
00036         catch (...)
00037         {
00038             std::cerr << "Failed at sending command." << std::endl;
00039             delete[] command_str;
00040             return;
00041         }
00042
00043         RequestData command_msg(command_id);
00044
00045         bool valid = true;
00046
00047         if (command_id == static_cast<CommandType>(ServerCommand::REQ_CONNECT))
00048         {
```



```

00049         std::cout << "Sending connect message" << std::endl;
00050     }
00051     else if (command_id == static_cast<CommandType>(ServerCommand::REQ_DISCONNECT))
00052     {
00053         std::cout << "Sending disconnect message" << std::endl;
00054     }
00055     else if (command_id == static_cast<CommandType>(ServerCommand::REQ_ALIVE))
00056     {
00057         std::cout << "Sending keepalive command." << std::endl;
00058     }
00059     else if (command_id == static_cast<CommandType>(AmelasServerCommand::REQ_GET_DATETIME))
00060     {
00061         std::cout << "Get datetime command not implemented yet." << std::endl;
00062         valid = false;
00063     }
00064     else if (command_id == static_cast<CommandType>(AmelasServerCommand::REQ_SET_DATETIME))
00065     {
00066         std::cout << "Set datetime command not implemented yet." << std::endl;
00067         valid = false;
00068     }
00069     else if (command_id == static_cast<CommandType>(AmelasServerCommand::REQ_GET_HOME_POSITION))
00070     {
00071         std::cout << "Sending get home position command." << std::endl;
00072     }
00073     else if (command_id == static_cast<CommandType>(AmelasServerCommand::REQ_SET_HOME_POSITION))
00074     {
00075         std::cout << "Sending set home position command." << std::endl;
00076
00077         bool valid_params = true;
00078         double az = 0., el = 0.;
00079         char *param_token = std::strtok(nullptr, " ");
00080
00081         try
00082         {
00083             az = std::stod(param_token);
00084         }
00085         catch (...)
00086         {
00087             std::cerr << "Bad parameter azimuth issued.";
00088             valid_params = false;
00089         }
00090
00091         if (valid_params)
00092         {
00093             param_token = std::strtok(nullptr, " ");
00094
00095             try
00096             {
00097                 el = std::stod(param_token);
00098             }
00099             catch (...)
00100             {
00101                 std::cerr << "Bad parameter elevation issued.";
00102                 valid_params = false;
00103             }
00104         }
00105
00106         if (valid_params)
00107         {
00108             std::cout << "Sending: " << az << " " << el << std::endl;
00109
00110             command_msg.params = std::unique_ptr<std::uint8_t>(new std::uint8_t[16]);
00111             command_msg.params_size = 16;
00112
00113             zmqutils::utils::binarySerializeDeserialize(&az, 8, command_msg.params.get());
00114             zmqutils::utils::binarySerializeDeserialize(&el, 8, command_msg.params.get() + 8);
00115         }
00116
00117         valid = valid_params;
00118     }
00119     else
00120     {
00121         valid = false;
00122     }
00123 }
00124
00125 // TODO MOVE ALL OF THIS TO A SUBCLASS IN A PURE VIRTUAL. THE FUNCTION WILL RETURN
ClientResult
00126 // TODO THE ERROR CONTROL MUST BE IN THE BASE CLIENT. THE SUBCLASS MUST CONTROL THE OUTPUT
DATA AND CUSTOM ERRORS ONLY.
00127 // TODO DISABLE SEND WITH THIS WAY THE RESERVED COMMANDS.
00128 // TODO CREATE doConnect, doDisconnect, checkServerAlive
00129 // TODO CREATE IN THE CLIENT THE INTERNAL CALLBACKS LIKE THE SERVER.
00130 // TODO MOVE THE PROCESSING OF EACH COMPLEX RESPONSE TO A FUNCTION.
00131
00132 if (valid)
00133 {

```

```

00134         // TODO MOVE ALL
00135         ClientResult result = ClientResult::COMMAND_OK;
00136         CommandReply reply;
00137
00138         send_result = client.sendCommand(command_msg, reply);
00139
00140
00141         if (send_result != ClientResult::COMMAND_OK)
00142         {
00143             //std::cerr << "Command sending failed with code: " << send_result << std::endl;
00144         }
00145         else
00146         {
00147             constexpr std::size_t res_sz = sizeof(amelas::common::ControllerError);
00148             constexpr std::size_t double_sz = sizeof(double);
00149
00150             std::cout<<"Server result: "<<static_cast<int>(reply.result)<<std::endl;
00151
00152             // Get the controller result.
00153             // TODO ERROR CONTROL
00154
00155
00156
00157             if (command_id ==
static_cast<CommandType>(AmelasServerCommand::REQ_GET_HOME_POSITION))
00158             {
00159                 if (reply.params_size == (res_sz + 2*double_sz))
00160                 {
00161                     amelas::common::ControllerError error;
00162                     double az;
00163                     double el;
00164
00165                     // Deserialize the parameters.
00166                     zmqutils::utils::binarySerializeDeserialize(reply.params.get(), res_sz,
reply.params.get());
00167                     zmqutils::utils::binarySerializeDeserialize(reply.params.get() + res_sz,
double_sz, &az);
00168                     zmqutils::utils::binarySerializeDeserialize(reply.params.get() + res_sz +
double_sz, double_sz, &el);
00169
00170                     // Generate the struct.
00171                     std::cout<<"Controller error: "<<static_cast<int>(error)<<std::endl;
00172                     std::cout<<"Az: "<<az<<std::endl;
00173                     std::cout<<"El: "<<el<<std::endl;
00174                 }
00175                 else
00176                 {
00177                     std::cout<<"BAD PARAMS"<<std::endl;
00178                     // RETURN BAD PARAMS
00179                     //result = ClientResult::
00180                 }
00181             }
00182         }
00183     }
00184     else
00185     {
00186         std::cerr << "Command is not implemented or valid" << std::endl;
00187     }
00188 }
00189 }
00190 else
00191 {
00192     std::cerr << "Not a valid command" << std::endl;
00193 }
00194
00195 delete[] command_str;
00196 }
00197 }
00198
00199
00200 int main(int argc, char**argv)
00201 {
00202
00203     int port = 9999;
00204     std::string ip = "127.0.0.1";
00205
00206     if (argc == 2)
00207     {
00208         ip = argv[1];
00209     }
00210     if (argc == 3)
00211     {
00212         ip = argv[1];
00213         try
00214         {
00215             port = std::stoi(argv[2]);
00216         } catch (...)

```

```

00217     {
00218         std::cerr << "Not recognized port in input: " << argv[2] << std::endl;
00219         return -1;
00220     }
00221
00222     }
00223     else if (argc > 3)
00224     {
00225         std::cout << "Usage: ZMQClient [ip] [port]" << std::endl;
00226         return 0;
00227     }
00228
00229     std::string endpoint = "tcp://" + ip + ":" + std::to_string(port);
00230     AmelasClient client(endpoint);
00231     client.startClient("Ethernet");
00232     //client.setClientHostIP("");
00233     std::cout << "Connecting to endpoint: " << endpoint << std::endl;
00234     //client.startAutoAlive();
00235     std::string command;
00236
00237     while(true)
00238     {
00239         std::cout<<"Write a command: ";
00240         std::getline(std::cin, command);
00241
00242         if (command == "exit")
00243             break;
00244
00245         parseCommand(client, command);
00246     }
00247
00248     std::cout << "Requested client to stop. Bye." << std::endl;
00249
00250     client.stopClient();
00251
00252
00253     return 0;
00254 }

```

7.7 examples/ExampleZMQCommanServerAmelas/AmelasExampleController/amelas_controller.h File Reference

```

#include <map>
#include <string>
#include <LibZMQUtils/CommandServer>
#include <LibZMQUtils/Utils>
#include "common.h"

```

Classes

- class [AmelasController](#)

Namespaces

- namespace [Amelas](#)

7.8 amelas_controller.h

[Go to the documentation of this file.](#)

```

00001
00002  /*****
00003   *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
00004   *

```

```

00003  *
00004  *   Copyright (C) 2023 Degoras Project Team
00005  *
00006  *   < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
00007  *
00008  *   < Jesús Relinque Madroñal >
00009  *
00010  *   This file is part of LibZMQUtils.
00011  *
00012  *
00013  *   Licensed under the European Union Public License (EURL), Version 1.2 or subsequent versions of
the EURL license *
00014  *   as soon they will be approved by the European Commission (IDABC).
00015  *
00016  *
00017  *   This project is free software: you can redistribute it and/or modify it under the terms of the
EURL license as *
00018  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
00019  *
00020  *
00021  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
or agreed to in *
00022  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
without even the *
00023  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EURL license to
check specific *
00024  *   language governing permissions and limitations and more details.
00025  *
00026  *
00027  *   You should use this project in compliance with the EURL license. You should have received a copy
of the license *
00028  *   along with this project. If not, see the license at < https://eupl.eu/ >.
00029  *
00030  *
00031  *
00032  *
00033  *
00034  *
00035  *
00036  *
00037  *
00038  *
00039  *
00040  *
00041  *
00042  *
00043  *
00044  *
00045  *
00046  *
00047  *
00048  *
00049  *
00050  *
00051  *
00052  *
00053  *
00054  *
00055  *
00056  *
00057  *
00058  *
00059  *
00060  *
00061  *
00062  *
00063  *
00064  *
00065  *
00066  *
00067  *
00068  *
00069  *
00070  *
00071  *
00072  *
00073  *
00074  *
00075  *
00076  *
00077  *
00078  *
00079  *
00080  *
00081  *
00082  *
00083  *
00084  *
00085  *
00086  *
00087  *
00088  *
00089  *
00090  *
00091  *
00092  *
00093  *
00094  *
00095  *
00096  *
00097  *
00098  *
00099  *
00100  *
00101  *
00102  *
00103  *
00104  *
00105  *
00106  *
00107  *
00108  *
00109  *
00110  *
00111  *
00112  *
00113  *
00114  *
00115  *
00116  *
00117  *
00118  *
00119  *
00120  *
00121  *
00122  *
00123  *
00124  *
00125  *
00126  *
00127  *
00128  *
00129  *
00130  *
00131  *
00132  *
00133  *
00134  *
00135  *
00136  *
00137  *
00138  *
00139  *
00140  *
00141  *
00142  *
00143  *
00144  *
00145  *
00146  *
00147  *
00148  *
00149  *
00150  *
00151  *
00152  *
00153  *
00154  *
00155  *
00156  *
00157  *
00158  *
00159  *
00160  *
00161  *
00162  *
00163  *
00164  *
00165  *
00166  *
00167  *
00168  *
00169  *
00170  *
00171  *
00172  *
00173  *
00174  *
00175  *
00176  *
00177  *
00178  *
00179  *
00180  *
00181  *
00182  *
00183  *
00184  *
00185  *
00186  *
00187  *
00188  *
00189  *
00190  *
00191  *
00192  *
00193  *
00194  *
00195  *
00196  *
00197  *
00198  *
00199  *
00200  *
00201  *
00202  *
00203  *
00204  *
00205  *
00206  *
00207  *
00208  *
00209  *
00210  *
00211  *
00212  *
00213  *
00214  *
00215  *
00216  *
00217  *
00218  *
00219  *
00220  *
00221  *
00222  *
00223  *
00224  *
00225  *
00226  *
00227  *
00228  *
00229  *
00230  *
00231  *
00232  *
00233  *
00234  *
00235  *
00236  *
00237  *
00238  *
00239  *
00240  *
00241  *
00242  *
00243  *
00244  *
00245  *
00246  *
00247  *
00248  *
00249  *
00250  *
00251  *
00252  *
00253  *
00254  *
00255  *
00256  *
00257  *
00258  *
00259  *
00260  *
00261  *
00262  *
00263  *
00264  *
00265  *
00266  *
00267  *
00268  *
00269  *
00270  *
00271  *
00272  *
00273  *
00274  *
00275  *
00276  *
00277  *
00278  *
00279  *
00280  *
00281  *
00282  *
00283  *
00284  *
00285  *
00286  *
00287  *
00288  *
00289  *
00290  *
00291  *
00292  *
00293  *
00294  *
00295  *
00296  *
00297  *
00298  *
00299  *
00300  *
00301  *
00302  *
00303  *
00304  *
00305  *
00306  *
00307  *
00308  *
00309  *
00310  *
00311  *
00312  *
00313  *
00314  *
00315  *
00316  *
00317  *
00318  *
00319  *
00320  *
00321  *
00322  *
00323  *
00324  *
00325  *
00326  *
00327  *
00328  *
00329  *
00330  *
00331  *
00332  *
00333  *
00334  *
00335  *
00336  *
00337  *
00338  *
00339  *
00340  *
00341  *
00342  *
00343  *
00344  *
00345  *
00346  *
00347  *
00348  *
00349  *
00350  *
00351  *
00352  *
00353  *
00354  *
00355  *
00356  *
00357  *
00358  *
00359  *
00360  *
00361  *
00362  *
00363  *
00364  *
00365  *
00366  *
00367  *
00368  *
00369  *
00370  *
00371  *
00372  *
00373  *
00374  *
00375  *
00376  *
00377  *
00378  *
00379  *
00380  *
00381  *
00382  *
00383  *
00384  *
00385  *
00386  *
00387  *
00388  *
00389  *
00390  *
00391  *
00392  *
00393  *
00394  *
00395  *
00396  *
00397  *
00398  *
00399  *
00400  *
00401  *
00402  *
00403  *
00404  *
00405  *
00406  *
00407  *
00408  *
00409  *
00410  *
00411  *
00412  *
00413  *
00414  *
00415  *
00416  *
00417  *
00418  *
00419  *
00420  *
00421  *
00422  *
00423  *
00424  *
00425  *
00426  *
00427  *
00428  *
00429  *
00430  *
00431  *
00432  *
00433  *
00434  *
00435  *
00436  *
00437  *
00438  *
00439  *
00440  *
00441  *
00442  *
00443  *
00444  *
00445  *
00446  *
00447  *
00448  *
00449  *
00450  *
00451  *
00452  *
00453  *
00454  *
00455  *
00456  *
00457  *
00458  *
00459  *
00460  *
00461  *
00462  *
00463  *
00464  *
00465  *
00466  *
00467  *
00468  *
00469  *
00470  *
00471  *
00472  *
00473  *
00474  *
00475  *
00476  *
00477  *
00478  *
00479  *
00480  *
00481  *
00482  *
00483  *
00484  *
00485  *
00486  *
00487  *
00488  *
00489  *
00490  *
00491  *
00492  *
00493  *
00494  *
00495  *
00496  *
00497  *
00498  *
00499  *
00500  *
00501  *
00502  *
00503  *
00504  *
00505  *
00506  *
00507  *
00508  *
00509  *
00510  *
00511  *
00512  *
00513  *
00514  *
00515  *
00516  *
00517  *
00518  *
00519  *
00520  *
00521  *
00522  *
00523  *
00524  *
00525  *
00526  *
00527  *
00528  *
00529  *
00530  *
00531  *
00532  *
00533  *
00534  *
00535  *
00536  *
00537  *
00538  *
00539  *
00540  *
00541  *
00542  *
00543  *
00544  *
00545  *
00546  *
00547  *
00548  *
00549  *
00550  *
00551  *
00552  *
00553  *
00554  *
00555  *
00556  *
00557  *
00558  *
00559  *
00560  *
00561  *
00562  *
00563  *
00564  *
00565  *
00566  *
00567  *
00568  *
00569  *
00570  *
00571  *
00572  *
00573  *
00574  *
00575  *
00576  *
00577  *
00578  *
00579  *
00580  *
00581  *
00582  *
00583  *
00584  *
00585  *
00586  *
00587  *
00588  *
00589  *
00590  *
00591  *
00592  *
00593  *
00594  *
00595  *
00596  *
00597  *
00598  *
00599  *
00600  *
00601  *
00602  *
00603  *
00604  *
00605  *
00606  *
00607  *
00608  *
00609  *
00610  *
00611  *
00612  *
00613  *
00614  *
00615  *
00616  *
00617  *
00618  *
00619  *
00620  *
00621  *
00622  *
00623  *
00624  *
00625  *
00626  *
00627  *
00628  *
00629  *
00630  *
00631  *
00632  *
00633  *
00634  *
00635  *
00636  *
00637  *
00638  *
00639  *
00640  *
00641  *
00642  *
00643  *
00644  *
00645  *
00646  *
00647  *
00648  *
00649  *
00650  *
00651  *
00652  *
00653  *
00654  *
00655  *
00656  *
00657  *
00658  *
00659  *
00660  *
00661  *
00662  *
00663  *
00664  *
00665  *
00666  *
00667  *
00668  *
00669  *
00670  *
00671  *
00672  *
00673  *
00674  *
00675  *
00676  *
00677  *
00678  *
00679  *
00680  *
00681  *
00682  *
00683  *
00684  *
00685  *
00686  *
00687  *
00688  *
00689  *
00690  *
00691  *
00692  *
00693  *
00694  *
00695  *
00696  *
00697  *
00698  *
00699  *
00700  *
00701  *
00702  *
00703  *
00704  *
00705  *
00706  *
00707  *
00708  *
00709  *
00710  *
00711  *
00712  *
00713  *
00714  *
00715  *
00716  *
00717  *
00718  *
00719  *
00720  *
00721  *
00722  *
00723  *
00724  *
00725  *
00726  *
00727  *
00728  *
00729  *
00730  *
00731  *
00732  *
00733  *
00734  *
00735  *
00736  *
00737  *
00738  *
00739  *
00740  *
00741  *
00742  *
00743  *
00744  *
00745  *
00746  *
00747  *
00748  *
00749  *
00750  *
00751  *
00752  *
00753  *
00754  *
00755  *
00756  *
00757  *
00758  *
00759  *
00760  *
00761  *
00762  *
00763  *
00764  *
00765  *
00766  *
00767  *
00768  *
00769  *
00770  *
00771  *
00772  *
00773  *
00774  *
00775  *
00776  *
00777  *
00778  *
00779  *
00780  *
00781  *
00782  *
00783  *
00784  *
00785  *
00786  *
00787  *
00788  *
00789  *
00790  *
00791  *
00792  *
00793  *
00794  *
00795  *
00796  *
00797  *
00798  *
00799  *
00800  *
00801  *
00802  *
00803  *
00804  *
00805  *
00806  *
00807  *
00808  *
00809  *
00810  *
00811  *
00812  *
00813  *
00814  *
00815  *
00816  *
00817  *
00818  *
00819  *
00820  *
00821  *
00822  *
00823  *
00824  *
00825  *
00826  *
00827  *
00828  *
00829  *
00830  *
00831  *
00832  *
00833  *
00834  *
00835  *
00836  *
00837  *
00838  *
00839  *
00840  *
00841  *
00842  *
00843  *
00844  *
00845  *
00846  *
00847  *
00848  *
00849  *
00850  *
00851  *
00852  *
00853  *
00854  *
00855  *
00856  *
00857  *
00858  *
00859  *
00860  *
00861  *
00862  *
00863  *
00864  *
00865  *
00866  *
00867  *
00868  *
00869  *
00870  *
00871  *
00872  *
00873  *
00874  *
00875  *
00876  *
00877  *
00878  *
00879  *
00880  *
00881  *
00882  *
00883  *
00884  *
00885  *
00886  *
00887  *
00888  *
00889  *
00890  *
00891  *
00892  *
00893  *
00894  *
00895  *
00896  *
00897  *
00898  *
00899  *
00900  *
00901  *
00902  *
00903  *
00904  *
00905  *
00906  *
00907  *
00908  *
00909  *
00910  *
00911  *
00912  *
00913  *
00914  *
00915  *
00916  *
00917  *
00918  *
00919  *
00920  *
00921  *
00922  *
00923  *
00924  *
00925  *
00926  *
00927  *
00928  *
00929  *
00930  *
00931  *
00932  *
00933  *
00934  *
00935  *
00936  *
00937  *
00938  *
00939  *
00940  *
00941  *
00942  *
00943  *
00944  *
00945  *
00946  *
00947  *
00948  *
00949  *
00950  *
00951  *
00952  *
00953  *
00954  *
00955  *
00956  *
00957  *
00958  *
00959  *
00960  *
00961  *
00962  *
00963  *
00964  *
00965  *
00966  *
00967  *
00968  *
00969  *
00970  *
00971  *
00972  *
00973  *
00974  *
00975  *
00976  *
00977  *
00978  *
00979  *
00980  *
00981  *
00982  *
00983  *
00984  *
00985  *
00986  *
00987  *
00988  *
00989  *
00990  *
00991  *
00992  *
00993  *
00994  *
00995  *
00996  *
00997  *
00998  *
00999  *
01000  *

```

```

00061     ControllerError setHomePosition(const AltAzPos& pos)
00062     {
00063         // Auxiliar result.
00064         ControllerError error = ControllerError::SUCCESS;
00065
00066         // Check the provided values.
00067         if (pos.az >= 360.0 || pos.az < 0.0 || pos.el >= 90. || pos.el < 0.)
00068         {
00069             error = ControllerError::INVALID_POSITION;
00070         }
00071         else
00072         {
00073             this->home_pos_ = pos;
00074         }
00075
00076         std::cout << std::string(100, '-') << std::endl;
00077         std::cout<<"<AMELAS CONTROLLER>"<<std::endl;
00078         std::cout<<"-> SET_HOME_POSITION"<<std::endl;
00079         std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00080         std::cout<<"Az: "<<pos.az<<std::endl;
00081         std::cout<<"El: "<<pos.el<<std::endl;
00082         std::cout << std::string(100, '-') << std::endl;
00083
00084         return error;
00085     }
00086
00087     ControllerError getHomePosition(AltAzPos& pos)
00088     {
00089         pos = this->home_pos_;
00090
00091         std::cout << std::string(100, '-') << std::endl;
00092         std::cout<<"<AMELAS CONTROLLER>"<<std::endl;
00093         std::cout<<"-> GET_HOME_POSITION"<<std::endl;
00094         std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00095         std::cout << std::string(100, '-') << std::endl;
00096
00097         return ControllerError::SUCCESS;
00098     }
00099
00100     ControllerError getDatetime(std::string&)
00101     {
00102         return ControllerError::SUCCESS;
00103     }
00104
00105 private:
00106     AltAzPos home_pos_;
00107 };
00108
00109 };
00110
00111 } // END NAMESPACES.
00112 //
=====
00113

```

7.9 examples/ExampleZMQCommanServerAmelas/AmelasExampleController/common.h File Reference

```

#include <string>
#include <map>
#include <vector>
#include <variant>
#include <functional>

```

Classes

- struct [amelas::common::AltAzPos](#)

Namespaces

- namespace [amelas](#)
- namespace [amelas::common](#)

Typedefs

- using `amelas::common::SetHomePositionCallback` = `std::function< ControllerError(const AltAzPos &)>`
- using `amelas::common::GetHomePositionCallback` = `std::function< ControllerError(AltAzPos &)>`
- using `amelas::common::GetDatetimeCallback` = `std::function< ControllerError(std::string &)>`
- using `amelas::common::ControllerCallback` = `std::variant< SetHomePositionCallback, GetHomePositionCallback, GetDatetimeCallback >`

Enumerations

- enum class `amelas::common::ControllerError` : `std::uint32_t` { `amelas::common::SUCCESS` = 0 , `amelas::common::INVALID_POSITION` = 1 , `amelas::common::UNSAFE_POSITION` = 2 }

7.10 common.h

[Go to the documentation of this file.](#)

```
00001
00002 //
=====
00003 #pragma once
00004 //
=====
00005
00006 // C++ INCLUDES
00007 //
=====
00008 #include <string>
00009 #include <map>
00010 #include <vector>
00011 #include <variant>
00012 #include <functional>
00013 //
=====
00014
00015 // AMELAS NAMESPACES
00016 //
=====
00017 namespace amelas{
00018 namespace common{
00019 //
=====
00020
00021 // CONSTANTS
00022 //
=====
00023
00024 //
=====
00025
00026 // CONVENIENT ALIAS, ENUMERATIONS AND CONSTEXPR
00027 //
=====
00028
00029 enum class ControllerError : std::uint32_t
00030 {
00031     SUCCESS = 0,
00032     INVALID_POSITION = 1,
00033     UNSAFE_POSITION = 2
00034 };
00035
00036 struct AltAzPos
00037 {
00038     AltAzPos(double az, double el):
00039         az(az), el(el){}
00040
00041     AltAzPos(): az(-1), el(-1){}
00042
00043     double az;
00044     double el;
00045 };
00046
00047 // Callback function type aliases
00048 using SetHomePositionCallback = std::function<ControllerError(const AltAzPos&)>;
00049 using GetHomePositionCallback = std::function<ControllerError(AltAzPos&)>;
```

```

00050 using GetDatetimeCallback = std::function<ControllerError(std::string&)>;
00051
00052 // Callback variant.
00053 using ControllerCallback = std::variant<SetHomePositionCallback,
00054                                         GetHomePositionCallback,
00055                                         GetDatetimeCallback>;
00056
00057
00058
00059
00060
00061 //
=====
00062
00063 }} // END NAMESPACES.
00064 //
=====

```

7.11 examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/common.h File Reference

```

#include <functional>
#include <any>
#include <LibZMQUtils/Utils>

```

Namespaces

- namespace [amelas](#)
- namespace [amelas::common](#)

Enumerations

- enum class [amelas::common::AmelasServerCommand](#) : [zmquutils::common::CommandType](#) {
[amelas::common::REQ_SET_DATETIME](#) = 11 , [amelas::common::REQ_GET_DATETIME](#) = 12 ,
[amelas::common::REQ_SET_HOME_POSITION](#) = 13 , [amelas::common::REQ_GET_HOME_POSITION](#) =
14 ,
[amelas::common::END_AMELAS_COMMANDS](#) }
- enum class [amelas::common::AmelasServerResult](#) : [zmquutils::common::ResultType](#) { [amelas::common::EMPTY_CALLBACK](#)
= 21 , [amelas::common::INVALID_CALLBACK](#) = 22 }

Variables

- static constexpr auto [amelas::common::AmelasServerCommandStr](#)
- static constexpr auto [amelas::common::AmelasServerResultStr](#)
- constexpr int [amelas::common::kMinCmdId](#) = static_cast<int>(zmquutils::common::ServerCommand::END_BASE_COMMAND
+ 1
- constexpr int [amelas::common::kMaxCmdId](#) = static_cast<int>(AmelasServerCommand::END_AMELAS_COMMANDS)
- 1

7.12 common.h

[Go to the documentation of this file.](#)

```

00001
00002 #include <functional>
00003 #include <any>
00004
00005 #include <LibZMQUtils/Utils>
00006
00007 //
=====
00008 #pragma once
00009 //
=====
00010
00011 // AMELAS NAMESPACES
00012 //
=====
00013 namespace amelas{
00014 namespace common{
00015 //
=====
00016
00017 // Specific subclass commands (0 to 4 are reserved for the base server).
00018 // WARNING: In our approach, the server commands must be always in order.
00019 enum class AmelasServerCommand : zmqutils::common::CommandType
00020 {
00021     REQ_SET_DATETIME      = 11,
00022     REQ_GET_DATETIME      = 12,
00023     REQ_SET_HOME_POSITION = 13,
00024     REQ_GET_HOME_POSITION = 14,
00025     END_AMELAS_COMMANDS
00026 };
00027
00028 // Specific subclass errors (0 to 20 are reserved for the base server).
00029 enum class AmelasServerResult : zmqutils::common::ResultType
00030 {
00031     EMPTY_CALLBACK = 21,
00032     INVALID_CALLBACK = 22
00033 };
00034
00035 // Extend the base command strings with those of the subclass.
00036 static constexpr auto AmelasServerCommandStr = zmqutils::utils::joinArraysConstexpr(
00037     zmqutils::common::ServerCommandStr,
00038     std::array<const char*, 5>
00039     {
00040         "REQ_SET_DATETIME",
00041         "REQ_GET_DATETIME",
00042         "REQ_SET_HOME_POSITION",
00043         "REQ_GET_HOME_POSITION",
00044         "END_DRGG_COMMANDS"
00045     });
00046
00047 // Extend the base result strings with those of the subclass.
00048 static constexpr auto AmelasServerResultStr = zmqutils::utils::joinArraysConstexpr(
00049     zmqutils::common::ServerResultStr,
00050     std::array<const char*, 2>
00051     {
00052         "EMPTY_CALLBACK - The external callback for the command is empty.",
00053         "INVALID_CALLBACK - The external callback for the command is invalid."
00054     });
00055
00056 // Usefull const expressions.
00057 constexpr int kMinCmdId = static_cast<int>(zmqutils::common::ServerCommand::END_BASE_COMMANDS) + 1;
00058 constexpr int kMaxCmdId = static_cast<int>(AmelasServerCommand::END_AMELAS_COMMANDS) - 1;
00059
00060 }} // END NAMESPACES.
00061 //
=====

```

7.13 includes/LibZMQUtils/CommandServerClient/common.h File Reference

This file contains common elements for the whole library.

```

#include <string>
#include <iostream>

```



```
#include <map>
#include <vector>
#include <cstring>
#include <memory>
#include <zmq/zmq.hpp>
#include <zmq/zmq_addon.hpp>
#include "LibZMQUtils/libzmqutils_global.h"
#include "LibZMQUtils/utils.h"
```

Classes

- struct [zmqutils::common::HostClient](#)
- struct [zmqutils::common::CommandRequest](#)
- struct [zmqutils::common::CommandReply](#)
- struct [zmqutils::common::RequestData](#)

Namespaces

- namespace [zmqutils](#)
- namespace [zmqutils::common](#)

Typedefs

- using [zmqutils::common::CommandType](#) = std::uint32_t
Type used for the BaseServerCommand enumeration.
- using [zmqutils::common::ResultType](#) = std::uint32_t
Type used for the BaseServerResult enumeration.

Enumerations

- enum class [zmqutils::common::ServerCommand](#) : CommandType {
 [zmqutils::common::INVALID_COMMAND](#) = 0 , [zmqutils::common::REQ_CONNECT](#) = 1 , [zmqutils::common::REQ_DISCONNECT](#) = 2 , [zmqutils::common::REQ_ALIVE](#) = 3 ,
 [zmqutils::common::RESERVED_COMMANDS](#) = 4 , [zmqutils::common::END_BASE_COMMANDS](#) = 10 }
- enum class [zmqutils::common::ServerResult](#) : ResultType {
 [zmqutils::common::COMMAND_OK](#) = 0 , [zmqutils::common::INTERNAL_ZMQ_ERROR](#) = 1 , [zmqutils::common::EMPTY_MESSAGE](#) = 2 , [zmqutils::common::EMPTY_CLIENT_IP](#) = 3 ,
 [zmqutils::common::EMPTY_CLIENT_NAME](#) = 4 , [zmqutils::common::EMPTY_CLIENT_PID](#) = 5 ,
 [zmqutils::common::EMPTY_PARAMS](#) = 6 , [zmqutils::common::TIMEOUT_REACHED](#) = 7 ,
 [zmqutils::common::INVALID_PARTS](#) = 8 , [zmqutils::common::UNKNOWN_COMMAND](#) = 9 , [zmqutils::common::INVALID_MESSAGE](#) = 10 , [zmqutils::common::CLIENT_NOT_CONNECTED](#) = 11 ,
 [zmqutils::common::ALREADY_CONNECTED](#) = 12 , [zmqutils::common::BAD_PARAMETERS](#) = 13 ,
 [zmqutils::common::COMMAND_FAILED](#) = 14 , [zmqutils::common::NOT_IMPLEMENTED](#) = 15 ,
 [zmqutils::common::BAD_NO_PARAMETERS](#) = 16 , [zmqutils::common::END_BASE_ERRORS](#) = 20 }
- enum class [zmqutils::common::ClientResult](#) : ResultType {
 [zmqutils::common::COMMAND_OK](#) = 0 , [zmqutils::common::INTERNAL_ZMQ_ERROR](#) = 1 , [zmqutils::common::EMPTY_MESSAGE](#) = 2 , [zmqutils::common::EMPTY_PARAMS](#) = 6 ,
 [zmqutils::common::TIMEOUT_REACHED](#) = 7 , [zmqutils::common::INVALID_PARTS](#) = 8 , [zmqutils::common::INVALID_MESSAGE](#) = 10 , [zmqutils::common::CLIENT_STOPPED](#) = 17 ,
 [zmqutils::common::END_BASE_ERRORS](#) = 20 }

Variables

- constexpr int `zmqutils::common::kDefaultClientAliveTimeoutMsec` = 8000
Default timeout for consider a client dead.
- constexpr int `zmqutils::common::kDefaultServerAliveTimeoutMsec` = 3000
Default timeout for consider a server dead.
- constexpr unsigned `zmqutils::common::kServerReconnTimes` = 10
Server reconnection default number of attempts.
- constexpr unsigned `zmqutils::common::kClientAlivePeriodMsec` = 1000
Default period for sending alive commands.
- constexpr int `zmqutils::common::kZmqEFSMError` = 156384765
ZMQ EFSM error.
- constexpr int `zmqutils::common::kMinBaseCmdId` = static_cast<int>(ServerCommand::INVALID_COMMAND) + 1
- constexpr int `zmqutils::common::kMaxBaseCmdId` = static_cast<int>(ServerCommand::END_BASE_COMMANDS) - 1
- static constexpr std::array< const char *, 11 > `zmqutils::common::ServerCommandStr`
- static constexpr std::array< const char *, 21 > `zmqutils::common::ServerResultStr`

7.13.1 Detailed Description

This file contains common elements for the whole library.

Author

Degoras Project Team

Copyright

EUPL License

Version

2307.1

Definition in file [common.h](#).

7.14 common.h

[Go to the documentation of this file.](#)

```
00001
```

```

00002  /*****
00003  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
00004  *
00005  *   Copyright (C) 2023 Degoras Project Team
00006  *
00007  *   < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
00008  *   < Jesús Relinque Madroñal >
00009  *
00010  *   This file is part of LibZMQUtils.
00011  *
00012  *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
00013  *   the EUPL license
00014  *   as soon they will be approved by the European Commission (IDABC).
00015  *

```

```

00012 *
00013 *   This project is free software: you can redistribute it and/or modify it under the terms of the
00014 *   EUPL license as
00015 *   published by the IDABC, either Version 1.2 or, at your option, any later version.
00016 *
00017 *   This project is distributed in the hope that it will be useful. Unless required by applicable law
00018 *   or agreed to in
00019 *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
00020 *   without even the
00021 *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
00022 *   check specific
00023 *   language governing permissions and limitations and more details.
00024 *
00025 *   You should use this project in compliance with the EUPL license. You should have received a copy
00026 *   of the license
00027 *   along with this project. If not, see the license at < https://eupl.eu/ >.
00028 *
00029 *****
00030 //
00031 //
00032 #pragma once
00033 //
00034 //
00035 //
00036 // C++ INCLUDES
00037 //
00038 //
00039 #include <string>
00040 #include <iostream>
00041 #include <map>
00042 #include <vector>
00043 #include <cstring>
00044 #include <memory>
00045 #include <zmq/zmq.hpp>
00046 #include <zmq/zmq_addon.hpp>
00047 //
00048 //
00049 // ZMQUTILS INCLUDES
00050 //
00051 #include "LibZMQUtils/libzmqutils_global.h"
00052 #include "LibZMQUtils/Utils.h"
00053 //
00054 //
00055 // ZMQUTILS NAMESPACES
00056 //
00057 namespace zmqutils{
00058 namespace common{
00059 //
00060 //
00061 // CONSTANTS
00062 //
00063 constexpr int kDefaultClientAliveTimeoutMsec = 8000;
00064 constexpr int kDefaultServerAliveTimeoutMsec = 3000;
00065 constexpr unsigned kServerReconnTimes = 10;
00066 constexpr unsigned kClientAlivePeriodMsec = 1000;
00067 constexpr int kZmqEFSMError = 156384765;
00068 //
00069 //
00070 // CONVENIENT ALIAS, ENUMERATIONS AND CONSTEXPR
00071 //
00072 //
00073 using CommandType = std::uint32_t;
00074 using ResultType = std::uint32_t;
00075 //
00076 enum class ServerCommand : CommandType
00077 {
00078     INVALID_COMMAND = 0,
00079     REQ_CONNECT = 1,
00080     REQ_DISCONNECT = 2,
00081     REQ_ALIVE = 3,
00082     RESERVED_COMMANDS = 4,
00083     END_BASE_COMMANDS = 10
00084 }

```

```

00091 };
00092
00098 enum class ServerResult : ResultType
00099 {
00100     COMMAND_OK                = 0,
00101     INTERNAL_ZMQ_ERROR        = 1,
00102     EMPTY_MSG                 = 2,
00103     EMPTY_CLIENT_IP           = 3,
00104     EMPTY_CLIENT_NAME         = 4,
00105     EMPTY_CLIENT_PID          = 5,
00106     EMPTY_PARAMS              = 6,
00107     TIMEOUT_REACHED           = 7,
00108     INVALID_PARTS             = 8,
00109     UNKNOWN_COMMAND           = 9,
00110     INVALID_MSG               = 10,
00111     CLIENT_NOT_CONNECTED      = 11,
00112     ALREADY_CONNECTED         = 12,
00113     BAD_PARAMETERS            = 13,
00114     COMMAND_FAILED            = 14,
00115     NOT_IMPLEMENTED           = 15,
00116     BAD_NO_PARAMETERS         = 16,
00117     END_BASE_ERRORS           = 20
00118 };
00119
00120
00121 // TODO MORE CASES RELATED TO THE CLIENT
00122 enum class ClientResult : ResultType
00123 {
00124     COMMAND_OK = 0,
00125     INTERNAL_ZMQ_ERROR        = 1,
00126     EMPTY_MSG                 = 2,
00127     EMPTY_PARAMS              = 6,
00128     TIMEOUT_REACHED           = 7,
00129     INVALID_PARTS             = 8,
00130     INVALID_MSG               = 10,
00131     CLIENT_STOPPED            = 17,
00132
00133     END_BASE_ERRORS           = 20
00134 };
00135 };
00136
00137 // Usefull const expressions.
00138 constexpr int kMinBaseCmdId = static_cast<int>(ServerCommand::INVALID_COMMAND) + 1;
00139 constexpr int kMaxBaseCmdId = static_cast<int>(ServerCommand::END_BASE_COMMANDS) - 1;
00140
00141 static constexpr std::array<const char*, 11> ServerCommandStr
00142 {
00143     "INVALID_COMMAND",
00144     "REQ_CONNECT",
00145     "REQ_DISCONNECT",
00146     "REQ_ALIVE",
00147     "RESERVED_BASE_COMMAND",
00148     "RESERVED_BASE_COMMAND",
00149     "RESERVED_BASE_COMMAND",
00150     "RESERVED_BASE_COMMAND",
00151     "RESERVED_BASE_COMMAND",
00152     "RESERVED_BASE_COMMAND",
00153     "END_BASE_COMMANDS"
00154 };
00155
00156 static constexpr std::array<const char*, 21> ServerResultStr
00157 {
00158     "COMMAND_OK - Command executed.",
00159     "INTERNAL_ZMQ_ERROR - Internal ZeroMQ error.",
00160     "EMPTY_MSG - Message is empty.",
00161     "EMPTY_CLIENT_IP - Client IP missing or empty.",
00162     "EMPTY_CLIENT_NAME - Client name missing or empty.",
00163     "EMPTY_CLIENT_PID - Client pid missing or empty.",
00164     "EMPTY_PARAMS - Command parameters missing or empty.",
00165     "TIMEOUT_REACHED - Operation timed out.",
00166     "INVALID_PARTS - Command has invalid parts.",
00167     "UNKNOWN_COMMAND - Command is not recognized.",
00168     "INVALID_COMMAND - Command is invalid.",
00169     "NOT_CONNECTED - Not connected to the server.",
00170     "ALREADY_CONNECTED - Already connected to the server.",
00171     "BAD_PARAMETERS - Provided parameters are invalid.",
00172     "COMMAND_FAILED - Command execution failed.",
00173     "NOT_IMPLEMENTED - Command is not implemented.",
00174     "RESERVED_BASE_ERROR",
00175     "RESERVED_BASE_ERROR",
00176     "RESERVED_BASE_ERROR",
00177     "RESERVED_BASE_ERROR",
00178     "RESERVED_BASE_ERROR"
00179 };
00180
00181 //
=====

```

```

00182
00183 // COMMON STRUCTS
00184 //
=====
00185
00186 struct LIBZMQUTILS_EXPORT HostClient
00187 {
00188     HostClient() = default;
00189
00190     HostClient(const HostClient&) = default;
00191
00192     HostClient(HostClient&&) = default;
00193
00194     HostClient& operator=(const HostClient&) = default;
00195
00196     HostClient& operator=(HostClient&&) = default;
00197
00198     HostClient(const std::string& ip, const std::string& name,
00199               const std::string& pid, const std::string& info = "");
00200
00201     // Struct members.
00202     std::string id;
00203     std::string ip;
00204     std::string hostname;
00205     std::string pid;
00206     std::string info;
00207     utils::SCTimePointStd last_connection;
00208 };
00209
00210 struct CommandRequest
00211 {
00212     CommandRequest():
00213         command(ServerCommand::INVALID_COMMAND),
00214         params(nullptr),
00215         params_size(0)
00216     {}
00217
00218     HostClient client;
00219     ServerCommand command;
00220     std::unique_ptr<std::uint8_t> params;
00221     zmq::multipart_t raw_msg;
00222     size_t params_size;
00223 };
00224
00225 struct CommandReply
00226 {
00227     CommandReply():
00228         params(nullptr),
00229         params_size(0),
00230         result(ServerResult::COMMAND_OK)
00231     {}
00232
00233     std::unique_ptr<std::uint8_t> params;
00234     zmq::multipart_t raw_msg;
00235     size_t params_size;
00236     ServerResult result;
00237 };
00238
00239 struct LIBZMQUTILS_EXPORT RequestData
00240 {
00241     RequestData(CommandType id) :
00242         command(id),
00243         params(nullptr),
00244         params_size(0){}
00245
00246     RequestData() :
00247         command(static_cast<CommandType>(ServerCommand::INVALID_COMMAND)),
00248         params(nullptr),
00249         params_size(0){}
00250
00251     CommandType command;
00252     std::unique_ptr<std::uint8_t> params;
00253     size_t params_size;
00254 };
00255
00256 //
=====
00257
00258 }} // END NAMESPACES.
00259 //
=====

```

7.15 examples/ExampleZMQCommanServerAmelas/AmelasExample↵ Controller/Utils.h File Reference

```
#include <string>
#include <map>
#include <vector>
#include <functional>
```

Namespaces

- namespace [amelas](#)
- namespace [amelas::utils](#)

Functions

- `template<typename ClassType, typename ReturnType, typename... Args>`
`static std::function< ReturnType(Args...)> amelas::utils::makeCallback (ClassType *object, Return↵`
`Type(ClassType::*memberFunction)(Args...))`

7.16 utils.h

[Go to the documentation of this file.](#)

```
00001
00002 //
=====
00003 #pragma once
00004 //
=====
00005
00006 // C++ INCLUDES
00007 //
=====
00008 #include <string>
00009 #include <map>
00010 #include <vector>
00011 #include <functional>
00012 //
=====
00013
00014 // AMELAS NAMESPACES
00015 //
=====
00016 namespace amelas{
00017 namespace utils{
00018 //
=====
00019
00020 template<typename ClassType, typename ReturnType, typename... Args>
00021 static std::function<ReturnType(Args...)> makeCallback(ClassType* object,
00022
00023     ReturnType(ClassType::*memberFunction)(Args...))
00024 {
00025     return [object, memberFunction](Args... args) -> ReturnType
00026     {
00027         return (object->*memberFunction)(std::forward<Args>(args)...);
00028     };
00029 }
00030 } // END NAMESPACES.
00031 //
```

7.17 includes/LibZMQUtils/Utils.h File Reference

This file contains the declaration of several utilities for the project development.

```
#include <algorithm>
#include <string>
#include <iostream>
#include <map>
```

```
#include <vector>
#include <cstring>
#include <chrono>
#include <array>
#include <utility>
#include "LibZMQUtils/libzmqutils_global.h"
```

Classes

- struct [zmqutils::utils::NetworkAdapterInfo](#)

Namespaces

- namespace [zmqutils](#)
- namespace [zmqutils::utils](#)
- namespace [zmqutils::utils::internal](#)

Macros

- #define [MKGMTIME](#) timegm

Typedefs

- using [zmqutils::utils::HRTIMEPOINTSTD](#) = std::chrono::time_point< std::chrono::high_resolution_clock >
High resolution time point to store datetimes (uses Unix Time).
- using [zmqutils::utils::SCTIMEPOINTSTD](#) = std::chrono::steady_clock::time_point
Steady clock time point for measuring intervals.

Functions

- [LIBZMQUTILS_EXPORT](#) void [zmqutils::utils::binarySerializeDeserialize](#) (const void *data, size_t data_size, void *_bytes, void *dest)
Binary serialization and deserialization.
- [LIBZMQUTILS_EXPORT](#) std::vector< [NetworkAdapterInfo](#) > [zmqutils::utils::getHostIPsWithInterfaces](#) ()
- [LIBZMQUTILS_EXPORT](#) std::string [zmqutils::utils::getHostname](#) ()
- [LIBZMQUTILS_EXPORT](#) unsigned [zmqutils::utils::getCurrentPID](#) ()
- [LIBZMQUTILS_EXPORT](#) std::string [zmqutils::utils::timePointToString](#) (const [HRTIMEPOINTSTD](#) &tp, const std::string &format="%Y-%m-%dT%H:%M:%S", bool add_ms=true, bool add_ns=false, bool utc=true)
- [LIBZMQUTILS_EXPORT](#) std::string [zmqutils::utils::timePointToISO8601](#) (const [HRTIMEPOINTSTD](#) &tp, bool add_ms=true, bool add_ns=false)
- [LIBZMQUTILS_EXPORT](#) std::string [zmqutils::utils::currentISO8601Date](#) (bool add_ms=true)
- template<typename T, std::size_t... Is1, std::size_t... Is2>
constexpr std::array< T, sizeof...(Is1)+sizeof...(Is2)> [zmqutils::utils::internal::joinArrays](#) (const std::array< T, sizeof...(Is1)> &a1, const std::array< T, sizeof...(Is2)> &a2, std::index_sequence< Is1... >, std::index_sequence< Is2... >)
- template<typename T, std::size_t N1, std::size_t N2>
constexpr std::array< T, N1+N2 > [zmqutils::utils::joinArraysConstexpr](#) (const std::array< T, N1 > &a1, const std::array< T, N2 > &a2)

7.17.1 Detailed Description

This file contains the declaration of several utilities for the project development.

Author

Degoras Project Team

Copyright

EUPL License

Version

2307.1

Definition in file [utils.h](#).

7.17.2 Macro Definition Documentation**7.17.2.1 MKGMTIME**

```
#define MKGMTIME timegm
```

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

7.18 utils.h

[Go to the documentation of this file.](#)

```
00001 /*****
00002  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
00003  *
00004  *   Copyright (C) 2023 Degoras Project Team
00005  *
00006  *   < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
00007  *
00008  *   < Jesús Relinque Madroñal >
00009  *
00010  *   This file is part of LibZMQUtils.
00011  *
00012  *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
00013  *   the EUPL license
00014  *   as soon they will be approved by the European Commission (IDABC).
00015  *
00016  *   This project is free software: you can redistribute it and/or modify it under the terms of the
00017  *   EUPL license as
00018  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
00019  *
00020  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
00021  *   or agreed to in
00022  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
00023  *   without even the
00024  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
00025  *   check specific
00026  *   language governing permissions and limitations and more details.
00027  *
00028  *   You should use this project in compliance with the EUPL license. You should have received a copy
00029  *   of the license
00030  *   along with this project. If not, see the license at < https://eupl.eu/ >.
00031  *
00032  *****/
00033 //
00034 #pragma once
00035 //
00036
00037 // C++ INCLUDES
00038 //
00039 #include <algorithm>
00040 #include <string>
00041 #include <iostream>
```



```

00042 #include <map>
00043 #include <vector>
00044 #include <cstring>
00045 #include <chrono>
00046 #include <array>
00047 #include <utility>
00048 //
=====
00049
00050 // ZMQUTILS INCLUDES
00051 //
=====
00052 #include "LibZMQUtils/libzmqutils_global.h"
00053 //
=====
00054
00055 // DEFINITIONS
00056 //
=====
00057 #if defined(__MINGW32__) || defined(_MSC_VER)
00058 #define MKGMTIME _mkgmtime
00059 #else
00060 #define MKGMTIME timegm
00061 #endif
00062 //
=====
00063
00064 // ZMQUTILS NAMESPACES
00065 //
=====
00066 namespace zmqutils{
00067 namespace utils{
00068 //
=====
00069
00070 // CONVENIENT ALIAS AND ENUMERATIONS
00071 //
=====
00072 using HRTimestampStd = std::chrono::time_point<std::chrono::high_resolution_clock>;
00073 using SCTimestampStd = std::chrono::steady_clock::time_point;
00074 //
=====
00075
00076 struct LIBZMQUTILS_EXPORT NetworkAdapterInfo
00077 {
00078     std::string id;
00079     std::string name;
00080     std::string descr;
00081     std::string ip;
00082 };
00083
00084 LIBZMQUTILS_EXPORT void binarySerializeDeserialize(const void* data, size_t data_size_bytes, void*
00085 dest);
00086
00087 LIBZMQUTILS_EXPORT std::vector<NetworkAdapterInfo> getHostIPsWithInterfaces();
00088
00089 LIBZMQUTILS_EXPORT std::string getHostname();
00090
00091 LIBZMQUTILS_EXPORT unsigned getCurrentPID();
00092
00093 LIBZMQUTILS_EXPORT std::string timePointToString(const HRTimestampStd& tp,
00094     const std::string& format = "%Y-%m-%dT%H:%M:%S",
00095     bool add_ms = true, bool add_ns = false, bool utc =
00096 true);
00097
00098 LIBZMQUTILS_EXPORT std::string timePointToIso8601(const HRTimestampStd& tp, bool add_ms = true, bool
00099 add_ns = false);
00100
00101 LIBZMQUTILS_EXPORT std::string currentISO8601Date(bool add_ms = true);
00102
00103 namespace internal
00104 {
00105     template <typename T, std::size_t... Is1, std::size_t... Is2>
00106     constexpr std::array<T, sizeof...(Is1) + sizeof...(Is2)>
00107     joinArrays(const std::array<T, sizeof...(Is1)>& a1, const std::array<T, sizeof...(Is2)>& a2,
00108         std::index_sequence<Is1...>, std::index_sequence<Is2...>)
00109     {
00110         return { a1[Is1]..., a2[Is2]... };
00111     }
00112 }
00113
00114 template <typename T, std::size_t N1, std::size_t N2>
00115 constexpr std::array<T, N1 + N2> joinArraysConstexpr(const std::array<T, N1>& a1, const std::array<T,
00116 N2>& a2)
00117 {
00118     return internal::joinArrays(a1, a2, std::make_index_sequence<N1>(),
00119         std::make_index_sequence<N2>());

```

```

00127 }
00128
00129 }} // END NAMESPACES.
00130 //

```

7.19 examples/ExampleZMQCommanServerAmelas/AmelasExample↵ Server/amelas_server.cpp File Reference

```
#include "amelas_server.h"
```

Namespaces

- namespace [amelas](#)

7.20 amelas_server.cpp

[Go to the documentation of this file.](#)

```

00001 #include "amelas_server.h"
00002
00003 // AMELAS NAMESPACES
00004 //
=====
00005 namespace amelas{
00006 //
=====
00007
00008 using common::AmelasServerCommandStr;
00009 using common::AmelasServerResultStr;
00010 using common::ControllerError;
00011 using common::AmelasServerCommand;
00012 using common::AmelasServerResult;
00013 using zmqutils::common::ServerCommand;
00014 using zmqutils::common::ServerResult;
00015 using zmqutils::common::ResultType;
00016
00017
00018 AmelasServer::AmelasServer(unsigned int port, const std::string &local_addr) :
00019     CommandServerBase(port, local_addr)
00020 {}
00021
00022 const std::map<AmelasServerCommand, common::ControllerCallback> &AmelasServer::getCallbackMap() const
00023 {
00024     return this->callback_map_;
00025 }
00026
00027 void AmelasServer::setCallback(common::AmelasServerCommand command, common::ControllerCallback
callback)
00028 {
00029     callback_map_[command] = callback;
00030 }
00031
00032 void AmelasServer::removeCallback(common::AmelasServerCommand command)
00033 {
00034     this->callback_map_.erase(command);
00035 }
00036
00037 bool AmelasServer::isCallbackSet(common::AmelasServerCommand command) const
00038 {
00039     return this->callback_map_.find(command) != this->callback_map_.end();
00040 }
00041
00042 void AmelasServer::clearCallbacks()
00043 {
00044     this->callback_map_.clear();
00045 }
00046
00047 void AmelasServer::processSetHomePosition(const CommandRequest& request, CommandReply& reply)
00048 {
00049     // Command and error.
00050     AmelasServerCommand cmd = AmelasServerCommand::REQ_SET_HOME_POSITION;
00051     ControllerError controller_err;
00052
00053     // Auxilar variables.
00054     double az, el;
00055     constexpr std::size_t double_sz = sizeof(double);

```

```

00056     bool result;
00057
00058     // Check the request parameters size.
00059     if (request.params_size == 0)
00060     {
00061         reply.result = ServerResult::EMPTY_PARAMS;
00062         return;
00063     }
00064     else if (request.params_size != double_sz*2)
00065     {
00066         reply.result = ServerResult::BAD_PARAMETERS;
00067         return;
00068     }
00069
00070     // Deserialize the parameters.
00071     zmqutils::utils::binarySerializeDeserialize(request.params.get(), double_sz, &az);
00072     zmqutils::utils::binarySerializeDeserialize(request.params.get() + double_sz, double_sz, &el);
00073
00074     // Generate the struct.
00075     common::AltAzPos pos = {az, el};
00076
00077     // Process the command.
00078     // Check the callback.
00079     if(!this->isCallbackSet(cmd))
00080     {
00081         reply.result = static_cast<ServerResult>(AmelasServerResult::EMPTY_CALLBACK);
00082         return;
00083     }
00084
00085     // Process the command.
00086     try{controller_err = this->invokeCallback<common::SetHomePositionCallback>(cmd, pos);}
00087     catch(...)
00088     {
00089         reply.result = static_cast<ServerResult>(AmelasServerResult::INVALID_CALLBACK);
00090         return;
00091     }
00092
00093     // Store the amelas error.
00094     reply.params = std::unique_ptr<std::uint8_t>(new std::uint8_t[sizeof(ResultType)]);
00095     ResultType amelas_res = static_cast<ResultType>(controller_err);
00096     zmqutils::utils::binarySerializeDeserialize(&amelas_res, sizeof(ResultType), reply.params.get());
00097     reply.params_size = sizeof(ResultType);
00098 }
00099
00100 void AmelasServer::processGetHomePosition(const CommandRequest &, CommandReply &reply)
00101 {
00102     // Command and error.
00103     AmelasServerCommand cmd = AmelasServerCommand::REQ_GET_HOME_POSITION;
00104     ControllerError controller_err;
00105
00106     // Auxiliar variables.
00107     constexpr std::size_t res_sz = sizeof(ControllerError);
00108     constexpr std::size_t double_sz = sizeof(double);
00109     ControllerError amelas_err = ControllerError::SUCCESS;
00110     common::AltAzPos pos;
00111
00112     // Process the command.
00113     try{controller_err = this->invokeCallback<common::GetHomePositionCallback>(cmd, pos);}
00114     catch(...)
00115     {
00116         reply.result = static_cast<ServerResult>(AmelasServerResult::INVALID_CALLBACK);
00117         return;
00118     }
00119
00120     // Serialize parameters
00121     reply.params = std::unique_ptr<std::uint8_t>(new std::uint8_t[res_sz + 2*double_sz]);
00122     reply.params_size = res_sz + 2*double_sz;
00123     zmqutils::utils::binarySerializeDeserialize(&amelas_err, res_sz, reply.params.get());
00124     zmqutils::utils::binarySerializeDeserialize(&pos.az, double_sz, reply.params.get() + res_sz);
00125     zmqutils::utils::binarySerializeDeserialize(&pos.el, double_sz, reply.params.get() + res_sz +
double_sz);
00126
00127     // Store the server result.
00128     reply.result = ServerResult::COMMAND_OK;
00129 }
00130
00131 void AmelasServer::processAmelasCommand(const CommandRequest& request, CommandReply& reply)
00132 {
00133     AmelasServerCommand command = static_cast<AmelasServerCommand>(request.command);
00134
00135     if(command == AmelasServerCommand::REQ_SET_HOME_POSITION)
00136     {
00137         this->processSetHomePosition(request, reply);
00138     }
00139     else if (command == AmelasServerCommand::REQ_GET_HOME_POSITION)
00140     {
00141         this->processGetHomePosition(request, reply);
00142     }

```

```

00142     }
00143     else
00144     {
00145         reply.result = ServerResult::NOT_IMPLEMENTED;
00146     }
00147 }
00148
00149 void AmelasServer::onCustomCommandReceived(const CommandRequest& request, CommandReply& reply)
00150 {
00151     // Get the command.
00152     AmelasServerCommand command = static_cast<AmelasServerCommand>(request.command);
00153
00154     // Get the command string.
00155     std::string cmd_str;
00156     std::uint32_t cmd_uint = static_cast<std::uint32_t>(request.command);
00157     cmd_str = (cmd_uint < AmelasServerCommandStr.size()) ? AmelasServerCommandStr[cmd_uint] : "Unknown
command";
00158
00159     // Log the command.
00160     std::cout << std::string(100, '-') << std::endl;
00161     std::cout<<"ON CUSTOM COMMAND RECEIVED: "<<std::endl;
00162     std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00163     std::cout<<"Client Id: "<<request.client.id<<std::endl;
00164     std::cout<<"Command: "<<cmd_uint<<" ("<<cmd_str<<")"<<std::endl;
00165     std::cout << std::string(100, '-') << std::endl;
00166
00167     // Process the command if it is implemented.
00168     if(command == AmelasServerCommand::END_AMELAS_COMMANDS)
00169     {
00170         // Update the result.
00171         reply.result = ServerResult::INVALID_MSG;
00172     }
00173     else if(AmelasServer::validateAmelasCommand(command))
00174     {
00175         this->processAmelasCommand(request, reply);
00176     }
00177     else
00178     {
00179         // Call to the base function.
00180         CommandServerBase::onCustomCommandReceived(request, reply);
00181     }
00182 }
00183
00184 void AmelasServer::onServerStart()
00185 {
00186     // Ips.
00187     std::string ips;
00188
00189     // Get listen interfaces ips.
00190     for(const auto& intrfc : this->getServerAddresses())
00191     {
00192         ips.append(intrfc.ip);
00193         ips.append(" - ");
00194     }
00195     ips.pop_back();
00196     ips.pop_back();
00197
00198     // Log.
00199     std::cout << std::string(100, '-') << std::endl;
00200     std::cout<<"<AMELAS SERVER>"<<std::endl;
00201     std::cout<<"-> ON SERVER START: "<<std::endl;
00202     std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00203     std::cout<<"Addresses: "<<ips<<std::endl;
00204     std::cout<<"Port: "<<this->getServerPort()<<std::endl;
00205     std::cout << std::string(100, '-') << std::endl;
00206 }
00207
00208 void AmelasServer::onServerStop()
00209 {
00210     // Log.
00211     std::cout << std::string(100, '-') << std::endl;
00212     std::cout<<"<AMELAS SERVER>"<<std::endl;
00213     std::cout<<"-> ON SERVER CLOSE: "<<std::endl;
00214     std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00215     std::cout << std::string(100, '-') << std::endl;
00216 }
00217
00218 void AmelasServer::onWaitingCommand()
00219 {
00220     // Log.
00221     std::cout << std::string(100, '-') << std::endl;
00222     std::cout<<"<AMELAS SERVER>"<<std::endl;
00223     std::cout<<"-> ON WAITING COMMAND: "<<std::endl;
00224     std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00225     std::cout << std::string(100, '-') << std::endl;
00226 }
00227

```

```

00228 void AmelasServer::onDeadClient(const HostClient& client)
00229 {
00230     // Log.
00231     std::cout << std::string(100, '-') << std::endl;
00232     std::cout<<"<AMELAS SERVER>"<<std::endl;
00233     std::cout<<"-> ON DEAD CLIENT: "<<std::endl;
00234     std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00235     std::cout<<"Current Clients: "<<this->getConnectedClients().size()<<std::endl;
00236     std::cout<<"Client Id: "<<client.id<<std::endl;
00237     std::cout<<"Client Ip: "<<client.ip<<std::endl;
00238     std::cout<<"Client Host: "<<client.hostname<<std::endl;
00239     std::cout<<"Client Process: "<<client.pid<<std::endl;
00240     std::cout << std::string(100, '-') << std::endl;
00241 }
00242
00243 void AmelasServer::onConnected(const HostClient& client)
00244 {
00245     // Log.
00246     std::cout << std::string(100, '-') << std::endl;
00247     std::cout<<"<AMELAS SERVER>"<<std::endl;
00248     std::cout<<"-> ON CONNECTED: "<<std::endl;
00249     std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00250     std::cout<<"Current Clients: "<<this->getConnectedClients().size()<<std::endl;
00251     std::cout<<"Client Id: "<<client.id<<std::endl;
00252     std::cout<<"Client Ip: "<<client.ip<<std::endl;
00253     std::cout<<"Client Host: "<<client.hostname<<std::endl;
00254     std::cout<<"Client Process: "<<client.pid<<std::endl;
00255     std::cout << std::string(100, '-') << std::endl;
00256 }
00257
00258 void AmelasServer::onDisconnected(const HostClient& client)
00259 {
00260     // Log.
00261     std::cout << std::string(100, '-') << std::endl;
00262     std::cout<<"<AMELAS SERVER>"<<std::endl;
00263     std::cout<<"-> ON DISCONNECTED: "<<std::endl;
00264     std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00265     std::cout<<"Current Clients: "<<this->getConnectedClients().size()<<std::endl;
00266     std::cout<<"Client Id: "<<client.id<<std::endl;
00267     std::cout<<"Client Ip: "<<client.ip<<std::endl;
00268     std::cout<<"Client Host: "<<client.hostname<<std::endl;
00269     std::cout<<"Client Process: "<<client.pid<<std::endl;
00270     std::cout << std::string(100, '-') << std::endl;
00271 }
00272
00273 void AmelasServer::onServerError(const zmq::error_t &error, const std::string &ext_info)
00274 {
00275     // Log.
00276     std::cout << std::string(100, '-') << std::endl;
00277     std::cout<<"<AMELAS SERVER>"<<std::endl;
00278     std::cout<<"-> ON SERVER ERROR: "<<std::endl;
00279     std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00280     std::cout<<"Code: "<<error.num()<<std::endl;
00281     std::cout<<"Error: "<<error.what()<<std::endl;
00282     std::cout<<"Info: "<<ext_info<<std::endl;
00283     std::cout << std::string(100, '-') << std::endl;
00284 }
00285
00286 void AmelasServer::onCommandReceived(const CommandRequest &cmd_req)
00287 {
00288     // Get the command string.
00289     std::string cmd_str;
00290     std::uint32_t command = static_cast<std::uint32_t>(cmd_req.command);
00291     cmd_str = (command < AmelasServerCommandStr.size()) ? AmelasServerCommandStr[command] : "Unknown
command";
00292     // Log.
00293     std::cout << std::string(100, '-') << std::endl;
00294     std::cout<<"<AMELAS SERVER>"<<std::endl;
00295     std::cout<<"-> ON COMMAND RECEIVED: "<<std::endl;
00296     std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00297     std::cout<<"Client Id: "<<cmd_req.client.id<<std::endl;
00298     std::cout<<"Command: "<<command<< ("<<cmd_str<<")"<<std::endl;
00299     std::cout << std::string(100, '-') << std::endl;
00300 }
00301
00302 void AmelasServer::onInvalidMsgReceived(const CommandRequest &cmd_req)
00303 {
00304     // Log.
00305     std::cout << std::string(100, '-') << std::endl;
00306     std::cout<<"<AMELAS SERVER>"<<std::endl;
00307     std::cout<<"-> ON BAD COMMAND RECEIVED: "<<std::endl;
00308     std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00309     std::cout<<"Raw Str: "<<cmd_req.raw_msg.str()<<std::endl;
00310     std::cout<<"Client Id: "<<cmd_req.client.id<<std::endl;
00311     std::cout<<"Client Ip: "<<cmd_req.client.ip<<std::endl;
00312     std::cout<<"Client Host: "<<cmd_req.client.hostname<<std::endl;
00313     std::cout<<"Client Process: "<<cmd_req.client.pid<<std::endl;

```

```

00314     std::cout<<"Command: "<<static_cast<int>(cmd_req.command)<<std::endl;
00315     std::cout<<"Params Size: "<<cmd_req.params_size<<std::endl;
00316     std::cout << std::string(100, '-') << std::endl;
00317 }
00318
00319 void AmelasServer::onSendingResponse(const CommandReply &cmd_rep)
00320 {
00321     // Log.
00322     int result = static_cast<int>(cmd_rep.result);
00323     std::cout << std::string(100, '-') << std::endl;
00324     std::cout<<"AMELAS SERVER"<<std::endl;
00325     std::cout<<"-> ON SENDING RESPONSE: "<<std::endl;
00326     std::cout<<"Time: "<<zmqutils::utils::currentISO8601Date()<<std::endl;
00327     std::cout<<"Result: "<<result<<" ("<<AmelasServerResultStr[result]<<")"<<std::endl;
00328     std::cout<<"Params Size: "<<cmd_rep.params_size<<std::endl;
00329     std::cout << std::string(100, '-') << std::endl;
00330 }
00331
00332 bool AmelasServer::validateAmelasCommand(AmelasServerCommand command)
00333 {
00334     // Auxiliar variables.
00335     bool result = false;
00336     zmqutils::common::CommandType cmd = static_cast<zmqutils::common::CommandType>(command);
00337     // Check if the command is within the range of implemented custom commands.
00338     if (cmd >= common::kMinCmdId && cmd <= common::kMaxCmdId)
00339         result = true;
00340     return result;
00341 }
00342
00343 } // END NAMESPACES.
00344 //
=====
00345

```

7.21 examples/ExampleZMQCommanServerAmelas/AmelasExample↵ Server/amelas_server.h File Reference

```

#include <unordered_map>
#include <string>
#include <any>
#include <variant>
#include <LibZMQUtils/CommandServer>
#include <LibZMQUtils/Utils>
#include "AmelasExampleController/common.h"
#include "AmelasExampleController/utils.h"
#include "common.h"

```

Classes

- class [amelas::AmelasServer](#)

Namespaces

- namespace [amelas](#)

7.22 amelas_server.h

[Go to the documentation of this file.](#)

```

00001 //
=====
00002 #pragma once
00003 //
=====
00004
00005 // C++ INCLUDES
00006 //
=====
00007 #include <unordered_map>
00008 #include <string>
00009 #include <any>
00010 #include <variant>

```

```

00011 //
00012 =====
00013 // ZMQUTILS INCLUDES
00014 //
00015 =====
00016 #include <LibZMQUtils/CommandServer>
00017 #include <LibZMQUtils/Utils>
00018 //
00019 =====
00020 // PROJECT INCLUDES
00021 //
00022 =====
00023 #include "AmelasExampleController/common.h"
00024 #include "AmelasExampleController/utils.h"
00025 #include "common.h"
00026 //
00027 =====
00028 namespace amelas{
00029 //
00030 =====
00031 using namespace zmqutils;
00032
00033 // Example of creating a command server from the base.
00034 class AmelasServer : public CommandServerBase
00035 {
00036 public:
00037
00038     AmelasServer(unsigned port, const std::string& local_addr = "*");
00039
00040     const std::map<common::AmelasServerCommand, common::ControllerCallback>& getCallbackMap() const;
00041
00042     void setCallback(common::AmelasServerCommand command, common::ControllerCallback callback);
00043
00044     template<typename ClassT = void, typename ReturnT = void, typename... Args>
00045     void setCallback(common::AmelasServerCommand command,
00046                     ClassT* object,
00047                     ReturnT(ClassT::*callback)(Args...))
00048     {
00049         callback_map_[command] = utils::makeCallback(object, callback);
00050     }
00051
00052     // Removes a callback for a command
00053     void removeCallback(common::AmelasServerCommand);
00054
00055     // Clears all the callbacks
00056     void clearCallbacks();
00057
00058     // Checks if a callback is set for a command
00059     bool isCallbackSet(common::AmelasServerCommand) const;
00060
00061 private:
00062
00063     template <typename CallbackType, typename... Args>
00064     common::ControllerError invokeCallback(common::AmelasServerCommand command, Args&&... args)
00065     {
00066         if (auto callback = std::get_if<CallbackType>(&callback_map_[command]))
00067         {
00068             return (*callback)(std::forward<Args>(args)...);
00069         }
00070         throw std::runtime_error("Invalid command or incorrect callback type");
00071     }
00072
00073     // Helper to check if the custom command is valid.
00074     static bool validateAmelasCommand(common::AmelasServerCommand command);
00075
00076     // Process the specific commands.
00077     void processAmelasCommand(const CommandRequest&, CommandReply&);
00078     void processSetHomePosition(const CommandRequest&, CommandReply&);
00079     void processGetHomePosition(const CommandRequest&, CommandReply&);
00080
00081     // Internal overridden custom command received callback.
00082     // WARNING The most important part.
00083     virtual void onCustomCommandReceived(const CommandRequest&, CommandReply&) final;
00084
00085     // Internal overridden start callback.
00086     virtual void onServerStart() final;
00087
00088     // Internal overridden close callback.
00089     virtual void onServerStop() final;
00090

```

```

00091     // Internal waiting command callback.
00092     virtual void onWaitingCommand() final;
00093
00094     // Internal dead client callback.
00095     virtual void onDeadClient(const HostClient&) final;
00096
00097     // Internal overridden connect callback.
00098     virtual void onConnected(const HostClient&) final;
00099
00100     // Internal overridden disconnect callback.
00101     virtual void onDisconnected(const HostClient&) final;
00102
00103     // Internal overridden command received callback.
00104     virtual void onCommandReceived(const CommandRequest&) final;
00105
00106     // Internal overridden bad command received callback.
00107     virtual void onInvalidMsgReceived(const CommandRequest&) final;
00108
00109     // Internal overridden sending response callback.
00110     virtual void onSendingResponse(const CommandReply&) final;
00111
00112     // Internal overridden server error callback.
00113     virtual void onServerError(const zmq::error_t&, const std::string& ext_info) final;
00114
00115     // External callbacks map.
00116     std::map<common::AmelasServerCommand, common::ControllerCallback> callback_map_;
00117 };
00118
00119 } // END NAMESPACES.
00120 //
=====

```

7.23 examples/ExampleZMQCommanServerAmelas/ExampleZMQServerAmelas.cpp File Reference

```

#include <iostream>
#include <chrono>
#include <thread>
#include <csignal>
#include <limits>
#include <LibZMQUtils/CommandServer>
#include <LibZMQUtils/Utils>
#include "AmelasExampleServer/amelas_server.h"
#include "AmelasExampleController/amelas_controller.h"

```

Functions

- int [main](#) (int argc, char **argv)

Variables

- volatile sig_atomic_t [gSignInterrupt](#) = 0
- std::condition_variable [gExitCv](#)
- std::mutex [gMtx](#)

7.23.1 Function Documentation

7.23.1.1 main()

```

int main (
    int argc,
    char ** argv )

```

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

7.23.2 Variable Documentation

7.23.2.1 gExitCv

std::condition_variable gExitCv

Definition at line 32 of file [ExampleZMQServerAmelas.cpp](#).

7.23.2.2 gMtx

std::mutex gMtx

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

7.23.2.3 gSignInterrupt

volatile sig_atomic_t gSignInterrupt = 0

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

7.24 ExampleZMQServerAmelas.cpp

[Go to the documentation of this file.](#)

```
00001
00002
00003 // C++ INCLUDES
00004 //
=====
00005 #ifdef _WIN32
00006 #include <Windows.h>
00007 #endif
00008 #include <iostream>
00009 #include <chrono>
00010 #include <thread>
00011 #include <csignal>
00012 #include <limits>
00013
00014 //
=====
00015
00016 // ZMQUTILS INCLUDES
00017 //
=====
00018 #include <LibZMQUtils/CommandServer>
00019 #include <LibZMQUtils/Utils>
00020 //
=====
00021
00022 // PROJECT INCLUDES
00023 //
=====
00024 #include "AmelasExampleServer/amelas_server.h"
00025 #include "AmelasExampleController/amelas_controller.h"
00026 //
=====
00027
00028 //
=====
00029
00030 // Global variables for safety ending.
00031 volatile sig_atomic_t gSignInterrupt = 0;
00032 std::condition_variable gExitCv;
00033 std::mutex gMtx;
00034
00035 // Signal handler for safety ending.
00036 #ifdef _WIN32
00037 BOOL WINAPI ConsoleCtrlHandler(DWORD dwCtrlType)
00038 {
00039     std::lock_guard<std::mutex> lock(gMtx);
00040     if (dwCtrlType == CTRL_C_EVENT || dwCtrlType == CTRL_BREAK_EVENT)
00041     {
00042         if (!gSignInterrupt)
00043         {
00044             gSignInterrupt = 1;
00045             gExitCv.notify_all();
00046         }
00047         return TRUE;
00048     }
00049     return FALSE;
00050 }
00051 #else
```

```

00052 // TODO
00053 #endif
00054
00055 //
-----
00056
00057 // Main function.
00058 //
00059 // In the main we will create an AmelasController and an AmelasServer that will
00060 // work together thanks to the callbacks. For safe finish, press ctrl-c.
00061 //
00062 int main(int argc, char**argv)
00063 {
00064     // Using.
00065     using amelas::common::AmelasServerCommand;
00066
00067     // Set up the Windows Console Control Handler
00068     SetConsoleCtrlHandler(ConsoleCtrlHandler, TRUE);
00069
00070     // Configuration variables.
00071     unsigned port = 9999;
00072     bool client_status_check = false;
00073
00074     // Get the port.
00075     if (argc == 2)
00076     {
00077         try
00078         {
00079             port = std::stoul(argv[1]);
00080         } catch (...)
00081         {
00082             std::cerr << "Not recognized port in input: " << argv[1] << std::endl;
00083             return -1;
00084         }
00085     }
00086
00087     else if (argc > 2)
00088     {
00089         std::cout << "Usage: ZMQServer [port]" << std::endl;
00090         return 0;
00091     }
00092
00093     // Instantiate the Amelas controller.
00094     amelas::AmelasController amelas_controller;
00095
00096     // Instantiate the server.
00097     amelas::AmelasServer amelas_server(port);
00098
00099     // Disable or enables the client status checking.
00100     amelas_server.setClientStatusCheck(client_status_check);
00101
00102     // -----
00103     // Set the controller callbacks in the server.
00104
00105     amelas_server.setCallback(AmelasServerCommand::REQ_SET_HOME_POSITION,
00106                               &amelas_controller,
00107                               &amelas::AmelasController::setHomePosition);
00108
00109     amelas_server.setCallback(AmelasServerCommand::REQ_GET_HOME_POSITION,
00110                               &amelas_controller,
00111                               &amelas::AmelasController::getHomePosition);
00112
00113     // -----
00114
00115     // Start the server.
00116     amelas_server.startServer();
00117
00118     // Use the condition variable as an infinite loop until ctrl-c.
00119     std::unique_lock<std::mutex> lock(gMtx);
00120     gExitCv.wait(lock, [] { return gSignInterrupt == 1; });
00121
00122     // Stop the server and wait the future.
00123     amelas_server.stopServer();
00124
00125     // Final log.
00126     std::cout << "Server stoped. Press Enter to exit!" << std::endl;
00127     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00128
00129     // Return.
00130     return 0;
00131 }
00132
00133 //
-----

```

7.25 external/zmq/includes/zmq/zmq.h File Reference

```
#include <errno.h>
#include <stddef.h>
#include <stdio.h>
#include <stdint.h>
```

Classes

- struct [zmq_msg_t](#)
- struct [zmq_pollitem_t](#)

Macros

- #define [ZMQ_VERSION_MAJOR](#) 4
- #define [ZMQ_VERSION_MINOR](#) 3
- #define [ZMQ_VERSION_PATCH](#) 4
- #define [ZMQ_MAKE_VERSION](#)(major, minor, patch) ((major) * 10000 + (minor) * 100 + (patch))
- #define [ZMQ_VERSION](#) [ZMQ_MAKE_VERSION](#) ([ZMQ_VERSION_MAJOR](#), [ZMQ_VERSION_MINOR](#), [ZMQ_VERSION_PATCH](#))
- #define [ZMQ_EXPORT](#)
- #define [ZMQ_DEFINED_STDINT](#) 1
- #define [ZMQ_HAUSNUMERO](#) 156384712
- #define [ENOTSUP](#) ([ZMQ_HAUSNUMERO](#) + 1)
- #define [EPROTONOSUPPORT](#) ([ZMQ_HAUSNUMERO](#) + 2)
- #define [ENOBUFS](#) ([ZMQ_HAUSNUMERO](#) + 3)
- #define [ENETDOWN](#) ([ZMQ_HAUSNUMERO](#) + 4)
- #define [EADDRINUSE](#) ([ZMQ_HAUSNUMERO](#) + 5)
- #define [EADDRNOTAVAIL](#) ([ZMQ_HAUSNUMERO](#) + 6)
- #define [ECONNREFUSED](#) ([ZMQ_HAUSNUMERO](#) + 7)
- #define [EINPROGRESS](#) ([ZMQ_HAUSNUMERO](#) + 8)
- #define [ENOTSOCK](#) ([ZMQ_HAUSNUMERO](#) + 9)
- #define [EMSGSIZE](#) ([ZMQ_HAUSNUMERO](#) + 10)
- #define [EAFNOSUPPORT](#) ([ZMQ_HAUSNUMERO](#) + 11)
- #define [ENETUNREACH](#) ([ZMQ_HAUSNUMERO](#) + 12)
- #define [ECONNABORTED](#) ([ZMQ_HAUSNUMERO](#) + 13)
- #define [ECONNRESET](#) ([ZMQ_HAUSNUMERO](#) + 14)
- #define [ENOTCONN](#) ([ZMQ_HAUSNUMERO](#) + 15)
- #define [ETIMEDOUT](#) ([ZMQ_HAUSNUMERO](#) + 16)
- #define [EHOSTUNREACH](#) ([ZMQ_HAUSNUMERO](#) + 17)
- #define [ENETRESET](#) ([ZMQ_HAUSNUMERO](#) + 18)
- #define [EFSM](#) ([ZMQ_HAUSNUMERO](#) + 51)
- #define [ENOCOMPATPROTO](#) ([ZMQ_HAUSNUMERO](#) + 52)
- #define [ETERM](#) ([ZMQ_HAUSNUMERO](#) + 53)
- #define [EMTHREAD](#) ([ZMQ_HAUSNUMERO](#) + 54)
- #define [ZMQ_IO_THREADS](#) 1
- #define [ZMQ_MAX_SOCKETS](#) 2
- #define [ZMQ_SOCKET_LIMIT](#) 3
- #define [ZMQ_THREAD_PRIORITY](#) 3
- #define [ZMQ_THREAD_SCHED_POLICY](#) 4
- #define [ZMQ_MAX_MSGSZ](#) 5
- #define [ZMQ_MSG_T_SIZE](#) 6
- #define [ZMQ_THREAD_AFFINITY_CPU_ADD](#) 7
- #define [ZMQ_THREAD_AFFINITY_CPU_REMOVE](#) 8
- #define [ZMQ_THREAD_NAME_PREFIX](#) 9

- `#define ZMQ_IO_THREADS_DFLT 1`
- `#define ZMQ_MAX_SOCKETS_DFLT 1023`
- `#define ZMQ_THREAD_PRIORITY_DFLT -1`
- `#define ZMQ_THREAD_SCHED_POLICY_DFLT -1`
- `#define ZMQ_PAIR 0`
- `#define ZMQ_PUB 1`
- `#define ZMQ_SUB 2`
- `#define ZMQ_REQ 3`
- `#define ZMQ_REP 4`
- `#define ZMQ_DEALER 5`
- `#define ZMQ_ROUTER 6`
- `#define ZMQ_PULL 7`
- `#define ZMQ_PUSH 8`
- `#define ZMQ_XPUB 9`
- `#define ZMQ_XSUB 10`
- `#define ZMQ_STREAM 11`
- `#define ZMQ_XREQ ZMQ_DEALER`
- `#define ZMQ_XREP ZMQ_ROUTER`
- `#define ZMQ_AFFINITY 4`
- `#define ZMQ_ROUTING_ID 5`
- `#define ZMQ_SUBSCRIBE 6`
- `#define ZMQ_UNSUBSCRIBE 7`
- `#define ZMQ_RATE 8`
- `#define ZMQ_RECOVERY_IVL 9`
- `#define ZMQ_SNDBUF 11`
- `#define ZMQ_RCVBUF 12`
- `#define ZMQ_RCVMORE 13`
- `#define ZMQ_FD 14`
- `#define ZMQ_EVENTS 15`
- `#define ZMQ_TYPE 16`
- `#define ZMQ_LINGER 17`
- `#define ZMQ_RECONNECT_IVL 18`
- `#define ZMQ_BACKLOG 19`
- `#define ZMQ_RECONNECT_IVL_MAX 21`
- `#define ZMQ_MAXMSGSIZE 22`
- `#define ZMQ_SNDHWM 23`
- `#define ZMQ_RCVHWM 24`
- `#define ZMQ_MULTICAST_HOPS 25`
- `#define ZMQ_RCVTIMEO 27`
- `#define ZMQ_SNDTIMEO 28`
- `#define ZMQ_LAST_ENDPOINT 32`
- `#define ZMQ_ROUTER_MANDATORY 33`
- `#define ZMQ_TCP_KEEPALIVE 34`
- `#define ZMQ_TCP_KEEPALIVE_CNT 35`
- `#define ZMQ_TCP_KEEPALIVE_IDLE 36`
- `#define ZMQ_TCP_KEEPALIVE_INTVL 37`
- `#define ZMQ_IMMEDIATE 39`
- `#define ZMQ_XPUB_VERBOSE 40`
- `#define ZMQ_ROUTER_RAW 41`
- `#define ZMQ_IPV6 42`
- `#define ZMQ_MECHANISM 43`
- `#define ZMQ_PLAIN_SERVER 44`
- `#define ZMQ_PLAIN_USERNAME 45`
- `#define ZMQ_PLAIN_PASSWORD 46`
- `#define ZMQ_CURVE_SERVER 47`

- #define [ZMQ_CURVE_PUBLICKEY](#) 48
- #define [ZMQ_CURVE_SECRETKEY](#) 49
- #define [ZMQ_CURVE_SERVERKEY](#) 50
- #define [ZMQ_PROBE_ROUTER](#) 51
- #define [ZMQ_REQ_CORRELATE](#) 52
- #define [ZMQ_REQ_RELAXED](#) 53
- #define [ZMQ_CONFLATE](#) 54
- #define [ZMQ_ZAP_DOMAIN](#) 55
- #define [ZMQ_ROUTER_HANDOVER](#) 56
- #define [ZMQ_TOS](#) 57
- #define [ZMQ_CONNECT_ROUTING_ID](#) 61
- #define [ZMQ_GSSAPI_SERVER](#) 62
- #define [ZMQ_GSSAPI_PRINCIPAL](#) 63
- #define [ZMQ_GSSAPI_SERVICE_PRINCIPAL](#) 64
- #define [ZMQ_GSSAPI_PLAINTEXT](#) 65
- #define [ZMQ_HANDSHAKE_IVL](#) 66
- #define [ZMQ SOCKS_PROXY](#) 68
- #define [ZMQ_XPUB_NODROP](#) 69
- #define [ZMQ_BLOCKY](#) 70
- #define [ZMQ_XPUB_MANUAL](#) 71
- #define [ZMQ_XPUB_WELCOME_MSG](#) 72
- #define [ZMQ_STREAM_NOTIFY](#) 73
- #define [ZMQ_INVERT_MATCHING](#) 74
- #define [ZMQ_HEARTBEAT_IVL](#) 75
- #define [ZMQ_HEARTBEAT_TTL](#) 76
- #define [ZMQ_HEARTBEAT_TIMEOUT](#) 77
- #define [ZMQ_XPUB_VERBOSE](#) 78
- #define [ZMQ_CONNECT_TIMEOUT](#) 79
- #define [ZMQ_TCP_MAXRT](#) 80
- #define [ZMQ_THREAD_SAFE](#) 81
- #define [ZMQ_MULTICAST_MAXTPDU](#) 84
- #define [ZMQ_VMCI_BUFFER_SIZE](#) 85
- #define [ZMQ_VMCI_BUFFER_MIN_SIZE](#) 86
- #define [ZMQ_VMCI_BUFFER_MAX_SIZE](#) 87
- #define [ZMQ_VMCI_CONNECT_TIMEOUT](#) 88
- #define [ZMQ_USE_FD](#) 89
- #define [ZMQ_GSSAPI_PRINCIPAL_NAME_TYPE](#) 90
- #define [ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAME_TYPE](#) 91
- #define [ZMQ_BINDTODEVICE](#) 92
- #define [ZMQ_MORE](#) 1
- #define [ZMQ_SHARED](#) 3
- #define [ZMQ_DONTWAIT](#) 1
- #define [ZMQ_SNDMORE](#) 2
- #define [ZMQ_NULL](#) 0
- #define [ZMQ_PLAIN](#) 1
- #define [ZMQ_CURVE](#) 2
- #define [ZMQ_GSSAPI](#) 3
- #define [ZMQ_GROUP_MAX_LENGTH](#) 255
- #define [ZMQ_IDENTITY](#) [ZMQ_ROUTING_ID](#)
- #define [ZMQ_CONNECT_RID](#) [ZMQ_CONNECT_ROUTING_ID](#)
- #define [ZMQ_TCP_ACCEPT_FILTER](#) 38
- #define [ZMQ_IPC_FILTER_PID](#) 58
- #define [ZMQ_IPC_FILTER_UID](#) 59
- #define [ZMQ_IPC_FILTER_GID](#) 60
- #define [ZMQ_IPV4ONLY](#) 31

- `#define ZMQ_DELAY_ATTACH_ON_CONNECT ZMQ_IMMEDIATE`
- `#define ZMQ_NOBLOCK ZMQ_DONTWAIT`
- `#define ZMQ_FAIL_UNROUTABLE ZMQ_ROUTER_MANDATORY`
- `#define ZMQ_ROUTER_BEHAVIOR ZMQ_ROUTER_MANDATORY`
- `#define ZMQ_SRCFD 2`
- `#define ZMQ_GSSAPI_NT_HOSTBASED 0`
- `#define ZMQ_GSSAPI_NT_USER_NAME 1`
- `#define ZMQ_GSSAPI_NT_KRB5_PRINCIPAL 2`
- `#define ZMQ_EVENT_CONNECTED 0x0001`
- `#define ZMQ_EVENT_CONNECT_DELAYED 0x0002`
- `#define ZMQ_EVENT_CONNECT_RETRIED 0x0004`
- `#define ZMQ_EVENT_LISTENING 0x0008`
- `#define ZMQ_EVENT_BIND_FAILED 0x0010`
- `#define ZMQ_EVENT_ACCEPTED 0x0020`
- `#define ZMQ_EVENT_ACCEPT_FAILED 0x0040`
- `#define ZMQ_EVENT_CLOSED 0x0080`
- `#define ZMQ_EVENT_CLOSE_FAILED 0x0100`
- `#define ZMQ_EVENT_DISCONNECTED 0x0200`
- `#define ZMQ_EVENT_MONITOR_STOPPED 0x0400`
- `#define ZMQ_EVENT_ALL 0xFFFF`
- `#define ZMQ_EVENT_HANDSHAKE_FAILED_NO_DETAIL 0x0800`
- `#define ZMQ_EVENT_HANDSHAKE_SUCCEEDED 0x1000`
- `#define ZMQ_EVENT_HANDSHAKE_FAILED_PROTOCOL 0x2000`
- `#define ZMQ_EVENT_HANDSHAKE_FAILED_AUTH 0x4000`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_UNSPECIFIED 0x10000000`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_UNEXPECTED_COMMAND 0x10000001`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_SEQUENCE 0x10000002`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_KEY_EXCHANGE 0x10000003`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_UNSPECIFIED 0x10000011`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_MESSAGE 0x10000012`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_HELLO 0x10000013`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_INITIATE 0x10000014`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_ERROR 0x10000015`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_READY 0x10000016`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_WELCOME 0x10000017`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_METADATA 0x10000018`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_CRYPTOGRAPHIC 0x11000001`
- `#define ZMQ_PROTOCOL_ERROR_ZMTP_MECHANISM_MISMATCH 0x11000002`
- `#define ZMQ_PROTOCOL_ERROR_ZAP_UNSPECIFIED 0x20000000`
- `#define ZMQ_PROTOCOL_ERROR_ZAP_MALFORMED_REPLY 0x20000001`
- `#define ZMQ_PROTOCOL_ERROR_ZAP_BAD_REQUEST_ID 0x20000002`
- `#define ZMQ_PROTOCOL_ERROR_ZAP_BAD_VERSION 0x20000003`
- `#define ZMQ_PROTOCOL_ERROR_ZAP_INVALID_STATUS_CODE 0x20000004`
- `#define ZMQ_PROTOCOL_ERROR_ZAP_INVALID_METADATA 0x20000005`
- `#define ZMQ_PROTOCOL_ERROR_WS_UNSPECIFIED 0x30000000`
- `#define ZMQ_POLLIN 1`
- `#define ZMQ_POLLOUT 2`
- `#define ZMQ_POLLERR 4`
- `#define ZMQ_POLLPRI 8`
- `#define ZMQ_POLLITEMS_DFLT 16`
- `#define ZMQ_HAS_CAPABILITIES 1`
- `#define ZMQ_STREAMER 1`
- `#define ZMQ_FORWARDER 2`
- `#define ZMQ_QUEUE 3`
- `#define ZMQ_HAVE_TIMERS`

Typedefs

- typedef struct [zmq_msg_t](#) [zmq_msg_t](#)
- typedef void() [zmq_free_fn](#)(void *data_, void *hint_)
- typedef int [zmq_fd_t](#)
- typedef struct [zmq_pollitem_t](#) [zmq_pollitem_t](#)
- typedef void() [zmq_timer_fn](#)(int timer_id, void *arg)
- typedef void() [zmq_thread_fn](#)(void *)

Functions

- [ZMQ_EXPORT](#) int [zmq_errno](#) (void)
- [ZMQ_EXPORT](#) const char * [zmq_strerror](#) (int errnum_)
- [ZMQ_EXPORT](#) void [zmq_version](#) (int *major_, int *minor_, int *patch_)
- [ZMQ_EXPORT](#) void * [zmq_ctx_new](#) (void)
- [ZMQ_EXPORT](#) int [zmq_ctx_term](#) (void *context_)
- [ZMQ_EXPORT](#) int [zmq_ctx_shutdown](#) (void *context_)
- [ZMQ_EXPORT](#) int [zmq_ctx_set](#) (void *context_, int option_, int optval_)
- [ZMQ_EXPORT](#) int [zmq_ctx_get](#) (void *context_, int option_)
- [ZMQ_EXPORT](#) void * [zmq_init](#) (int io_threads_)
- [ZMQ_EXPORT](#) int [zmq_term](#) (void *context_)
- [ZMQ_EXPORT](#) int [zmq_ctx_destroy](#) (void *context_)
- [ZMQ_EXPORT](#) int [zmq_msg_init](#) ([zmq_msg_t](#) *msg_)
- [ZMQ_EXPORT](#) int [zmq_msg_init_size](#) ([zmq_msg_t](#) *msg_, size_t size_)
- [ZMQ_EXPORT](#) int [zmq_msg_init_data](#) ([zmq_msg_t](#) *msg_, void *data_, size_t size_, [zmq_free_fn](#) *ffn_, void *hint_)
- [ZMQ_EXPORT](#) int [zmq_msg_send](#) ([zmq_msg_t](#) *msg_, void *s_, int flags_)
- [ZMQ_EXPORT](#) int [zmq_msg_recv](#) ([zmq_msg_t](#) *msg_, void *s_, int flags_)
- [ZMQ_EXPORT](#) int [zmq_msg_close](#) ([zmq_msg_t](#) *msg_)
- [ZMQ_EXPORT](#) int [zmq_msg_move](#) ([zmq_msg_t](#) *dest_, [zmq_msg_t](#) *src_)
- [ZMQ_EXPORT](#) int [zmq_msg_copy](#) ([zmq_msg_t](#) *dest_, [zmq_msg_t](#) *src_)
- [ZMQ_EXPORT](#) void * [zmq_msg_data](#) ([zmq_msg_t](#) *msg_)
- [ZMQ_EXPORT](#) size_t [zmq_msg_size](#) (const [zmq_msg_t](#) *msg_)
- [ZMQ_EXPORT](#) int [zmq_msg_more](#) (const [zmq_msg_t](#) *msg_)
- [ZMQ_EXPORT](#) int [zmq_msg_get](#) (const [zmq_msg_t](#) *msg_, int property_)
- [ZMQ_EXPORT](#) int [zmq_msg_set](#) ([zmq_msg_t](#) *msg_, int property_, int optval_)
- [ZMQ_EXPORT](#) const char * [zmq_msg_gets](#) (const [zmq_msg_t](#) *msg_, const char *property_)
- [ZMQ_EXPORT](#) void * [zmq_socket](#) (void *, int type_)
- [ZMQ_EXPORT](#) int [zmq_close](#) (void *s_)
- [ZMQ_EXPORT](#) int [zmq_setsockopt](#) (void *s_, int option_, const void *optval_, size_t optvallen_)
- [ZMQ_EXPORT](#) int [zmq_getsockopt](#) (void *s_, int option_, void *optval_, size_t *optvallen_)
- [ZMQ_EXPORT](#) int [zmq_bind](#) (void *s_, const char *addr_)
- [ZMQ_EXPORT](#) int [zmq_connect](#) (void *s_, const char *addr_)
- [ZMQ_EXPORT](#) int [zmq_unbind](#) (void *s_, const char *addr_)
- [ZMQ_EXPORT](#) int [zmq_disconnect](#) (void *s_, const char *addr_)
- [ZMQ_EXPORT](#) int [zmq_send](#) (void *s_, const void *buf_, size_t len_, int flags_)
- [ZMQ_EXPORT](#) int [zmq_send_const](#) (void *s_, const void *buf_, size_t len_, int flags_)
- [ZMQ_EXPORT](#) int [zmq_recv](#) (void *s_, void *buf_, size_t len_, int flags_)
- [ZMQ_EXPORT](#) int [zmq_socket_monitor](#) (void *s_, const char *addr_, int events_)
- [ZMQ_EXPORT](#) int [zmq_poll](#) ([zmq_pollitem_t](#) *items_, int nitems_, long timeout_)
- [ZMQ_EXPORT](#) int [zmq_proxy](#) (void *frontend_, void *backend_, void *capture_)
- [ZMQ_EXPORT](#) int [zmq_proxy_steerable](#) (void *frontend_, void *backend_, void *capture_, void *control_)
- [ZMQ_EXPORT](#) int [zmq_has](#) (const char *capability_)
- [ZMQ_EXPORT](#) int [zmq_device](#) (int type_, void *frontend_, void *backend_)
- [ZMQ_EXPORT](#) int [zmq_sendmsg](#) (void *s_, [zmq_msg_t](#) *msg_, int flags_)

- [ZMQ_EXPORT](#) int [zmq_recvmsg](#) (void *s_, [zmq_msg_t](#) *msg_, int flags_)
- [ZMQ_EXPORT](#) int [zmq_sendiov](#) (void *s_, struct iovec *iov_, size_t count_, int flags_)
- [ZMQ_EXPORT](#) int [zmq_recviov](#) (void *s_, struct iovec *iov_, size_t *count_, int flags_)
- [ZMQ_EXPORT](#) char * [zmq_z85_encode](#) (char *dest_, const uint8_t *data_, size_t size_)
- [ZMQ_EXPORT](#) uint8_t * [zmq_z85_decode](#) (uint8_t *dest_, const char *string_)
- [ZMQ_EXPORT](#) int [zmq_curve_keypair](#) (char *z85_public_key_, char *z85_secret_key_)
- [ZMQ_EXPORT](#) int [zmq_curve_public](#) (char *z85_public_key_, const char *z85_secret_key_)
- [ZMQ_EXPORT](#) void * [zmq_atomic_counter_new](#) (void)
- [ZMQ_EXPORT](#) void [zmq_atomic_counter_set](#) (void *counter_, int value_)
- [ZMQ_EXPORT](#) int [zmq_atomic_counter_inc](#) (void *counter_)
- [ZMQ_EXPORT](#) int [zmq_atomic_counter_dec](#) (void *counter_)
- [ZMQ_EXPORT](#) int [zmq_atomic_counter_value](#) (void *counter_)
- [ZMQ_EXPORT](#) void [zmq_atomic_counter_destroy](#) (void **counter_p_)
- [ZMQ_EXPORT](#) void * [zmq_timers_new](#) (void)
- [ZMQ_EXPORT](#) int [zmq_timers_destroy](#) (void **timers_p)
- [ZMQ_EXPORT](#) int [zmq_timers_add](#) (void *timers, size_t interval, [zmq_timer_fn](#) handler, void *arg)
- [ZMQ_EXPORT](#) int [zmq_timers_cancel](#) (void *timers, int timer_id)
- [ZMQ_EXPORT](#) int [zmq_timers_set_interval](#) (void *timers, int timer_id, size_t interval)
- [ZMQ_EXPORT](#) int [zmq_timers_reset](#) (void *timers, int timer_id)
- [ZMQ_EXPORT](#) long [zmq_timers_timeout](#) (void *timers)
- [ZMQ_EXPORT](#) int [zmq_timers_execute](#) (void *timers)
- [ZMQ_EXPORT](#) void * [zmq_stopwatch_start](#) (void)
- [ZMQ_EXPORT](#) unsigned long [zmq_stopwatch_intermediate](#) (void *watch_)
- [ZMQ_EXPORT](#) unsigned long [zmq_stopwatch_stop](#) (void *watch_)
- [ZMQ_EXPORT](#) void [zmq_sleep](#) (int seconds_)
- [ZMQ_EXPORT](#) void * [zmq_threadstart](#) ([zmq_thread_fn](#) *func_, void *arg_)
- [ZMQ_EXPORT](#) void [zmq_threadclose](#) (void *thread_)

7.25.1 Macro Definition Documentation

7.25.1.1 EADDRINUSE

```
#define EADDRINUSE (ZMQ_HAUSNUMERO + 5)
```

Definition at line 149 of file [zmq.h](#).

7.25.1.2 EADDRNOTAVAIL

```
#define EADDRNOTAVAIL (ZMQ_HAUSNUMERO + 6)
```

Definition at line 152 of file [zmq.h](#).

7.25.1.3 EAFNOSUPPORT

```
#define EAFNOSUPPORT (ZMQ_HAUSNUMERO + 11)
```

Definition at line 167 of file [zmq.h](#).

7.25.1.4 ECONNABORTED

```
#define ECONNABORTED (ZMQ_HAUSNUMERO + 13)
```

Definition at line 173 of file [zmq.h](#).

7.25.1.5 ECONNREFUSED

```
#define ECONNREFUSED (ZMQ_HAUSNUMERO + 7)
```

Definition at line 155 of file [zmq.h](#).

7.25.1.6 ECONNRESET

```
#define ECONNRESET (ZMQ_HAUSNUMERO + 14)
```

Definition at line 176 of file [zmq.h](#).

7.25.1.7 EFSM

```
#define EFSM (ZMQ_HAUSNUMERO + 51)
```

Definition at line 192 of file [zmq.h](#).

7.25.1.8 EHOSTUNREACH

```
#define EHOSTUNREACH (ZMQ_HAUSNUMERO + 17)
```

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

7.25.1.9 EINPROGRESS

```
#define EINPROGRESS (ZMQ_HAUSNUMERO + 8)
```

Definition at line 158 of file [zmq.h](#).

7.25.1.10 EMSGSIZE

```
#define EMSGSIZE (ZMQ_HAUSNUMERO + 10)
```

Definition at line 164 of file [zmq.h](#).

7.25.1.11 EMTHREAD

```
#define EMTHREAD (ZMQ_HAUSNUMERO + 54)
```

Definition at line 195 of file [zmq.h](#).

7.25.1.12 ENETDOWN

```
#define ENETDOWN (ZMQ_HAUSNUMERO + 4)
```

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

7.25.1.13 ENETRESET

```
#define ENETRESET (ZMQ_HAUSNUMERO + 18)
```

Definition at line 188 of file [zmq.h](#).

7.25.1.14 ENETUNREACH

```
#define ENETUNREACH (ZMQ_HAUSNUMERO + 12)
```

Definition at line 170 of file [zmq.h](#).

7.25.1.15 ENOBUFS

```
#define ENOBUFS (ZMQ_HAUSNUMERO + 3)
```

Definition at line 143 of file [zmq.h](#).

7.25.1.16 ENOCOMPATPROTO

```
#define ENOCOMPATPROTO (ZMQ_HAUSNUMERO + 52)
```

Definition at line 193 of file [zmq.h](#).

7.25.1.17 ENOTCONN

```
#define ENOTCONN (ZMQ_HAUSNUMERO + 15)
```

Definition at line 179 of file [zmq.h](#).

7.25.1.18 ENOTSOCK

```
#define ENOTSOCK (ZMQ_HAUSNUMERO + 9)
```

Definition at line 161 of file [zmq.h](#).

7.25.1.19 ENOTSUP

```
#define ENOTSUP (ZMQ_HAUSNUMERO + 1)
```

Definition at line 137 of file [zmq.h](#).

7.25.1.20 EPROTONOSUPPORT

```
#define EPROTONOSUPPORT (ZMQ_HAUSNUMERO + 2)
```

Definition at line 140 of file [zmq.h](#).

7.25.1.21 ETERM

```
#define ETERM (ZMQ_HAUSNUMERO + 53)
```

Definition at line 194 of file [zmq.h](#).

7.25.1.22 ETIMEDOUT

```
#define ETIMEDOUT (ZMQ_HAUSNUMERO + 16)
```

Definition at line 182 of file [zmq.h](#).

7.25.1.23 ZMQ_AFFINITY

```
#define ZMQ_AFFINITY 4
```

Definition at line 309 of file [zmq.h](#).

7.25.1.24 ZMQ_BACKLOG

```
#define ZMQ_BACKLOG 19
```

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

7.25.1.25 ZMQ_BINDTODEVICE

```
#define ZMQ_BINDTODEVICE 92
```

Definition at line 384 of file [zmq.h](#).

7.25.1.26 ZMQ_BLOCKY

```
#define ZMQ_BLOCKY 70
```

Definition at line 364 of file [zmq.h](#).

7.25.1.27 ZMQ_CONFLATE

```
#define ZMQ_CONFLATE 54
```

Definition at line 352 of file [zmq.h](#).

7.25.1.28 ZMQ_CONNECT_RID

```
#define ZMQ_CONNECT_RID ZMQ_CONNECT_ROUTING_ID
```

Definition at line 405 of file [zmq.h](#).

7.25.1.29 ZMQ_CONNECT_ROUTING_ID

```
#define ZMQ_CONNECT_ROUTING_ID 61
```

Definition at line 356 of file [zmq.h](#).

7.25.1.30 ZMQ_CONNECT_TIMEOUT

```
#define ZMQ_CONNECT_TIMEOUT 79
```

Definition at line 373 of file [zmq.h](#).

7.25.1.31 ZMQ_CURVE

```
#define ZMQ_CURVE 2
```

Definition at line 397 of file [zmq.h](#).

7.25.1.32 ZMQ_CURVE_PUBLICKEY

```
#define ZMQ_CURVE_PUBLICKEY 48
```

Definition at line 346 of file [zmq.h](#).

7.25.1.33 ZMQ_CURVE_SECRETKEY

```
#define ZMQ_CURVE_SECRETKEY 49
```

Definition at line 347 of file [zmq.h](#).

7.25.1.34 ZMQ_CURVE_SERVER

```
#define ZMQ_CURVE_SERVER 47
```

Definition at line 345 of file [zmq.h](#).

7.25.1.35 ZMQ_CURVE_SERVERKEY

```
#define ZMQ_CURVE_SERVERKEY 50
```

Definition at line 348 of file [zmq.h](#).

7.25.1.36 ZMQ_DEALER

```
#define ZMQ_DEALER 5
```

Definition at line 296 of file [zmq.h](#).

7.25.1.37 ZMQ_DEFINED_STDINT

```
#define ZMQ_DEFINED_STDINT 1
```

Definition at line 96 of file [zmq.h](#).

7.25.1.38 ZMQ_DELAY_ATTACH_ON_CONNECT

```
#define ZMQ_DELAY_ATTACH_ON_CONNECT ZMQ_IMMEDIATE
```

Definition at line 411 of file [zmq.h](#).

7.25.1.39 ZMQ_DONTWAIT

```
#define ZMQ_DONTWAIT 1
```

Definition at line 391 of file [zmq.h](#).

7.25.1.40 ZMQ_EVENT_ACCEPT_FAILED

```
#define ZMQ_EVENT_ACCEPT_FAILED 0x0040
```

Definition at line 440 of file [zmq.h](#).

7.25.1.41 ZMQ_EVENT_ACCEPTED

```
#define ZMQ_EVENT_ACCEPTED 0x0020
```

Definition at line 439 of file [zmq.h](#).

7.25.1.42 ZMQ_EVENT_ALL

```
#define ZMQ_EVENT_ALL 0xFFFF
```

Definition at line 445 of file [zmq.h](#).

7.25.1.43 ZMQ_EVENT_BIND_FAILED

```
#define ZMQ_EVENT_BIND_FAILED 0x0010
```

Definition at line 438 of file [zmq.h](#).

7.25.1.44 ZMQ_EVENT_CLOSE_FAILED

```
#define ZMQ_EVENT_CLOSE_FAILED 0x0100
```

Definition at line 442 of file [zmq.h](#).

7.25.1.45 ZMQ_EVENT_CLOSED

```
#define ZMQ_EVENT_CLOSED 0x0080
```

Definition at line 441 of file [zmq.h](#).

7.25.1.46 ZMQ_EVENT_CONNECT_DELAYED

```
#define ZMQ_EVENT_CONNECT_DELAYED 0x0002
```

Definition at line 435 of file [zmq.h](#).

7.25.1.47 ZMQ_EVENT_CONNECT_RETRIED

```
#define ZMQ_EVENT_CONNECT_RETRIED 0x0004
```

Definition at line 436 of file [zmq.h](#).

7.25.1.48 ZMQ_EVENT_CONNECTED

```
#define ZMQ_EVENT_CONNECTED 0x0001
```

Definition at line 434 of file [zmq.h](#).

7.25.1.49 ZMQ_EVENT_DISCONNECTED

```
#define ZMQ_EVENT_DISCONNECTED 0x0200
```

Definition at line 443 of file [zmq.h](#).

7.25.1.50 ZMQ_EVENT_HANDSHAKE_FAILED_AUTH

```
#define ZMQ_EVENT_HANDSHAKE_FAILED_AUTH 0x4000
```

Definition at line 456 of file [zmq.h](#).

7.25.1.51 ZMQ_EVENT_HANDSHAKE_FAILED_NO_DETAIL

```
#define ZMQ_EVENT_HANDSHAKE_FAILED_NO_DETAIL 0x0800
```

Definition at line 447 of file [zmq.h](#).

7.25.1.52 ZMQ_EVENT_HANDSHAKE_FAILED_PROTOCOL

```
#define ZMQ_EVENT_HANDSHAKE_FAILED_PROTOCOL 0x2000
```

Definition at line 453 of file [zmq.h](#).

7.25.1.53 ZMQ_EVENT_HANDSHAKE_SUCCEEDED

```
#define ZMQ_EVENT_HANDSHAKE_SUCCEEDED 0x1000
```

Definition at line 450 of file [zmq.h](#).

7.25.1.54 ZMQ_EVENT_LISTENING

```
#define ZMQ_EVENT_LISTENING 0x0008
```

Definition at line 437 of file [zmq.h](#).

7.25.1.55 ZMQ_EVENT_MONITOR_STOPPED

```
#define ZMQ_EVENT_MONITOR_STOPPED 0x0400
```

Definition at line 444 of file [zmq.h](#).

7.25.1.56 ZMQ_EVENTS

```
#define ZMQ_EVENTS 15
```

Definition at line 319 of file [zmq.h](#).

7.25.1.57 ZMQ_EXPORT

```
#define ZMQ_EXPORT
```

Definition at line 91 of file [zmq.h](#).

7.25.1.58 ZMQ_FAIL_UNROUTABLE

```
#define ZMQ_FAIL_UNROUTABLE ZMQ_ROUTER_MANDATORY
```

Definition at line 413 of file [zmq.h](#).

7.25.1.59 ZMQ_FD

```
#define ZMQ_FD 14
```

Definition at line 318 of file [zmq.h](#).

7.25.1.60 ZMQ_FORWARDER

```
#define ZMQ_FORWARDER 2
```

Definition at line 551 of file [zmq.h](#).

7.25.1.61 ZMQ_GROUP_MAX_LENGTH

```
#define ZMQ_GROUP_MAX_LENGTH 255
```

Definition at line 401 of file [zmq.h](#).

7.25.1.62 ZMQ_GSSAPI

```
#define ZMQ_GSSAPI 3
```

Definition at line 398 of file [zmq.h](#).

7.25.1.63 ZMQ_GSSAPI_NT_HOSTBASED

```
#define ZMQ_GSSAPI_NT_HOSTBASED 0
```

Definition at line 424 of file [zmq.h](#).

7.25.1.64 ZMQ_GSSAPI_NT_KRB5_PRINCIPAL

```
#define ZMQ_GSSAPI_NT_KRB5_PRINCIPAL 2
```

Definition at line 426 of file [zmq.h](#).

7.25.1.65 ZMQ_GSSAPI_NT_USER_NAME

```
#define ZMQ_GSSAPI_NT_USER_NAME 1
```

Definition at line 425 of file [zmq.h](#).

7.25.1.66 ZMQ_GSSAPI_PLAINTEXT

```
#define ZMQ_GSSAPI_PLAINTEXT 65
```

Definition at line 360 of file [zmq.h](#).

7.25.1.67 ZMQ_GSSAPI_PRINCIPAL

```
#define ZMQ_GSSAPI_PRINCIPAL 63
```

Definition at line 358 of file [zmq.h](#).

7.25.1.68 ZMQ_GSSAPI_PRINCIPAL_NAMETYPE

```
#define ZMQ_GSSAPI_PRINCIPAL_NAMETYPE 90
```

Definition at line 382 of file [zmq.h](#).

7.25.1.69 ZMQ_GSSAPI_SERVER

```
#define ZMQ_GSSAPI_SERVER 62
```

Definition at line 357 of file [zmq.h](#).

7.25.1.70 ZMQ_GSSAPI_SERVICE_PRINCIPAL

```
#define ZMQ_GSSAPI_SERVICE_PRINCIPAL 64
```

Definition at line 359 of file [zmq.h](#).

7.25.1.71 ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAMETYPE

```
#define ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAMETYPE 91
```

Definition at line 383 of file [zmq.h](#).

7.25.1.72 ZMQ_HANDSHAKE_IVL

```
#define ZMQ_HANDSHAKE_IVL 66
```

Definition at line 361 of file [zmq.h](#).

7.25.1.73 ZMQ_HAS_CAPABILITIES

```
#define ZMQ_HAS_CAPABILITIES 1
```

Definition at line 546 of file [zmq.h](#).

7.25.1.74 ZMQ_HAUSNUMERO

```
#define ZMQ_HAUSNUMERO 156384712
```

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

7.25.1.75 ZMQ_HAVE_TIMERS

```
#define ZMQ_HAVE_TIMERS
```

Definition at line 599 of file [zmq.h](#).

7.25.1.76 ZMQ_HEARTBEAT_IVL

```
#define ZMQ_HEARTBEAT_IVL 75
```

Definition at line 369 of file [zmq.h](#).

7.25.1.77 ZMQ_HEARTBEAT_TIMEOUT

```
#define ZMQ_HEARTBEAT_TIMEOUT 77
```

Definition at line 371 of file [zmq.h](#).

7.25.1.78 ZMQ_HEARTBEAT_TTL

```
#define ZMQ_HEARTBEAT_TTL 76
```

Definition at line 370 of file [zmq.h](#).

7.25.1.79 ZMQ_IDENTITY

```
#define ZMQ_IDENTITY ZMQ_ROUTING_ID
```

Definition at line 404 of file [zmq.h](#).

7.25.1.80 ZMQ_IMMEDIATE

```
#define ZMQ_IMMEDIATE 39
```

Definition at line 337 of file [zmq.h](#).

7.25.1.81 ZMQ_INVERT_MATCHING

```
#define ZMQ_INVERT_MATCHING 74
```

Definition at line 368 of file [zmq.h](#).

7.25.1.82 ZMQ_IO_THREADS

```
#define ZMQ_IO_THREADS 1
```

Definition at line 214 of file [zmq.h](#).

7.25.1.83 ZMQ_IO_THREADS_DFLT

```
#define ZMQ_IO_THREADS_DFLT 1
```

Definition at line 226 of file [zmq.h](#).

7.25.1.84 ZMQ_IPC_FILTER_GID

```
#define ZMQ_IPC_FILTER_GID 60
```

Definition at line 409 of file [zmq.h](#).

7.25.1.85 ZMQ_IPC_FILTER_PID

```
#define ZMQ_IPC_FILTER_PID 58
```

Definition at line 407 of file [zmq.h](#).

7.25.1.86 ZMQ_IPC_FILTER_UID

```
#define ZMQ_IPC_FILTER_UID 59
```

Definition at line 408 of file [zmq.h](#).

7.25.1.87 ZMQ_IPV4ONLY

```
#define ZMQ_IPV4ONLY 31
```

Definition at line 410 of file [zmq.h](#).

7.25.1.88 ZMQ_IPV6

```
#define ZMQ_IPV6 42
```

Definition at line 340 of file [zmq.h](#).

7.25.1.89 ZMQ_LAST_ENDPOINT

```
#define ZMQ_LAST_ENDPOINT 32
```

Definition at line 331 of file [zmq.h](#).

7.25.1.90 ZMQ_LINGER

```
#define ZMQ_LINGER 17
```

Definition at line 321 of file [zmq.h](#).

7.25.1.91 ZMQ_MAKE_VERSION

```
#define ZMQ_MAKE_VERSION(  
    major,  
    minor,  
    patch ) ((major) *10000 + (minor) *100 + (patch))
```

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

7.25.1.92 ZMQ_MAX_MSGSZ

```
#define ZMQ_MAX_MSGSZ 5
```

Definition at line 219 of file [zmq.h](#).

7.25.1.93 ZMQ_MAX_SOCKETS

```
#define ZMQ_MAX_SOCKETS 2
```

Definition at line 215 of file [zmq.h](#).

7.25.1.94 ZMQ_MAX_SOCKETS_DFLT

```
#define ZMQ_MAX_SOCKETS_DFLT 1023
```

Definition at line 227 of file [zmq.h](#).

7.25.1.95 ZMQ_MAXMSGSIZE

```
#define ZMQ_MAXMSGSIZE 22
```

Definition at line 325 of file [zmq.h](#).

7.25.1.96 ZMQ_MECHANISM

```
#define ZMQ_MECHANISM 43
```

Definition at line 341 of file [zmq.h](#).

7.25.1.97 ZMQ_MORE

```
#define ZMQ_MORE 1
```

Definition at line 387 of file [zmq.h](#).

7.25.1.98 ZMQ_MSG_T_SIZE

```
#define ZMQ_MSG_T_SIZE 6
```

Definition at line 220 of file [zmq.h](#).

7.25.1.99 ZMQ_MULTICAST_HOPS

```
#define ZMQ_MULTICAST_HOPS 25
```

Definition at line 328 of file [zmq.h](#).

7.25.1.100 ZMQ_MULTICAST_MAXTPDU

```
#define ZMQ_MULTICAST_MAXTPDU 84
```

Definition at line 376 of file [zmq.h](#).

7.25.1.101 ZMQ_NOBLOCK

```
#define ZMQ_NOBLOCK ZMQ_DONTWAIT
```

Definition at line 412 of file [zmq.h](#).

7.25.1.102 ZMQ_NULL

```
#define ZMQ_NULL 0
```

Definition at line 395 of file [zmq.h](#).

7.25.1.103 ZMQ_PAIR

```
#define ZMQ_PAIR 0
```

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

7.25.1.104 ZMQ_PLAIN

```
#define ZMQ_PLAIN 1
```

Definition at line 396 of file [zmq.h](#).

7.25.1.105 ZMQ_PLAIN_PASSWORD

```
#define ZMQ_PLAIN_PASSWORD 46
```

Definition at line 344 of file [zmq.h](#).

7.25.1.106 ZMQ_PLAIN_SERVER

```
#define ZMQ_PLAIN_SERVER 44
```

Definition at line 342 of file [zmq.h](#).

7.25.1.107 ZMQ_PLAIN_USERNAME

```
#define ZMQ_PLAIN_USERNAME 45
```

Definition at line 343 of file [zmq.h](#).

7.25.1.108 ZMQ_POLLERR

```
#define ZMQ_POLLERR 4
```

Definition at line 517 of file [zmq.h](#).

7.25.1.109 ZMQ_POLLIN

```
#define ZMQ_POLLIN 1
```

Definition at line 515 of file [zmq.h](#).

7.25.1.110 ZMQ_POLLITEMS_DFLT

```
#define ZMQ_POLLITEMS_DFLT 16
```

Definition at line 528 of file [zmq.h](#).

7.25.1.111 ZMQ_POLLOUT

```
#define ZMQ_POLLOUT 2
```

Definition at line 516 of file [zmq.h](#).

7.25.1.112 ZMQ_POLLPRI

```
#define ZMQ_POLLPRI 8
```

Definition at line 518 of file [zmq.h](#).

7.25.1.113 ZMQ_PROBE_ROUTER

```
#define ZMQ_PROBE_ROUTER 51
```

Definition at line 349 of file [zmq.h](#).

7.25.1.114 ZMQ_PROTOCOL_ERROR_WS_UNSPECIFIED

#define ZMQ_PROTOCOL_ERROR_WS_UNSPECIFIED 0x30000000
Definition at line 478 of file [zmq.h](#).

7.25.1.115 ZMQ_PROTOCOL_ERROR_ZAP_BAD_REQUEST_ID

#define ZMQ_PROTOCOL_ERROR_ZAP_BAD_REQUEST_ID 0x20000002
Definition at line 474 of file [zmq.h](#).

7.25.1.116 ZMQ_PROTOCOL_ERROR_ZAP_BAD_VERSION

#define ZMQ_PROTOCOL_ERROR_ZAP_BAD_VERSION 0x20000003
Definition at line 475 of file [zmq.h](#).

7.25.1.117 ZMQ_PROTOCOL_ERROR_ZAP_INVALID_METADATA

#define ZMQ_PROTOCOL_ERROR_ZAP_INVALID_METADATA 0x20000005
Definition at line 477 of file [zmq.h](#).

7.25.1.118 ZMQ_PROTOCOL_ERROR_ZAP_INVALID_STATUS_CODE

#define ZMQ_PROTOCOL_ERROR_ZAP_INVALID_STATUS_CODE 0x20000004
Definition at line 476 of file [zmq.h](#).

7.25.1.119 ZMQ_PROTOCOL_ERROR_ZAP_MALFORMED_REPLY

#define ZMQ_PROTOCOL_ERROR_ZAP_MALFORMED_REPLY 0x20000001
Definition at line 473 of file [zmq.h](#).

7.25.1.120 ZMQ_PROTOCOL_ERROR_ZAP_UNSPECIFIED

#define ZMQ_PROTOCOL_ERROR_ZAP_UNSPECIFIED 0x20000000
Definition at line 472 of file [zmq.h](#).

7.25.1.121 ZMQ_PROTOCOL_ERROR_ZMTP_CRYPTOGRAPHIC

#define ZMQ_PROTOCOL_ERROR_ZMTP_CRYPTOGRAPHIC 0x11000001
Definition at line 470 of file [zmq.h](#).

7.25.1.122 ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_METADATA

#define ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_METADATA 0x10000018
Definition at line 468 of file [zmq.h](#).

7.25.1.123 ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_SEQUENCE

#define ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_SEQUENCE 0x10000002
Definition at line 459 of file [zmq.h](#).

7.25.1.124 ZMQ_PROTOCOL_ERROR_ZMTP_KEY_EXCHANGE

#define ZMQ_PROTOCOL_ERROR_ZMTP_KEY_EXCHANGE 0x10000003
Definition at line 460 of file [zmq.h](#).

7.25.1.125 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_ERROR

#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_ERROR 0x10000015
Definition at line 465 of file [zmq.h](#).

7.25.1.126 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_HELLO

```
#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_HELLO 0x10000013
```

Definition at line 463 of file [zmq.h](#).

7.25.1.127 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_INITIATE

```
#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_INITIATE 0x10000014
```

Definition at line 464 of file [zmq.h](#).

7.25.1.128 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_MESSAGE

```
#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_MESSAGE 0x10000012
```

Definition at line 462 of file [zmq.h](#).

7.25.1.129 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_READY

```
#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_READY 0x10000016
```

Definition at line 466 of file [zmq.h](#).

7.25.1.130 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_UNSPECIFIED

```
#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_UNSPECIFIED 0x10000011
```

Definition at line 461 of file [zmq.h](#).

7.25.1.131 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_WELCOME

```
#define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_WELCOME 0x10000017
```

Definition at line 467 of file [zmq.h](#).

7.25.1.132 ZMQ_PROTOCOL_ERROR_ZMTP_MECHANISM_MISMATCH

```
#define ZMQ_PROTOCOL_ERROR_ZMTP_MECHANISM_MISMATCH 0x11000002
```

Definition at line 471 of file [zmq.h](#).

7.25.1.133 ZMQ_PROTOCOL_ERROR_ZMTP_UNEXPECTED_COMMAND

```
#define ZMQ_PROTOCOL_ERROR_ZMTP_UNEXPECTED_COMMAND 0x10000001
```

Definition at line 458 of file [zmq.h](#).

7.25.1.134 ZMQ_PROTOCOL_ERROR_ZMTP_UNSPECIFIED

```
#define ZMQ_PROTOCOL_ERROR_ZMTP_UNSPECIFIED 0x10000000
```

Definition at line 457 of file [zmq.h](#).

7.25.1.135 ZMQ_PUB

```
#define ZMQ_PUB 1
```

Definition at line 292 of file [zmq.h](#).

7.25.1.136 ZMQ_PULL

```
#define ZMQ_PULL 7
```

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

7.25.1.137 ZMQ_PUSH

```
#define ZMQ_PUSH 8
```

Definition at line 299 of file [zmq.h](#).

7.25.1.138 ZMQ_QUEUE

```
#define ZMQ_QUEUE 3
```

Definition at line 552 of file [zmq.h](#).

7.25.1.139 ZMQ_RATE

```
#define ZMQ_RATE 8
```

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

7.25.1.140 ZMQ_RCVBUF

```
#define ZMQ_RCVBUF 12
```

Definition at line 316 of file [zmq.h](#).

7.25.1.141 ZMQ_RCVHWM

```
#define ZMQ_RCVHWM 24
```

Definition at line 327 of file [zmq.h](#).

7.25.1.142 ZMQ_RCVMORE

```
#define ZMQ_RCVMORE 13
```

Definition at line 317 of file [zmq.h](#).

7.25.1.143 ZMQ_RCVTIMEO

```
#define ZMQ_RCVTIMEO 27
```

Definition at line 329 of file [zmq.h](#).

7.25.1.144 ZMQ_RECONNECT_IVL

```
#define ZMQ_RECONNECT_IVL 18
```

Definition at line 322 of file [zmq.h](#).

7.25.1.145 ZMQ_RECONNECT_IVL_MAX

```
#define ZMQ_RECONNECT_IVL_MAX 21
```

Definition at line 324 of file [zmq.h](#).

7.25.1.146 ZMQ_RECOVERY_IVL

```
#define ZMQ_RECOVERY_IVL 9
```

Definition at line 314 of file [zmq.h](#).

7.25.1.147 ZMQ_REP

```
#define ZMQ_REP 4
```

Definition at line 295 of file [zmq.h](#).

7.25.1.148 ZMQ_REQ

```
#define ZMQ_REQ 3
```

Definition at line 294 of file [zmq.h](#).

7.25.1.149 ZMQ_REQ_CORRELATE

```
#define ZMQ_REQ_CORRELATE 52
```

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

7.25.1.150 ZMQ_REQ_RELAXED

```
#define ZMQ_REQ_RELAXED 53
```

Definition at line 351 of file [zmq.h](#).

7.25.1.151 ZMQ_ROUTER

```
#define ZMQ_ROUTER 6
```

Definition at line 297 of file [zmq.h](#).

7.25.1.152 ZMQ_ROUTER_BEHAVIOR

```
#define ZMQ_ROUTER_BEHAVIOR ZMQ_ROUTER_MANDATORY
```

Definition at line 414 of file [zmq.h](#).

7.25.1.153 ZMQ_ROUTER_HANOVER

```
#define ZMQ_ROUTER_HANOVER 56
```

Definition at line 354 of file [zmq.h](#).

7.25.1.154 ZMQ_ROUTER_MANDATORY

```
#define ZMQ_ROUTER_MANDATORY 33
```

Definition at line 332 of file [zmq.h](#).

7.25.1.155 ZMQ_ROUTER_RAW

```
#define ZMQ_ROUTER_RAW 41
```

Definition at line 339 of file [zmq.h](#).

7.25.1.156 ZMQ_ROUTING_ID

```
#define ZMQ_ROUTING_ID 5
```

Definition at line 310 of file [zmq.h](#).

7.25.1.157 ZMQ_SHARED

```
#define ZMQ_SHARED 3
```

Definition at line 388 of file [zmq.h](#).

7.25.1.158 ZMQ_SNDBUF

```
#define ZMQ_SNDBUF 11
```

Definition at line 315 of file [zmq.h](#).

7.25.1.159 ZMQ_SNDHWM

```
#define ZMQ_SNDHWM 23
```

Definition at line 326 of file [zmq.h](#).

7.25.1.160 ZMQ_SNDMORE

```
#define ZMQ_SNDMORE 2
```

Definition at line 392 of file [zmq.h](#).

7.25.1.161 ZMQ_SNDTIMEO

```
#define ZMQ_SNDTIMEO 28
```

Definition at line 330 of file [zmq.h](#).

7.25.1.162 ZMQ_SOCKET_LIMIT

```
#define ZMQ_SOCKET_LIMIT 3
```

Definition at line 216 of file [zmq.h](#).

7.25.1.163 ZMQ SOCKS_PROXY

```
#define ZMQ SOCKS_PROXY 68
```

Definition at line 362 of file [zmq.h](#).

7.25.1.164 ZMQ_SRCFD

```
#define ZMQ_SRCFD 2
```

Definition at line 417 of file [zmq.h](#).

7.25.1.165 ZMQ_STREAM

```
#define ZMQ_STREAM 11
```

Definition at line 302 of file [zmq.h](#).

7.25.1.166 ZMQ_STREAM_NOTIFY

```
#define ZMQ_STREAM_NOTIFY 73
```

Definition at line 367 of file [zmq.h](#).

7.25.1.167 ZMQ_STREAMER

```
#define ZMQ_STREAMER 1
```

Definition at line 550 of file [zmq.h](#).

7.25.1.168 ZMQ_SUB

```
#define ZMQ_SUB 2
```

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

7.25.1.169 ZMQ_SUBSCRIBE

```
#define ZMQ_SUBSCRIBE 6
```

Definition at line 311 of file [zmq.h](#).

7.25.1.170 ZMQ_TCP_ACCEPT_FILTER

```
#define ZMQ_TCP_ACCEPT_FILTER 38
```

Definition at line 406 of file [zmq.h](#).

7.25.1.171 ZMQ_TCP_KEEPA_LIVE

```
#define ZMQ_TCP_KEEPA_LIVE 34
```

Definition at line 333 of file [zmq.h](#).

7.25.1.172 ZMQ_TCP_KEEPA_LIVE_CNT

```
#define ZMQ_TCP_KEEPA_LIVE_CNT 35
```

Definition at line 334 of file [zmq.h](#).

7.25.1.173 ZMQ_TCP_KEEPA_LIVE_IDLE

```
#define ZMQ_TCP_KEEPA_LIVE_IDLE 36
```

Definition at line 335 of file [zmq.h](#).

7.25.1.174 ZMQ_TCP_KEEPALIVE_INTVL

```
#define ZMQ_TCP_KEEPALIVE_INTVL 37
```

Definition at line 336 of file [zmq.h](#).

7.25.1.175 ZMQ_TCP_MAXRT

```
#define ZMQ_TCP_MAXRT 80
```

Definition at line 374 of file [zmq.h](#).

7.25.1.176 ZMQ_THREAD_AFFINITY_CPU_ADD

```
#define ZMQ_THREAD_AFFINITY_CPU_ADD 7
```

Definition at line 221 of file [zmq.h](#).

7.25.1.177 ZMQ_THREAD_AFFINITY_CPU_REMOVE

```
#define ZMQ_THREAD_AFFINITY_CPU_REMOVE 8
```

Definition at line 222 of file [zmq.h](#).

7.25.1.178 ZMQ_THREAD_NAME_PREFIX

```
#define ZMQ_THREAD_NAME_PREFIX 9
```

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

7.25.1.179 ZMQ_THREAD_PRIORITY

```
#define ZMQ_THREAD_PRIORITY 3
```

Definition at line 217 of file [zmq.h](#).

7.25.1.180 ZMQ_THREAD_PRIORITY_DFLT

```
#define ZMQ_THREAD_PRIORITY_DFLT -1
```

Definition at line 228 of file [zmq.h](#).

7.25.1.181 ZMQ_THREAD_SAFE

```
#define ZMQ_THREAD_SAFE 81
```

Definition at line 375 of file [zmq.h](#).

7.25.1.182 ZMQ_THREAD_SCHED_POLICY

```
#define ZMQ_THREAD_SCHED_POLICY 4
```

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

7.25.1.183 ZMQ_THREAD_SCHED_POLICY_DFLT

```
#define ZMQ_THREAD_SCHED_POLICY_DFLT -1
```

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

7.25.1.184 ZMQ_TOS

```
#define ZMQ_TOS 57
```

Definition at line 355 of file [zmq.h](#).

7.25.1.185 ZMQ_TYPE

```
#define ZMQ_TYPE 16
```

Definition at line 320 of file [zmq.h](#).

7.25.1.186 ZMQ_UNSUBSCRIBE

```
#define ZMQ_UNSUBSCRIBE 7
```

Definition at line 312 of file [zmq.h](#).

7.25.1.187 ZMQ_USE_FD

```
#define ZMQ_USE_FD 89
```

Definition at line 381 of file [zmq.h](#).

7.25.1.188 ZMQ_VERSION

```
#define ZMQ_VERSION ZMQ_MAKE_VERSION (ZMQ_VERSION_MAJOR, ZMQ_VERSION_MINOR, ZMQ_VERSION_PATCH)
```

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

7.25.1.189 ZMQ_VERSION_MAJOR

```
#define ZMQ_VERSION_MAJOR 4
```

Definition at line 42 of file [zmq.h](#).

7.25.1.190 ZMQ_VERSION_MINOR

```
#define ZMQ_VERSION_MINOR 3
```

Definition at line 43 of file [zmq.h](#).

7.25.1.191 ZMQ_VERSION_PATCH

```
#define ZMQ_VERSION_PATCH 4
```

Definition at line 44 of file [zmq.h](#).

7.25.1.192 ZMQ_VMCI_BUFFER_MAX_SIZE

```
#define ZMQ_VMCI_BUFFER_MAX_SIZE 87
```

Definition at line 379 of file [zmq.h](#).

7.25.1.193 ZMQ_VMCI_BUFFER_MIN_SIZE

```
#define ZMQ_VMCI_BUFFER_MIN_SIZE 86
```

Definition at line 378 of file [zmq.h](#).

7.25.1.194 ZMQ_VMCI_BUFFER_SIZE

```
#define ZMQ_VMCI_BUFFER_SIZE 85
```

Definition at line 377 of file [zmq.h](#).

7.25.1.195 ZMQ_VMCI_CONNECT_TIMEOUT

```
#define ZMQ_VMCI_CONNECT_TIMEOUT 88
```

Definition at line 380 of file [zmq.h](#).

7.25.1.196 ZMQ_XPUB

```
#define ZMQ_XPUB 9
```

Definition at line 300 of file [zmq.h](#).

7.25.1.197 ZMQ_XPUB_MANUAL

```
#define ZMQ_XPUB_MANUAL 71
```

Definition at line 365 of file [zmq.h](#).

7.25.1.198 ZMQ_XPUB_NODROP

```
#define ZMQ_XPUB_NODROP 69
```

Definition at line 363 of file [zmq.h](#).

7.25.1.199 ZMQ_XPUB_VERBOSE

```
#define ZMQ_XPUB_VERBOSE 40
```

Definition at line 338 of file [zmq.h](#).

7.25.1.200 ZMQ_XPUB_VERBOSESER

```
#define ZMQ_XPUB_VERBOSESER 78
```

Definition at line 372 of file [zmq.h](#).

7.25.1.201 ZMQ_XPUB_WELCOME_MSG

```
#define ZMQ_XPUB_WELCOME_MSG 72
```

Definition at line 366 of file [zmq.h](#).

7.25.1.202 ZMQ_XREP

```
#define ZMQ_XREP ZMQ_ROUTER
```

Definition at line 306 of file [zmq.h](#).

7.25.1.203 ZMQ_XREQ

```
#define ZMQ_XREQ ZMQ DEALER
```

Definition at line 305 of file [zmq.h](#).

7.25.1.204 ZMQ_XSUB

```
#define ZMQ_XSUB 10
```

Definition at line 301 of file [zmq.h](#).

7.25.1.205 ZMQ_ZAP_DOMAIN

```
#define ZMQ_ZAP_DOMAIN 55
```

Definition at line 353 of file [zmq.h](#).

7.25.2 Typedef Documentation

7.25.2.1 zmq_fd_t

```
typedef int zmq_fd_t
```

Definition at line 508 of file [zmq.h](#).

7.25.2.2 zmq_free_fn

```
typedef void() zmq_free_fn(void *data_, void *hint_)
```

Definition at line 267 of file [zmq.h](#).

7.25.2.3 zmq_msg_t

```
typedef struct zmq_msg_t zmq_msg_t
```

7.25.2.4 zmq_pollitem_t

```
typedef struct zmq_pollitem_t zmq_pollitem_t
```

7.25.2.5 zmq_thread_fn

typedef void() zmq_thread_fn(void *)
Definition at line 638 of file [zmq.h](#).

7.25.2.6 zmq_timer_fn

typedef void() zmq_timer_fn(int timer_id, void *arg)
Definition at line 601 of file [zmq.h](#).

7.25.3 Function Documentation

7.25.3.1 zmq_atomic_counter_dec()

```
ZMQ_EXPORT int zmq_atomic_counter_dec (  
    void * counter_ )
```

7.25.3.2 zmq_atomic_counter_destroy()

```
ZMQ_EXPORT void zmq_atomic_counter_destroy (  
    void ** counter_p_ )
```

7.25.3.3 zmq_atomic_counter_inc()

```
ZMQ_EXPORT int zmq_atomic_counter_inc (  
    void * counter_ )
```

7.25.3.4 zmq_atomic_counter_new()

```
ZMQ_EXPORT void * zmq_atomic_counter_new (  
    void )
```

7.25.3.5 zmq_atomic_counter_set()

```
ZMQ_EXPORT void zmq_atomic_counter_set (  
    void * counter_,  
    int value_ )
```

7.25.3.6 zmq_atomic_counter_value()

```
ZMQ_EXPORT int zmq_atomic_counter_value (  
    void * counter_ )
```

7.25.3.7 zmq_bind()

```
ZMQ_EXPORT int zmq_bind (  
    void * s_,  
    const char * addr_ )
```

7.25.3.8 zmq_close()

```
ZMQ_EXPORT int zmq_close (  
    void * s_ )
```

7.25.3.9 zmq_connect()

```
ZMQ_EXPORT int zmq_connect (  
    void * s_,  
    const char * addr_ )
```

7.25.3.10 zmq_ctx_destroy()

```
ZMQ_EXPORT int zmq_ctx_destroy (  
    void * context_ )
```

7.25.3.11 zmq_ctx_get()

```
ZMQ_EXPORT int zmq_ctx_get (  
    void * context_,  
    int option_ )
```

7.25.3.12 zmq_ctx_new()

```
ZMQ_EXPORT void * zmq_ctx_new (  
    void )
```

7.25.3.13 zmq_ctx_set()

```
ZMQ_EXPORT int zmq_ctx_set (  
    void * context_,  
    int option_,  
    int optval_ )
```

7.25.3.14 zmq_ctx_shutdown()

```
ZMQ_EXPORT int zmq_ctx_shutdown (  
    void * context_ )
```

7.25.3.15 zmq_ctx_term()

```
ZMQ_EXPORT int zmq_ctx_term (  
    void * context_ )
```

7.25.3.16 zmq_curve_keypair()

```
ZMQ_EXPORT int zmq_curve_keypair (  
    char * z85_public_key_,  
    char * z85_secret_key_ )
```

7.25.3.17 zmq_curve_public()

```
ZMQ_EXPORT int zmq_curve_public (  
    char * z85_public_key_,  
    const char * z85_secret_key_ )
```

7.25.3.18 zmq_device()

```
ZMQ_EXPORT int zmq_device (  
    int type_,  
    void * frontend_,  
    void * backend_ )
```

7.25.3.19 zmq_disconnect()

```
ZMQ_EXPORT int zmq_disconnect (  
    void * s_,  
    const char * addr_ )
```

7.25.3.20 zmq_errno()

```
ZMQ_EXPORT int zmq_errno (
    void )
```

7.25.3.21 zmq_getsockopt()

```
ZMQ_EXPORT int zmq_getsockopt (
    void * s_,
    int option_,
    void * optval_,
    size_t * optvallen_ )
```

7.25.3.22 zmq_has()

```
ZMQ_EXPORT int zmq_has (
    const char * capability_ )
```

7.25.3.23 zmq_init()

```
ZMQ_EXPORT void * zmq_init (
    int io_threads_ )
```

7.25.3.24 zmq_msg_close()

```
ZMQ_EXPORT int zmq_msg_close (
    zmq_msg_t * msg_ )
```

7.25.3.25 zmq_msg_copy()

```
ZMQ_EXPORT int zmq_msg_copy (
    zmq_msg_t * dest_,
    zmq_msg_t * src_ )
```

7.25.3.26 zmq_msg_data()

```
ZMQ_EXPORT void * zmq_msg_data (
    zmq_msg_t * msg_ )
```

7.25.3.27 zmq_msg_get()

```
ZMQ_EXPORT int zmq_msg_get (
    const zmq_msg_t * msg_,
    int property_ )
```

7.25.3.28 zmq_msg_gets()

```
ZMQ_EXPORT const char * zmq_msg_gets (
    const zmq_msg_t * msg_,
    const char * property_ )
```

7.25.3.29 zmq_msg_init()

```
ZMQ_EXPORT int zmq_msg_init (
    zmq_msg_t * msg_ )
```

7.25.3.30 zmq_msg_init_data()

```
ZMQ_EXPORT int zmq_msg_init_data (
    zmq_msg_t * msg_,
    void * data_,
    size_t size_,
    zmq_free_fn * ffn_,
    void * hint_ )
```

7.25.3.31 zmq_msg_init_size()

```
ZMQ_EXPORT int zmq_msg_init_size (
    zmq_msg_t * msg_,
    size_t size_ )
```

7.25.3.32 zmq_msg_more()

```
ZMQ_EXPORT int zmq_msg_more (
    const zmq_msg_t * msg_ )
```

7.25.3.33 zmq_msg_move()

```
ZMQ_EXPORT int zmq_msg_move (
    zmq_msg_t * dest_,
    zmq_msg_t * src_ )
```

7.25.3.34 zmq_msg_recv()

```
ZMQ_EXPORT int zmq_msg_recv (
    zmq_msg_t * msg_,
    void * s_,
    int flags_ )
```

7.25.3.35 zmq_msg_send()

```
ZMQ_EXPORT int zmq_msg_send (
    zmq_msg_t * msg_,
    void * s_,
    int flags_ )
```

7.25.3.36 zmq_msg_set()

```
ZMQ_EXPORT int zmq_msg_set (
    zmq_msg_t * msg_,
    int property_,
    int optval_ )
```

7.25.3.37 zmq_msg_size()

```
ZMQ_EXPORT size_t zmq_msg_size (
    const zmq_msg_t * msg_ )
```

7.25.3.38 zmq_poll()

```
ZMQ_EXPORT int zmq_poll (
    zmq_pollitem_t * items_,
    int nitems_,
    long timeout_ )
```

7.25.3.39 zmq_proxy()

```
ZMQ_EXPORT int zmq_proxy (
    void * frontend_,
    void * backend_,
    void * capture_ )
```

7.25.3.40 zmq_proxy_steerable()

```
ZMQ_EXPORT int zmq_proxy_steerable (
    void * frontend_,
    void * backend_,
    void * capture_,
    void * control_ )
```

7.25.3.41 zmq_recv()

```
ZMQ_EXPORT int zmq_recv (
    void * s_,
    void * buf_,
    size_t len_,
    int flags_ )
```

7.25.3.42 zmq_recviov()

```
ZMQ_EXPORT int zmq_recviov (
    void * s_,
    struct iovec * iov_,
    size_t * count_,
    int flags_ )
```

7.25.3.43 zmq_recvmsg()

```
ZMQ_EXPORT int zmq_recvmsg (
    void * s_,
    zmq_msg_t * msg_,
    int flags_ )
```

7.25.3.44 zmq_send()

```
ZMQ_EXPORT int zmq_send (
    void * s_,
    const void * buf_,
    size_t len_,
    int flags_ )
```

7.25.3.45 zmq_send_const()

```
ZMQ_EXPORT int zmq_send_const (
    void * s_,
    const void * buf_,
    size_t len_,
    int flags_ )
```

7.25.3.46 zmq_sendiov()

```
ZMQ_EXPORT int zmq_sendiov (
    void * s_,
    struct iovec * iov_,
```

```
size_t count_,  
int flags_ )
```

7.25.3.47 zmq_sendmsg()

```
ZMQ_EXPORT int zmq_sendmsg (  
    void * s_,  
    zmq_msg_t * msg_,  
    int flags_ )
```

7.25.3.48 zmq_setsockopt()

```
ZMQ_EXPORT int zmq_setsockopt (  
    void * s_,  
    int option_,  
    const void * optval_,  
    size_t optvallen_ )
```

7.25.3.49 zmq_sleep()

```
ZMQ_EXPORT void zmq_sleep (  
    int seconds_ )
```

7.25.3.50 zmq_socket()

```
ZMQ_EXPORT void * zmq_socket (  
    void * ,  
    int type_ )
```

7.25.3.51 zmq_socket_monitor()

```
ZMQ_EXPORT int zmq_socket_monitor (  
    void * s_,  
    const char * addr_,  
    int events_ )
```

7.25.3.52 zmq_stopwatch_intermediate()

```
ZMQ_EXPORT unsigned long zmq_stopwatch_intermediate (  
    void * watch_ )
```

7.25.3.53 zmq_stopwatch_start()

```
ZMQ_EXPORT void * zmq_stopwatch_start (  
    void )
```

7.25.3.54 zmq_stopwatch_stop()

```
ZMQ_EXPORT unsigned long zmq_stopwatch_stop (  
    void * watch_ )
```

7.25.3.55 zmq_strerror()

```
ZMQ_EXPORT const char * zmq_strerror (  
    int errnum_ )
```

7.25.3.56 zmq_term()

```
ZMQ_EXPORT int zmq_term (  
    void * context_ )
```

7.25.3.57 zmq_threadclose()

```
ZMQ_EXPORT void zmq_threadclose (
    void * thread_ )
```

7.25.3.58 zmq_threadstart()

```
ZMQ_EXPORT void * zmq_threadstart (
    zmq_thread_fn * func_,
    void * arg_ )
```

7.25.3.59 zmq_timers_add()

```
ZMQ_EXPORT int zmq_timers_add (
    void * timers,
    size_t interval,
    zmq_timer_fn handler,
    void * arg )
```

7.25.3.60 zmq_timers_cancel()

```
ZMQ_EXPORT int zmq_timers_cancel (
    void * timers,
    int timer_id )
```

7.25.3.61 zmq_timers_destroy()

```
ZMQ_EXPORT int zmq_timers_destroy (
    void ** timers_p )
```

7.25.3.62 zmq_timers_execute()

```
ZMQ_EXPORT int zmq_timers_execute (
    void * timers )
```

7.25.3.63 zmq_timers_new()

```
ZMQ_EXPORT void * zmq_timers_new (
    void )
```

7.25.3.64 zmq_timers_reset()

```
ZMQ_EXPORT int zmq_timers_reset (
    void * timers,
    int timer_id )
```

7.25.3.65 zmq_timers_set_interval()

```
ZMQ_EXPORT int zmq_timers_set_interval (
    void * timers,
    int timer_id,
    size_t interval )
```

7.25.3.66 zmq_timers_timeout()

```
ZMQ_EXPORT long zmq_timers_timeout (
    void * timers )
```


7.25.3.67 zmq_unbind()

```
ZMQ_EXPORT int zmq_unbind (
    void * s_,
    const char * addr_ )
```

7.25.3.68 zmq_version()

```
ZMQ_EXPORT void zmq_version (
    int * major_,
    int * minor_,
    int * patch_ )
```

7.25.3.69 zmq_z85_decode()

```
ZMQ_EXPORT uint8_t * zmq_z85_decode (
    uint8_t * dest_,
    const char * string_ )
```

7.25.3.70 zmq_z85_encode()

```
ZMQ_EXPORT char * zmq_z85_encode (
    char * dest_,
    const uint8_t * data_,
    size_t size_ )
```

7.26 zmq.h

[Go to the documentation of this file.](#)

```
00001 /*
00002     Copyright (c) 2007-2016 Contributors as noted in the AUTHORS file
00003
00004     This file is part of libzmq, the ZeroMQ core engine in C++.
00005
00006     libzmq is free software; you can redistribute it and/or modify it under
00007     the terms of the GNU Lesser General Public License (LGPL) as published
00008     by the Free Software Foundation; either version 3 of the License, or
00009     (at your option) any later version.
00010
00011     As a special exception, the Contributors give you permission to link
00012     this library with independent modules to produce an executable,
00013     regardless of the license terms of these independent modules, and to
00014     copy and distribute the resulting executable under terms of your choice,
00015     provided that you also meet, for each linked independent module, the
00016     terms and conditions of the license of that module. An independent
00017     module is a module which is not derived from or based on this library.
00018     If you modify this library, you must extend this exception to your
00019     version of the library.
00020
00021     libzmq is distributed in the hope that it will be useful, but WITHOUT
00022     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
00023     FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
00024     License for more details.
00025
00026     You should have received a copy of the GNU Lesser General Public License
00027     along with this program. If not, see <http://www.gnu.org/licenses/>.
00028
00029     *****
00030     NOTE to contributors. This file comprises the principal public contract
00031     for ZeroMQ API users. Any change to this file supplied in a stable
00032     release SHOULD not break existing applications.
00033     In practice this means that the value of constants must not change, and
00034     that old values may not be reused for new constants.
00035     *****
00036 */
00037
00038 #ifndef __ZMQ_H_INCLUDED__
00039 #define __ZMQ_H_INCLUDED__
00040
00041 /* Version macros for compile-time API version detection */
00042 #define ZMQ_VERSION_MAJOR 4
00043 #define ZMQ_VERSION_MINOR 3
00044 #define ZMQ_VERSION_PATCH 4
```

```

00045
00046 #define ZMQ_MAKE_VERSION(major, minor, patch)          \
00047     ((major) *10000 + (minor) *100 + (patch))          \
00048 #define ZMQ_VERSION                                     \
00049     ZMQ_MAKE_VERSION (ZMQ_VERSION_MAJOR, ZMQ_VERSION_MINOR, ZMQ_VERSION_PATCH)
00050
00051 #ifdef __cplusplus
00052 extern "C" {
00053 #endif
00054
00055 #if !defined _WIN32_WCE
00056 #include <errno.h>
00057 #endif
00058 #include <stddef.h>
00059 #include <stdio.h>
00060 #if defined _WIN32
00061 // Set target version to Windows Server 2008, Windows Vista or higher.
00062 // Windows XP (0x0501) is supported but without client & server socket types.
00063 #ifndef _WIN32_WINNT
00064 #define _WIN32_WINNT 0x0600
00065 #endif
00066
00067 #ifdef __MINGW32__
00068 // Require Windows XP or higher with MinGW for getaddrinfo().
00069 #if (_WIN32_WINNT >= 0x0501)
00070 #else
00071 #error You need at least Windows XP target
00072 #endif
00073 #endif
00074 #endif
00075
00076 /* Handle DSO symbol visibility */
00077 #if defined _WIN32
00078 #if defined ZMQ_STATIC
00079 #define ZMQ_EXPORT
00080 #elif defined DLL_EXPORT
00081 #define ZMQ_EXPORT __declspec(dllexport)
00082 #else
00083 #define ZMQ_EXPORT __declspec(dllimport)
00084 #endif
00085 #else
00086 #if defined __SUNPRO_C || defined __SUNPRO_CC
00087 #define ZMQ_EXPORT __global
00088 #elif (defined __GNUC__ && __GNUC__ >= 4) || defined __INTEL_COMPILER
00089 #define ZMQ_EXPORT __attribute__((visibility ("default")))
00090 #else
00091 #define ZMQ_EXPORT
00092 #endif
00093 #endif
00094
00095 /* Define integer types needed for event interface */
00096 #define ZMQ_DEFINED_STDINT 1
00097 #if defined ZMQ_HAVE_SOLARIS || defined ZMQ_HAVE_OPENVMS
00098 #include <inttypes.h>
00099 #elif defined _MSC_VER && _MSC_VER < 1600
00100 #ifndef uint64_t
00101 typedef unsigned __int64 uint64_t;
00102 #endif
00103 #ifndef int32_t
00104 typedef __int32 int32_t;
00105 #endif
00106 #ifndef uint32_t
00107 typedef unsigned __int32 uint32_t;
00108 #endif
00109 #ifndef uint16_t
00110 typedef unsigned __int16 uint16_t;
00111 #endif
00112 #ifndef uint8_t
00113 typedef unsigned __int8 uint8_t;
00114 #endif
00115 #else
00116 #include <stdint.h>
00117 #endif
00118
00119 // 32-bit AIX's pollfd struct members are called regevents and rtnevents so it
00120 // defines compatibility macros for them. Need to include that header first to
00121 // stop build failures since zmq_pollset_t defines them as events and revents.
00122 #ifdef ZMQ_HAVE_AIX
00123 #include <poll.h>
00124 #endif
00125
00126
00127 /*****
00128 /* OMQ errors. */
00129 /*****
00130
00131 /* A number random enough not to collide with different errno ranges on */

```

```

00132 /* different OSes. The assumption is that error_t is at least 32-bit type. */
00133 #define ZMQ_HAUSNUMERO 156384712
00134
00135 /* On Windows platform some of the standard POSIX errno are not defined. */
00136 #ifndef ENOTSUP
00137 #define ENOTSUP (ZMQ_HAUSNUMERO + 1)
00138 #endif
00139 #ifndef EPROTONOSUPPORT
00140 #define EPROTONOSUPPORT (ZMQ_HAUSNUMERO + 2)
00141 #endif
00142 #ifndef ENOBUFS
00143 #define ENOBUFS (ZMQ_HAUSNUMERO + 3)
00144 #endif
00145 #ifndef ENETDOWN
00146 #define ENETDOWN (ZMQ_HAUSNUMERO + 4)
00147 #endif
00148 #ifndef EADDRINUSE
00149 #define EADDRINUSE (ZMQ_HAUSNUMERO + 5)
00150 #endif
00151 #ifndef EADDRNOTAVAIL
00152 #define EADDRNOTAVAIL (ZMQ_HAUSNUMERO + 6)
00153 #endif
00154 #ifndef ECONNREFUSED
00155 #define ECONNREFUSED (ZMQ_HAUSNUMERO + 7)
00156 #endif
00157 #ifndef EINPROGRESS
00158 #define EINPROGRESS (ZMQ_HAUSNUMERO + 8)
00159 #endif
00160 #ifndef ENOTSOCK
00161 #define ENOTSOCK (ZMQ_HAUSNUMERO + 9)
00162 #endif
00163 #ifndef EMSGSIZE
00164 #define EMSGSIZE (ZMQ_HAUSNUMERO + 10)
00165 #endif
00166 #ifndef EAFNOSUPPORT
00167 #define EAFNOSUPPORT (ZMQ_HAUSNUMERO + 11)
00168 #endif
00169 #ifndef ENETUNREACH
00170 #define ENETUNREACH (ZMQ_HAUSNUMERO + 12)
00171 #endif
00172 #ifndef ECONNABORTED
00173 #define ECONNABORTED (ZMQ_HAUSNUMERO + 13)
00174 #endif
00175 #ifndef ECONNRESET
00176 #define ECONNRESET (ZMQ_HAUSNUMERO + 14)
00177 #endif
00178 #ifndef ENOTCONN
00179 #define ENOTCONN (ZMQ_HAUSNUMERO + 15)
00180 #endif
00181 #ifndef ETIMEDOUT
00182 #define ETIMEDOUT (ZMQ_HAUSNUMERO + 16)
00183 #endif
00184 #ifndef EHOSTUNREACH
00185 #define EHOSTUNREACH (ZMQ_HAUSNUMERO + 17)
00186 #endif
00187 #ifndef ENETRESET
00188 #define ENETRESET (ZMQ_HAUSNUMERO + 18)
00189 #endif
00190
00191 /* Native OMQ error codes. */
00192 #define EFSM (ZMQ_HAUSNUMERO + 51)
00193 #define ENOCOMPATPROTO (ZMQ_HAUSNUMERO + 52)
00194 #define ETERM (ZMQ_HAUSNUMERO + 53)
00195 #define EMTHREAD (ZMQ_HAUSNUMERO + 54)
00196
00197 /* This function retrieves the errno as it is known to OMQ library. The goal */
00198 /* of this function is to make the code 100% portable, including where OMQ */
00199 /* compiled with certain CRT library (on Windows) is linked to an */
00200 /* application that uses different CRT library. */
00201 ZMQ_EXPORT int zmq_errno (void);
00202
00203 /* Resolves system errors and OMQ errors to human-readable string. */
00204 ZMQ_EXPORT const char *zmq_strerror (int errnum_);
00205
00206 /* Run-time API version detection */
00207 ZMQ_EXPORT void zmq_version (int *major_, int *minor_, int *patch_);
00208
00209 /*****
00210 /* OMQ infrastructure (a.k.a. context) initialisation & termination. */
00211 *****/
00212
00213 /* Context options */
00214 #define ZMQ_IO_THREADS 1
00215 #define ZMQ_MAX_SOCKETS 2
00216 #define ZMQ_SOCKET_LIMIT 3
00217 #define ZMQ_THREAD_PRIORITY 3
00218 #define ZMQ_THREAD_SCHED_POLICY 4

```

```

00219 #define ZMQ_MAX_MSGSZ 5
00220 #define ZMQ_MSG_T_SIZE 6
00221 #define ZMQ_THREAD_AFFINITY_CPU_ADD 7
00222 #define ZMQ_THREAD_AFFINITY_CPU_REMOVE 8
00223 #define ZMQ_THREAD_NAME_PREFIX 9
00224
00225 /* Default for new contexts */
00226 #define ZMQ_IO_THREADS_DFLT 1
00227 #define ZMQ_MAX_SOCKETS_DFLT 1023
00228 #define ZMQ_THREAD_PRIORITY_DFLT -1
00229 #define ZMQ_THREAD_SCHED_POLICY_DFLT -1
00230
00231 ZMQ_EXPORT void *zmq_ctx_new (void);
00232 ZMQ_EXPORT int zmq_ctx_term (void *context_);
00233 ZMQ_EXPORT int zmq_ctx_shutdown (void *context_);
00234 ZMQ_EXPORT int zmq_ctx_set (void *context_, int option_, int optval_);
00235 ZMQ_EXPORT int zmq_ctx_get (void *context_, int option_);
00236
00237 /* Old (legacy) API */
00238 ZMQ_EXPORT void *zmq_init (int io_threads_);
00239 ZMQ_EXPORT int zmq_term (void *context_);
00240 ZMQ_EXPORT int zmq_ctx_destroy (void *context_);
00241
00242
00243 /*****
00244  * OMQ message definition.
00245  *****/
00246
00247 /* Some architectures, like sparc64 and some variants of aarch64, enforce pointer
00248  * alignment and raise sigbus on violations. Make sure applications allocate
00249  * zmq_msg_t on addresses aligned on a pointer-size boundary to avoid this issue.
00250  */
00251 typedef struct zmq_msg_t
00252 {
00253     #if defined(_MSC_VER) && (defined(_M_X64) || defined(_M_ARM64))
00254         __declspec(aligned (8)) unsigned char _[64];
00255     #elif defined(_MSC_VER)
00256         && (defined(_M_IX86) || defined(_M_ARM_ARMV7VE) || defined(_M_ARM))
00257         __declspec(aligned (4)) unsigned char _[64];
00258     #elif defined(__GNUC__) || defined(__INTEL_COMPILER)
00259         || (defined(__SUNPRO_C) && __SUNPRO_C >= 0x590)
00260         || (defined(__SUNPRO_CC) && __SUNPRO_CC >= 0x590)
00261         unsigned char _[64] __attribute__((aligned (sizeof (void *)))));
00262     #else
00263         unsigned char _[64];
00264     #endif
00265 } zmq_msg_t;
00266
00267 typedef void (zmq_free_fn) (void *data_, void *hint_);
00268
00269 ZMQ_EXPORT int zmq_msg_init (zmq_msg_t *msg_);
00270 ZMQ_EXPORT int zmq_msg_init_size (zmq_msg_t *msg_, size_t size_);
00271 ZMQ_EXPORT int zmq_msg_init_data (
00272     zmq_msg_t *msg_, void *data_, size_t size_, zmq_free_fn *ffn_, void *hint_);
00273 ZMQ_EXPORT int zmq_msg_send (zmq_msg_t *msg_, void *s_, int flags_);
00274 ZMQ_EXPORT int zmq_msg_recv (zmq_msg_t *msg_, void *s_, int flags_);
00275 ZMQ_EXPORT int zmq_msg_close (zmq_msg_t *msg_);
00276 ZMQ_EXPORT int zmq_msg_move (zmq_msg_t *dest_, zmq_msg_t *src_);
00277 ZMQ_EXPORT int zmq_msg_copy (zmq_msg_t *dest_, zmq_msg_t *src_);
00278 ZMQ_EXPORT void *zmq_msg_data (zmq_msg_t *msg_);
00279 ZMQ_EXPORT size_t zmq_msg_size (const zmq_msg_t *msg_);
00280 ZMQ_EXPORT int zmq_msg_more (const zmq_msg_t *msg_);
00281 ZMQ_EXPORT int zmq_msg_get (const zmq_msg_t *msg_, int property_);
00282 ZMQ_EXPORT int zmq_msg_set (zmq_msg_t *msg_, int property_, int optval_);
00283 ZMQ_EXPORT const char *zmq_msg_gets (const zmq_msg_t *msg_,
00284     const char *property_);
00285
00286 /*****
00287  * OMQ socket definition.
00288  *****/
00289
00290 /* Socket types. */
00291 #define ZMQ_PAIR 0
00292 #define ZMQ_PUB 1
00293 #define ZMQ_SUB 2
00294 #define ZMQ_REQ 3
00295 #define ZMQ_REP 4
00296 #define ZMQ_DEALER 5
00297 #define ZMQ_ROUTER 6
00298 #define ZMQ_PULL 7
00299 #define ZMQ_PUSH 8
00300 #define ZMQ_XPUB 9
00301 #define ZMQ_XSUB 10
00302 #define ZMQ_STREAM 11
00303
00304 /* Deprecated aliases */
00305 #define ZMQ_XREQ ZMQ_DEALER

```

```

00306 #define ZMQ_XREP ZMQ_ROUTER
00307
00308 /* Socket options. */
00309 #define ZMQ_AFFINITY 4
00310 #define ZMQ_ROUTING_ID 5
00311 #define ZMQ_SUBSCRIBE 6
00312 #define ZMQ_UNSUBSCRIBE 7
00313 #define ZMQ_RATE 8
00314 #define ZMQ_RECOVERY_IVL 9
00315 #define ZMQ_SNDBUF 11
00316 #define ZMQ_RCVBUF 12
00317 #define ZMQ_RCVMORE 13
00318 #define ZMQ_FD 14
00319 #define ZMQ_EVENTS 15
00320 #define ZMQ_TYPE 16
00321 #define ZMQ_LINGER 17
00322 #define ZMQ_RECONNECT_IVL 18
00323 #define ZMQ_BACKLOG 19
00324 #define ZMQ_RECONNECT_IVL_MAX 21
00325 #define ZMQ_MAXMSGSIZE 22
00326 #define ZMQ_SNDHWM 23
00327 #define ZMQ_RCVHWM 24
00328 #define ZMQ_MULTICAST_HOPS 25
00329 #define ZMQ_RCVTIMEO 27
00330 #define ZMQ_SNDTIMEO 28
00331 #define ZMQ_LAST_ENDPOINT 32
00332 #define ZMQ_ROUTER_MANDATORY 33
00333 #define ZMQ_TCP_KEEPA_LIVE 34
00334 #define ZMQ_TCP_KEEPA_LIVE_CNT 35
00335 #define ZMQ_TCP_KEEPA_LIVE_IDLE 36
00336 #define ZMQ_TCP_KEEPA_LIVE_INTVL 37
00337 #define ZMQ_IMMEDIATE 39
00338 #define ZMQ_XPUB_VERBOSE 40
00339 #define ZMQ_ROUTER_RAW 41
00340 #define ZMQ_IPV6 42
00341 #define ZMQ_MECHANISM 43
00342 #define ZMQ_PLAIN_SERVER 44
00343 #define ZMQ_PLAIN_USERNAME 45
00344 #define ZMQ_PLAIN_PASSWORD 46
00345 #define ZMQ_CURVE_SERVER 47
00346 #define ZMQ_CURVE_PUBLICKEY 48
00347 #define ZMQ_CURVE_SECRETKEY 49
00348 #define ZMQ_CURVE_SERVERKEY 50
00349 #define ZMQ_PROBE_ROUTER 51
00350 #define ZMQ_REQ_CORRELATE 52
00351 #define ZMQ_REQ_RELAXED 53
00352 #define ZMQ_CONFLATE 54
00353 #define ZMQ_ZAP_DOMAIN 55
00354 #define ZMQ_ROUTER_HANDOVER 56
00355 #define ZMQ_TOS 57
00356 #define ZMQ_CONNECT_ROUTING_ID 61
00357 #define ZMQ_GSSAPI_SERVER 62
00358 #define ZMQ_GSSAPI_PRINCIPAL 63
00359 #define ZMQ_GSSAPI_SERVICE_PRINCIPAL 64
00360 #define ZMQ_GSSAPI_PLAINTEXT 65
00361 #define ZMQ_HANDSHAKE_IVL 66
00362 #define ZMQ_SOCKS_PROXY 68
00363 #define ZMQ_XPUB_NODROP 69
00364 #define ZMQ_BLOCKY 70
00365 #define ZMQ_XPUB_MANUAL 71
00366 #define ZMQ_XPUB_WELCOME_MSG 72
00367 #define ZMQ_STREAM_NOTIFY 73
00368 #define ZMQ_INVERT_MATCHING 74
00369 #define ZMQ_HEARTBEAT_IVL 75
00370 #define ZMQ_HEARTBEAT_TTL 76
00371 #define ZMQ_HEARTBEAT_TIMEOUT 77
00372 #define ZMQ_XPUB_VERBOSE 78
00373 #define ZMQ_CONNECT_TIMEOUT 79
00374 #define ZMQ_TCP_MAXRT 80
00375 #define ZMQ_THREAD_SAFE 81
00376 #define ZMQ_MULTICAST_MAXTPDU 84
00377 #define ZMQ_VMCI_BUFFER_SIZE 85
00378 #define ZMQ_VMCI_BUFFER_MIN_SIZE 86
00379 #define ZMQ_VMCI_BUFFER_MAX_SIZE 87
00380 #define ZMQ_VMCI_CONNECT_TIMEOUT 88
00381 #define ZMQ_USE_FD 89
00382 #define ZMQ_GSSAPI_PRINCIPAL_NAME_TYPE 90
00383 #define ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAME_TYPE 91
00384 #define ZMQ_BINDTODEVICE 92
00385
00386 /* Message options */
00387 #define ZMQ_MORE 1
00388 #define ZMQ_SHARED 3
00389
00390 /* Send/rcv options. */
00391 #define ZMQ_DONTWAIT 1
00392 #define ZMQ_SNDMORE 2

```

```

00393
00394 /* Security mechanisms */
00395 #define ZMQ_NULL 0
00396 #define ZMQ_PLAIN 1
00397 #define ZMQ_CURVE 2
00398 #define ZMQ_GSSAPI 3
00399
00400 /* RADIO-DISH protocol */
00401 #define ZMQ_GROUP_MAX_LENGTH 255
00402
00403 /* Deprecated options and aliases */
00404 #define ZMQ_IDENTITY ZMQ_ROUTING_ID
00405 #define ZMQ_CONNECT_RID ZMQ_CONNECT_ROUTING_ID
00406 #define ZMQ_TCP_ACCEPT_FILTER 38
00407 #define ZMQ_IPC_FILTER_PID 58
00408 #define ZMQ_IPC_FILTER_UID 59
00409 #define ZMQ_IPC_FILTER_GID 60
00410 #define ZMQ_IPV4ONLY 31
00411 #define ZMQ_DELAY_ATTACH_ON_CONNECT ZMQ_IMMEDIATE
00412 #define ZMQ_NOBLOCK ZMQ_DONTWAIT
00413 #define ZMQ_FAIL_UNROUTABLE ZMQ_ROUTER_MANDATORY
00414 #define ZMQ_ROUTER_BEHAVIOR ZMQ_ROUTER_MANDATORY
00415
00416 /* Deprecated Message options */
00417 #define ZMQ_SRCFD 2
00418
00419 /*****
00420 /* GSSAPI definitions */
00421 /*****
00422
00423 /* GSSAPI principal name types */
00424 #define ZMQ_GSSAPI_NT_HOSTBASED 0
00425 #define ZMQ_GSSAPI_NT_USER_NAME 1
00426 #define ZMQ_GSSAPI_NT_KRB5_PRINCIPAL 2
00427
00428 /*****
00429 /* OMQ socket events and monitoring */
00430 /*****
00431
00432 /* Socket transport events (TCP, IPC and TIPC only) */
00433
00434 #define ZMQ_EVENT_CONNECTED 0x0001
00435 #define ZMQ_EVENT_CONNECT_DELAYED 0x0002
00436 #define ZMQ_EVENT_CONNECT_RETRIED 0x0004
00437 #define ZMQ_EVENT_LISTENING 0x0008
00438 #define ZMQ_EVENT_BIND_FAILED 0x0010
00439 #define ZMQ_EVENT_ACCEPTED 0x0020
00440 #define ZMQ_EVENT_ACCEPT_FAILED 0x0040
00441 #define ZMQ_EVENT_CLOSED 0x0080
00442 #define ZMQ_EVENT_CLOSE_FAILED 0x0100
00443 #define ZMQ_EVENT_DISCONNECTED 0x0200
00444 #define ZMQ_EVENT_MONITOR_STOPPED 0x0400
00445 #define ZMQ_EVENT_ALL 0xFFFF
00446 /* Unspecified system errors during handshake. Event value is an errno. */
00447 #define ZMQ_EVENT_HANDSHAKE_FAILED_NO_DETAIL 0x0800
00448 /* Handshake complete successfully with successful authentication (if
00449 * enabled). Event value is unused. */
00450 #define ZMQ_EVENT_HANDSHAKE_SUCCEEDED 0x1000
00451 /* Protocol errors between ZMTP peers or between server and ZAP handler.
00452 * Event value is one of ZMQ_PROTOCOL_ERROR_* */
00453 #define ZMQ_EVENT_HANDSHAKE_FAILED_PROTOCOL 0x2000
00454 /* Failed authentication requests. Event value is the numeric ZAP status
00455 * code, i.e. 300, 400 or 500. */
00456 #define ZMQ_EVENT_HANDSHAKE_FAILED_AUTH 0x4000
00457 #define ZMQ_PROTOCOL_ERROR_ZMTP_UNSPECIFIED 0x10000000
00458 #define ZMQ_PROTOCOL_ERROR_ZMTP_UNEXPECTED_COMMAND 0x10000001
00459 #define ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_SEQUENCE 0x10000002
00460 #define ZMQ_PROTOCOL_ERROR_ZMTP_KEY_EXCHANGE 0x10000003
00461 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_UNSPECIFIED 0x10000011
00462 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_MESSAGE 0x10000012
00463 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_HELLO 0x10000013
00464 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_INITIATE 0x10000014
00465 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_ERROR 0x10000015
00466 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_READY 0x10000016
00467 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_WELCOME 0x10000017
00468 #define ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_METADATA 0x10000018
00469 // the following two may be due to erroneous configuration of a peer
00470 #define ZMQ_PROTOCOL_ERROR_ZMTP_CRYPTOGRAPHIC 0x11000001
00471 #define ZMQ_PROTOCOL_ERROR_ZMTP_MECHANISM_MISMATCH 0x11000002
00472 #define ZMQ_PROTOCOL_ERROR_ZAP_UNSPECIFIED 0x20000000
00473 #define ZMQ_PROTOCOL_ERROR_ZAP_MALFORMED_REPLY 0x20000001
00474 #define ZMQ_PROTOCOL_ERROR_ZAP_BAD_REQUEST_ID 0x20000002
00475 #define ZMQ_PROTOCOL_ERROR_ZAP_BAD_VERSION 0x20000003
00476 #define ZMQ_PROTOCOL_ERROR_ZAP_INVALID_STATUS_CODE 0x20000004
00477 #define ZMQ_PROTOCOL_ERROR_ZAP_INVALID_METADATA 0x20000005
00478 #define ZMQ_PROTOCOL_ERROR_WS_UNSPECIFIED 0x30000000
00479

```

```

00480 ZMQ_EXPORT void *zmq_socket (void *, int type_);
00481 ZMQ_EXPORT int zmq_close (void *s_);
00482 ZMQ_EXPORT int
00483 zmq_setsockopt (void *s_, int option_, const void *optval_, size_t optvallen_);
00484 ZMQ_EXPORT int
00485 zmq_getsockopt (void *s_, int option_, void *optval_, size_t *optvallen_);
00486 ZMQ_EXPORT int zmq_bind (void *s_, const char *addr_);
00487 ZMQ_EXPORT int zmq_connect (void *s_, const char *addr_);
00488 ZMQ_EXPORT int zmq_unbind (void *s_, const char *addr_);
00489 ZMQ_EXPORT int zmq_disconnect (void *s_, const char *addr_);
00490 ZMQ_EXPORT int zmq_send (void *s_, const void *buf_, size_t len_, int flags_);
00491 ZMQ_EXPORT int
00492 zmq_send_const (void *s_, const void *buf_, size_t len_, int flags_);
00493 ZMQ_EXPORT int zmq_recv (void *s_, void *buf_, size_t len_, int flags_);
00494 ZMQ_EXPORT int zmq_socket_monitor (void *s_, const char *addr_, int events_);
00495
00496 /*****
00497  * Hide socket fd type; this was before zmq_poller_event_t typedef below
00498  */
00499 /*****
00500  #if defined _WIN32
00501  // Windows uses a pointer-sized unsigned integer to store the socket fd.
00502  #if defined _WIN64
00503  typedef unsigned __int64 zmq_fd_t;
00504  #else
00505  typedef unsigned int zmq_fd_t;
00506  #endif
00507  #else
00508  typedef int zmq_fd_t;
00509  #endif
00510
00511  /*****
00512  *  Deprecated I/O multiplexing. Prefer using zmq_poller API
00513  */
00514  /*****
00515  #define ZMQ_POLLIN 1
00516  #define ZMQ_POLLOUT 2
00517  #define ZMQ_POLLERR 4
00518  #define ZMQ_POLLPRI 8
00519
00520  typedef struct zmq_pollitem_t
00521  {
00522      void *socket;
00523      zmq_fd_t fd;
00524      short events;
00525      short revents;
00526  } zmq_pollitem_t;
00527
00528  #define ZMQ_POLLITEMS_DFLT 16
00529
00530  ZMQ_EXPORT int zmq_poll (zmq_pollitem_t *items_, int nitems_, long timeout_);
00531
00532  /*****
00533  *  Message proxying
00534  */
00535  /*****
00536  ZMQ_EXPORT int zmq_proxy (void *frontend_, void *backend_, void *capture_);
00537  ZMQ_EXPORT int zmq_proxy_steerable (void *frontend_,
00538                                     void *backend_,
00539                                     void *capture_,
00540                                     void *control_);
00541
00542  /*****
00543  *  Probe library capabilities
00544  */
00545  /*****
00546  #define ZMQ_HAS_CAPABILITIES 1
00547  ZMQ_EXPORT int zmq_has (const char *capability_);
00548
00549  /*  Deprecated aliases */
00550  #define ZMQ_STREAMER 1
00551  #define ZMQ_FORWARDER 2
00552  #define ZMQ_QUEUE 3
00553
00554  /*  Deprecated methods */
00555  ZMQ_EXPORT int zmq_device (int type_, void *frontend_, void *backend_);
00556  ZMQ_EXPORT int zmq_sendmsg (void *s_, zmq_msg_t *msg_, int flags_);
00557  ZMQ_EXPORT int zmq_recvmsg (void *s_, zmq_msg_t *msg_, int flags_);
00558  struct iovec;
00559  ZMQ_EXPORT int
00560  zmq_sendiov (void *s_, struct iovec *iov_, size_t count_, int flags_);
00561  ZMQ_EXPORT int
00562  zmq_recviov (void *s_, struct iovec *iov_, size_t *count_, int flags_);
00563
00564  /*****
00565  *  Encryption functions
00566  */
00567  /*****

```

```

00567
00568 /* Encode data with Z85 encoding. Returns encoded data */
00569 ZMQ_EXPORT char *
00570 zmq_z85_encode (char *dest_, const uint8_t *data_, size_t size_);
00571
00572 /* Decode data with Z85 encoding. Returns decoded data */
00573 ZMQ_EXPORT uint8_t *zmq_z85_decode (uint8_t *dest_, const char *string_);
00574
00575 /* Generate z85-encoded public and private keypair with tweetnacl/libsodium. */
00576 /* Returns 0 on success. */
00577 ZMQ_EXPORT int zmq_curve_keypair (char *z85_public_key_, char *z85_secret_key_);
00578
00579 /* Derive the z85-encoded public key from the z85-encoded secret key. */
00580 /* Returns 0 on success. */
00581 ZMQ_EXPORT int zmq_curve_public (char *z85_public_key_,
00582                                 const char *z85_secret_key_);
00583
00584 /*****
00585 /* Atomic utility methods */
00586 /*****
00587
00588 ZMQ_EXPORT void *zmq_atomic_counter_new (void);
00589 ZMQ_EXPORT void zmq_atomic_counter_set (void *counter_, int value_);
00590 ZMQ_EXPORT int zmq_atomic_counter_inc (void *counter_);
00591 ZMQ_EXPORT int zmq_atomic_counter_dec (void *counter_);
00592 ZMQ_EXPORT int zmq_atomic_counter_value (void *counter_);
00593 ZMQ_EXPORT void zmq_atomic_counter_destroy (void **counter_p_);
00594
00595 /*****
00596 /* Scheduling timers */
00597 /*****
00598
00599 #define ZMQ_HAVE_TIMERS
00600
00601 typedef void(zmq_timer_fn) (int timer_id, void *arg);
00602
00603 ZMQ_EXPORT void *zmq_timers_new (void);
00604 ZMQ_EXPORT int zmq_timers_destroy (void **timers_p);
00605 ZMQ_EXPORT int
00606 zmq_timers_add (void *timers, size_t interval, zmq_timer_fn handler, void *arg);
00607 ZMQ_EXPORT int zmq_timers_cancel (void *timers, int timer_id);
00608 ZMQ_EXPORT int
00609 zmq_timers_set_interval (void *timers, int timer_id, size_t interval);
00610 ZMQ_EXPORT int zmq_timers_reset (void *timers, int timer_id);
00611 ZMQ_EXPORT long zmq_timers_timeout (void *timers);
00612 ZMQ_EXPORT int zmq_timers_execute (void *timers);
00613
00614
00615 /*****
00616 /* These functions are not documented by man pages -- use at your own risk. */
00617 /* If you need these to be part of the formal ZMQ API, then (a) write a man */
00618 /* page, and (b) write a test case in tests. */
00619 /*****
00620
00621 /* Helper functions are used by perf tests so that they don't have to care */
00622 /* about minutiae of time-related functions on different OS platforms. */
00623
00624 /* Starts the stopwatch. Returns the handle to the watch. */
00625 ZMQ_EXPORT void *zmq_stopwatch_start (void);
00626
00627 /* Returns the number of microseconds elapsed since the stopwatch was */
00628 /* started, but does not stop or deallocate the stopwatch. */
00629 ZMQ_EXPORT unsigned long zmq_stopwatch_intermediate (void *watch_);
00630
00631 /* Stops the stopwatch. Returns the number of microseconds elapsed since */
00632 /* the stopwatch was started, and deallocates that watch. */
00633 ZMQ_EXPORT unsigned long zmq_stopwatch_stop (void *watch_);
00634
00635 /* Sleeps for specified number of seconds. */
00636 ZMQ_EXPORT void zmq_sleep (int seconds_);
00637
00638 typedef void(zmq_thread_fn) (void *);
00639
00640 /* Start a thread. Returns a handle to the thread. */
00641 ZMQ_EXPORT void *zmq_threadstart (zmq_thread_fn *func_, void *arg_);
00642
00643 /* Wait for thread to complete then free up resources. */
00644 ZMQ_EXPORT void zmq_threadclose (void *thread_);
00645
00646
00647 /*****
00648 /* These functions are DRAFT and disabled in stable releases, and subject to */
00649 /* change at ANY time until declared stable. */
00650 /*****
00651
00652 #ifndef ZMQ_BUILD_DRAFT_API
00653

```



```

00654 /* DRAFT Socket types. */
00655 #define ZMQ_SERVER 12
00656 #define ZMQ_CLIENT 13
00657 #define ZMQ_RADIO 14
00658 #define ZMQ_DISH 15
00659 #define ZMQ_GATHER 16
00660 #define ZMQ_SCATTER 17
00661 #define ZMQ_DGRAM 18
00662 #define ZMQ_PEER 19
00663 #define ZMQ_CHANNEL 20
00664
00665 /* DRAFT Socket options. */
00666 #define ZMQ_ZAP_ENFORCE_DOMAIN 93
00667 #define ZMQ_LOOPBACK_FASTPATH 94
00668 #define ZMQ_METADATA 95
00669 #define ZMQ_MULTICAST_LOOP 96
00670 #define ZMQ_ROUTER_NOTIFY 97
00671 #define ZMQ_XPUB_MANUAL_LAST_VALUE 98
00672 #define ZMQ SOCKS_USERNAME 99
00673 #define ZMQ SOCKS_PASSWORD 100
00674 #define ZMQ_IN_BATCH_SIZE 101
00675 #define ZMQ_OUT_BATCH_SIZE 102
00676 #define ZMQ_WSS_KEY_PEM 103
00677 #define ZMQ_WSS_CERT_PEM 104
00678 #define ZMQ_WSS_TRUST_PEM 105
00679 #define ZMQ_WSS_HOSTNAME 106
00680 #define ZMQ_WSS_TRUST_SYSTEM 107
00681 #define ZMQ_ONLY_FIRST_SUBSCRIBE 108
00682 #define ZMQ_RECONNECT_STOP 109
00683 #define ZMQ_HELLO_MSG 110
00684 #define ZMQ_DISCONNECT_MSG 111
00685 #define ZMQ_PRIORITY 112
00686
00687 /* DRAFT ZMQ_RECONNECT_STOP options */
00688 #define ZMQ_RECONNECT_STOP_CONN_REFUSED 0x1
00689 #define ZMQ_RECONNECT_STOP_HANDSHAKE_FAILED 0x2
00690 #define ZMQ_RECONNECT_STOP_AFTER_DISCONNECT 0x3
00691
00692 /* DRAFT Context options */
00693 #define ZMQ_ZERO_COPY_RECV 10
00694
00695 /* DRAFT Context methods. */
00696 ZMQ_EXPORT int zmq_ctx_set_ext (void *context_,
00697                                int option_,
00698                                const void *optval_,
00699                                size_t optvallen_);
00700 ZMQ_EXPORT int zmq_ctx_get_ext (void *context_,
00701                                int option_,
00702                                void *optval_,
00703                                size_t *optvallen_);
00704
00705 /* DRAFT Socket methods. */
00706 ZMQ_EXPORT int zmq_join (void *s, const char *group);
00707 ZMQ_EXPORT int zmq_leave (void *s, const char *group);
00708 ZMQ_EXPORT uint32_t zmq_connect_peer (void *s_, const char *addr_);
00709
00710 /* DRAFT Msg methods. */
00711 ZMQ_EXPORT int zmq_msg_set_routing_id (zmq_msg_t *msg, uint32_t routing_id);
00712 ZMQ_EXPORT uint32_t zmq_msg_routing_id (zmq_msg_t *msg);
00713 ZMQ_EXPORT int zmq_msg_set_group (zmq_msg_t *msg, const char *group);
00714 ZMQ_EXPORT const char *zmq_msg_group (zmq_msg_t *msg);
00715 ZMQ_EXPORT int
00716 zmq_msg_init_buffer (zmq_msg_t *msg_, const void *buf_, size_t size_);
00717
00718 /* DRAFT Msg property names. */
00719 #define ZMQ_MSG_PROPERTY_ROUTING_ID "Routing-Id"
00720 #define ZMQ_MSG_PROPERTY_SOCKET_TYPE "Socket-Type"
00721 #define ZMQ_MSG_PROPERTY_USER_ID "User-Id"
00722 #define ZMQ_MSG_PROPERTY_PEER_ADDRESS "Peer-Address"
00723
00724 /* Router notify options */
00725 #define ZMQ_NOTIFY_CONNECT 1
00726 #define ZMQ_NOTIFY_DISCONNECT 2
00727
00728 /*****
00729 /* Poller polling on sockets, fd and thread-safe sockets */
00730 *****/
00731
00732 #define ZMQ_HAVE_POLLER
00733
00734 typedef struct zmq_poller_event_t
00735 {
00736     void *socket;
00737     zmq_fd_t fd;
00738     void *user_data;
00739     short events;
00740 } zmq_poller_event_t;

```

```

00741
00742 ZMQ_EXPORT void *zmq_poller_new (void);
00743 ZMQ_EXPORT int zmq_poller_destroy (void **poller_p);
00744 ZMQ_EXPORT int zmq_poller_size (void *poller);
00745 ZMQ_EXPORT int
00746 zmq_poller_add (void *poller, void *socket, void *user_data, short events);
00747 ZMQ_EXPORT int zmq_poller_modify (void *poller, void *socket, short events);
00748 ZMQ_EXPORT int zmq_poller_remove (void *poller, void *socket);
00749 ZMQ_EXPORT int
00750 zmq_poller_wait (void *poller, zmq_poller_event_t *event, long timeout);
00751 ZMQ_EXPORT int zmq_poller_wait_all (void *poller,
00752                                     zmq_poller_event_t *events,
00753                                     int n_events,
00754                                     long timeout);
00755 ZMQ_EXPORT int zmq_poller_fd (void *poller, zmq_fd_t *fd);
00756
00757 ZMQ_EXPORT int
00758 zmq_poller_add_fd (void *poller, zmq_fd_t fd, void *user_data, short events);
00759 ZMQ_EXPORT int zmq_poller_modify_fd (void *poller, zmq_fd_t fd, short events);
00760 ZMQ_EXPORT int zmq_poller_remove_fd (void *poller, zmq_fd_t fd);
00761
00762 ZMQ_EXPORT int zmq_socket_get_peer_state (void *socket,
00763                                           const void *routing_id,
00764                                           size_t routing_id_size);
00765
00766 /* DRAFT Socket monitoring events */
00767 #define ZMQ_EVENT_PIPES_STATS 0x10000
00768
00769 #define ZMQ_CURRENT_EVENT_VERSION 1
00770 #define ZMQ_CURRENT_EVENT_VERSION_DRAFT 2
00771
00772 #define ZMQ_EVENT_ALL_V1 ZMQ_EVENT_ALL
00773 #define ZMQ_EVENT_ALL_V2 ZMQ_EVENT_ALL_V1 | ZMQ_EVENT_PIPES_STATS
00774
00775 ZMQ_EXPORT int zmq_socket_monitor_versioned (
00776     void *s_, const char *addr_, uint64_t events_, int event_version_, int type_);
00777 ZMQ_EXPORT int zmq_socket_monitor_pipes_stats (void *s);
00778
00779 #endif // ZMQ_BUILD_DRAFT_API
00780
00781
00782 #undef ZMQ_EXPORT
00783
00784 #ifdef __cplusplus
00785 }
00786 #endif
00787
00788 #endif

```

7.27 external/zmq/includes/zmq/zmq.hpp File Reference

```

#include "zmq.h"
#include <cassert>
#include <cstring>
#include <algorithm>
#include <exception>
#include <iomanip>
#include <sstream>
#include <string>
#include <vector>

```

Classes

- struct [zmq_event_t](#)
- class [zmq::error_t](#)
- class [zmq::message_t](#)
- class [zmq::context_t](#)
- class [zmq::detail::socket_base](#)
- struct [zmq::from_handle_t](#)
- struct [zmq::from_handle_t::private](#)
- class [zmq::socket_ref](#)
- class [zmq::socket_t](#)

- class [zmq::monitor_t](#)

Namespaces

- namespace [zmq](#)
- namespace [zmq::detail](#)

Macros

- #define [CPPZMQ_LANG](#) __cplusplus
- #define [ZMQ_DEPRECATED](#)(msg)
- #define [ZMQ_NODISCARD](#)
- #define [ZMQ_NOTHROW](#) throw()
- #define [ZMQ_EXPLICIT](#)
- #define [ZMQ_OVERRIDE](#)
- #define [ZMQ_NULLPTR](#) 0
- #define [ZMQ_CONSTEXPR_FN](#)
- #define [ZMQ_CONSTEXPR_VAR](#) const
- #define [ZMQ_CPP11_DEPRECATED](#)(msg)
- #define [ZMQ_INLINE_VAR](#)
- #define [ZMQ_CONSTEXPR_IF](#)
- #define [CPPZMQ_HAS_INCLUDE_CPP17](#)(X) 0
- #define [CPPZMQ_HAS_OPTIONAL](#) 0
- #define [CPPZMQ_HAS_STRING_VIEW](#) 0
- #define [CPPZMQ_VERSION_MAJOR](#) 4
- #define [CPPZMQ_VERSION_MINOR](#) 10
- #define [CPPZMQ_VERSION_PATCH](#) 0
- #define [CPPZMQ_VERSION](#)
- #define [ZMQ_DELETED_FUNCTION](#)
- #define [ZMQ_NEW_MONITOR_EVENT_LAYOUT](#)
- #define [ZMQ_HAS_PROXY_STEERABLE](#)
- #define [ZMQ_ASSERT](#)(expression) assert(expression)

Typedefs

- typedef [zmq_free_fn](#) [zmq::free_fn](#)
- typedef [zmq_pollitem_t](#) [zmq::pollitem_t](#)
- typedef int [zmq::fd_t](#)

Functions

- int [zmq::detail::poll](#) ([zmq_pollitem_t](#) *items_, size_t nitems_, long timeout_)
- int [zmq::poll](#) ([zmq_pollitem_t](#) *items_, size_t nitems_, long timeout_=-1)
- int [zmq::poll](#) ([zmq_pollitem_t](#) const *items_, size_t nitems_, long timeout_=-1)
- void [zmq::version](#) (int *major_, int *minor_, int *patch_)
- void [zmq::swap](#) ([message_t](#) &a, [message_t](#) &b) [ZMQ_NOTHROW](#)
- void [zmq::swap](#) ([context_t](#) &a, [context_t](#) &b) [ZMQ_NOTHROW](#)
- bool [zmq::operator==](#) (const [detail::socket_base](#) &a, const [detail::socket_base](#) &b) [ZMQ_NOTHROW](#)
- bool [zmq::operator!=](#) (const [detail::socket_base](#) &a, const [detail::socket_base](#) &b) [ZMQ_NOTHROW](#)
- bool [zmq::operator<](#) (const [detail::socket_base](#) &a, const [detail::socket_base](#) &b) [ZMQ_NOTHROW](#)
- bool [zmq::operator>](#) (const [detail::socket_base](#) &a, const [detail::socket_base](#) &b) [ZMQ_NOTHROW](#)
- bool [zmq::operator<=](#) (const [detail::socket_base](#) &a, const [detail::socket_base](#) &b) [ZMQ_NOTHROW](#)
- bool [zmq::operator>=](#) (const [detail::socket_base](#) &a, const [detail::socket_base](#) &b) [ZMQ_NOTHROW](#)
- void [zmq::swap](#) ([socket_t](#) &a, [socket_t](#) &b) [ZMQ_NOTHROW](#)
- void [zmq::proxy](#) (void *frontend, void *backend, void *capture)
- void [zmq::proxy](#) ([socket_ref](#) frontend, [socket_ref](#) backend, [socket_ref](#) capture=[socket_ref](#)())
- void [zmq::proxy_steerable](#) (void *frontend, void *backend, void *capture, void *control)
- void [zmq::proxy_steerable](#) ([socket_ref](#) frontend, [socket_ref](#) backend, [socket_ref](#) capture, [socket_ref](#) control)
- std::ostream & [zmq::operator<<](#) (std::ostream &os, const [message_t](#) &msg)

Variables

- [ZMQ_CONSTEXPR_VAR](#) from `handle_t` `zmq::from_handle`

7.27.1 Macro Definition Documentation**7.27.1.1 CPPZMQ_HAS_INCLUDE_CPP17**

```
#define CPPZMQ_HAS_INCLUDE_CPP17(  
    X ) 0
```

Definition at line 127 of file [zmq.hpp](#).

7.27.1.2 CPPZMQ_HAS_OPTIONAL

```
#define CPPZMQ_HAS_OPTIONAL 0
```

Definition at line 134 of file [zmq.hpp](#).

7.27.1.3 CPPZMQ_HAS_STRING_VIEW

```
#define CPPZMQ_HAS_STRING_VIEW 0
```

Definition at line 143 of file [zmq.hpp](#).

7.27.1.4 CPPZMQ_LANG

```
#define CPPZMQ_LANG __cplusplus
```

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

7.27.1.5 CPPZMQ_VERSION

```
#define CPPZMQ_VERSION
```

Value:

```
ZMQ_MAKE_VERSION(CPPZMQ_VERSION_MAJOR, CPPZMQ_VERSION_MINOR,  
                  CPPZMQ_VERSION_PATCH)
```

Definition at line 153 of file [zmq.hpp](#).

7.27.1.6 CPPZMQ_VERSION_MAJOR

```
#define CPPZMQ_VERSION_MAJOR 4
```

Definition at line 149 of file [zmq.hpp](#).

7.27.1.7 CPPZMQ_VERSION_MINOR

```
#define CPPZMQ_VERSION_MINOR 10
```

Definition at line 150 of file [zmq.hpp](#).

7.27.1.8 CPPZMQ_VERSION_PATCH

```
#define CPPZMQ_VERSION_PATCH 0
```

Definition at line 151 of file [zmq.hpp](#).

7.27.1.9 ZMQ_ASSERT

```
#define ZMQ_ASSERT(  
    expression ) assert(expression)
```

Definition at line 223 of file [zmq.hpp](#).

7.27.1.10 ZMQ_CONSTEXPR_FN

```
#define ZMQ_CONSTEXPR_FN
```

Definition at line 93 of file [zmq.hpp](#).

7.27.1.11 ZMQ_CONSTEXPR_IF

```
#define ZMQ_CONSTEXPR_IF
```

Definition at line 105 of file [zmq.hpp](#).

7.27.1.12 ZMQ_CONSTEXPR_VAR

```
#define ZMQ_CONSTEXPR_VAR const
```

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

7.27.1.13 ZMQ_CPP11_DEPRECATED

```
#define ZMQ_CPP11_DEPRECATED(  
    msg )
```

Definition at line 95 of file [zmq.hpp](#).

7.27.1.14 ZMQ_DELETED_FUNCTION

```
#define ZMQ_DELETED_FUNCTION
```

Definition at line 179 of file [zmq.hpp](#).

7.27.1.15 ZMQ_DEPRECATED

```
#define ZMQ_DEPRECATED(  
    msg )
```

Definition at line 71 of file [zmq.hpp](#).

7.27.1.16 ZMQ_EXPLICIT

```
#define ZMQ_EXPLICIT
```

Definition at line 90 of file [zmq.hpp](#).

7.27.1.17 ZMQ_HAS_PROXY_STEERABLE

```
#define ZMQ_HAS_PROXY_STEERABLE
```

Definition at line 205 of file [zmq.hpp](#).

7.27.1.18 ZMQ_INLINE_VAR

```
#define ZMQ_INLINE_VAR
```

Definition at line 104 of file [zmq.hpp](#).

7.27.1.19 ZMQ_NEW_MONITOR_EVENT_LAYOUT

```
#define ZMQ_NEW_MONITOR_EVENT_LAYOUT
```

Definition at line 201 of file [zmq.hpp](#).

7.27.1.20 ZMQ_NODISCARD

```
#define ZMQ_NODISCARD
```

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

7.27.1.21 ZMQ_NOTHROW

```
#define ZMQ_NOTHROW throw()
```

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

7.27.1.22 ZMQ_NULLPTR

```
#define ZMQ_NULLPTR 0
```

Definition at line 92 of file [zmq.hpp](#).

7.27.1.23 ZMQ_OVERRIDE

```
#define ZMQ_OVERRIDE
```

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

7.28 zmq.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002     Copyright (c) 2016-2017 ZeroMQ community
00003     Copyright (c) 2009-2011 250bpm s.r.o.
00004     Copyright (c) 2011 Botond Ballo
00005     Copyright (c) 2007-2009 iMatix Corporation
00006
00007     Permission is hereby granted, free of charge, to any person obtaining a copy
00008     of this software and associated documentation files (the "Software"), to
00009     deal in the Software without restriction, including without limitation the
00010     rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
00011     sell copies of the Software, and to permit persons to whom the Software is
00012     furnished to do so, subject to the following conditions:
00013
00014     The above copyright notice and this permission notice shall be included in
00015     all copies or substantial portions of the Software.
00016
00017     THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00018     IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00019     FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00020     AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00021     LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
00022     FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
00023     IN THE SOFTWARE.
00024 */
00025
00026 #ifndef __ZMQ_HPP_INCLUDED__
00027 #define __ZMQ_HPP_INCLUDED__
00028
00029 #ifdef _WIN32
00030 #ifndef NOMINMAX
00031 #define NOMINMAX
00032 #endif
00033 #endif
00034
00035 // included here for _HAS_CXX* macros
00036 #include "zmq.h"
00037
00038 #if defined(_MSVC_LANG)
00039 #define CPPZMQ_LANG _MSVC_LANG
00040 #else
00041 #define CPPZMQ_LANG __cplusplus
00042 #endif
00043 // overwrite if specific language macros indicate higher version
00044 #if defined(_HAS_CXX14) && _HAS_CXX14 && CPPZMQ_LANG < 201402L
00045 #undef CPPZMQ_LANG
00046 #define CPPZMQ_LANG 201402L
00047 #endif
00048 #if defined(_HAS_CXX17) && _HAS_CXX17 && CPPZMQ_LANG < 201703L
00049 #undef CPPZMQ_LANG
00050 #define CPPZMQ_LANG 201703L
00051 #endif
00052
00053 // macros defined if has a specific standard or greater
00054 #if CPPZMQ_LANG >= 201103L || (defined(_MSC_VER) && _MSC_VER >= 1900)
00055 #define ZMQ_CPP11
00056 #endif
00057 #if CPPZMQ_LANG >= 201402L
00058 #define ZMQ_CPP14
00059 #endif
00060 #if CPPZMQ_LANG >= 201703L
00061 #define ZMQ_CPP17
00062 #endif
00063
00064 #if defined(ZMQ_CPP14) && !defined(_MSC_VER)
00065 #define ZMQ_DEPRECATED(msg) [[deprecated(msg)]]
00066 #elif defined(_MSC_VER)
00067 #define ZMQ_DEPRECATED(msg) __declspec(deprecated(msg))
00068 #elif defined(__GNUC__)
00069 #define ZMQ_DEPRECATED(msg) __attribute__((deprecated(msg)))
00070 #else
00071 #define ZMQ_DEPRECATED(msg)
00072 #endif
00073
00074 #if defined(ZMQ_CPP17)
00075 #define ZMQ_NODISCARD [[nodiscard]]
```

```

00076 #else
00077 #define ZMQ_NODISCARD
00078 #endif
00079
00080 #if defined(ZMQ_CPP11)
00081 #define ZMQ_NOTHROW noexcept
00082 #define ZMQ_EXPLICIT explicit
00083 #define ZMQ_OVERRIDE override
00084 #define ZMQ_NULLPTR nullptr
00085 #define ZMQ_CONSTEXPR_FN constexpr
00086 #define ZMQ_CONSTEXPR_VAR constexpr
00087 #define ZMQ_CPP11_DEPRECATED(msg) ZMQ_DEPRECATED(msg)
00088 #else
00089 #define ZMQ_NOTHROW throw()
00090 #define ZMQ_EXPLICIT
00091 #define ZMQ_OVERRIDE
00092 #define ZMQ_NULLPTR 0
00093 #define ZMQ_CONSTEXPR_FN
00094 #define ZMQ_CONSTEXPR_VAR const
00095 #define ZMQ_CPP11_DEPRECATED(msg)
00096 #endif
00097 #if defined(ZMQ_CPP14) && (!defined(_MSC_VER) || _MSC_VER > 1900) && (!defined(__GNUC__) || __GNUC__ >
    5 || (__GNUC__ == 5 && __GNUC_MINOR__ > 3))
00098 #define ZMQ_EXTENDED_CONSTEXPR
00099 #endif
00100 #if defined(ZMQ_CPP17)
00101 #define ZMQ_INLINE_VAR inline
00102 #define ZMQ_CONSTEXPR_IF constexpr
00103 #else
00104 #define ZMQ_INLINE_VAR
00105 #define ZMQ_CONSTEXPR_IF
00106 #endif
00107
00108 #include <cassert>
00109 #include <cstring>
00110
00111 #include <algorithm>
00112 #include <exception>
00113 #include <iomanip>
00114 #include <sstream>
00115 #include <string>
00116 #include <vector>
00117 #ifdef ZMQ_CPP11
00118 #include <array>
00119 #include <chrono>
00120 #include <tuple>
00121 #include <memory>
00122 #endif
00123
00124 #if defined(__has_include) && defined(ZMQ_CPP17)
00125 #define CPPZMQ_HAS_INCLUDE_CPP17(X) __has_include(X)
00126 #else
00127 #define CPPZMQ_HAS_INCLUDE_CPP17(X) 0
00128 #endif
00129
00130 #if CPPZMQ_HAS_INCLUDE_CPP17(<optional>) && !defined(CPPZMQ_HAS_OPTIONAL)
00131 #define CPPZMQ_HAS_OPTIONAL 1
00132 #endif
00133 #ifndef CPPZMQ_HAS_OPTIONAL
00134 #define CPPZMQ_HAS_OPTIONAL 0
00135 #elif CPPZMQ_HAS_OPTIONAL
00136 #include <optional>
00137 #endif
00138
00139 #if CPPZMQ_HAS_INCLUDE_CPP17(<string_view>) && !defined(CPPZMQ_HAS_STRING_VIEW)
00140 #define CPPZMQ_HAS_STRING_VIEW 1
00141 #endif
00142 #ifndef CPPZMQ_HAS_STRING_VIEW
00143 #define CPPZMQ_HAS_STRING_VIEW 0
00144 #elif CPPZMQ_HAS_STRING_VIEW
00145 #include <string_view>
00146 #endif
00147
00148 /* Version macros for compile-time API version detection */
00149 #define CPPZMQ_VERSION_MAJOR 4
00150 #define CPPZMQ_VERSION_MINOR 10
00151 #define CPPZMQ_VERSION_PATCH 0
00152
00153 #define CPPZMQ_VERSION \
00154     ZMQ_MAKE_VERSION(CPPZMQ_VERSION_MAJOR, CPPZMQ_VERSION_MINOR, \
00155         CPPZMQ_VERSION_PATCH)
00156
00157 // Detect whether the compiler supports C++11 rvalue references.
00158 #if (defined(__GNUC__) && (__GNUC__ > 4 || (__GNUC__ == 4 && __GNUC_MINOR__ > 2)) \
00159     && defined(__GXX_EXPERIMENTAL_CXX0X__))
00160 #define ZMQ_HAS_RVALUE_REFS
00161 #define ZMQ_DELETED_FUNCTION = delete

```

```

00162 #elif defined(__clang__)
00163 #if __has_feature(cxx_rvalue_references)
00164 #define ZMQ_HAS_RVALUE_REFS
00165 #endif
00166
00167 #if __has_feature(cxx_deleted_functions)
00168 #define ZMQ_DELETED_FUNCTION = delete
00169 #else
00170 #define ZMQ_DELETED_FUNCTION
00171 #endif
00172 #elif defined(_MSC_VER) && (_MSC_VER >= 1900)
00173 #define ZMQ_HAS_RVALUE_REFS
00174 #define ZMQ_DELETED_FUNCTION = delete
00175 #elif defined(_MSC_VER) && (_MSC_VER >= 1600)
00176 #define ZMQ_HAS_RVALUE_REFS
00177 #define ZMQ_DELETED_FUNCTION
00178 #else
00179 #define ZMQ_DELETED_FUNCTION
00180 #endif
00181
00182 #if defined(ZMQ_CPP11) && !defined(__llvm__) && !defined(__INTEL_COMPILER) \
00183     && defined(__GNUC__) && __GNUC__ < 5
00184 #define ZMQ_CPP11_PARTIAL
00185 #elif defined(__GLIBCXX__) && __GLIBCXX__ < 20160805
00186 //the date here is the last date of gcc 4.9.4, which
00187 // effectively means libstdc++ from gcc 5.5 and higher won't trigger this branch
00188 #define ZMQ_CPP11_PARTIAL
00189 #endif
00190
00191 #ifdef ZMQ_CPP11
00192 #ifdef ZMQ_CPP11_PARTIAL
00193 #define ZMQ_IS_TRIVIALLY_COPYABLE(T) __has_trivial_copy(T)
00194 #else
00195 #include <type_traits>
00196 #define ZMQ_IS_TRIVIALLY_COPYABLE(T) std::is_trivially_copyable<T>::value
00197 #endif
00198 #endif
00199
00200 #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(3, 3, 0)
00201 #define ZMQ_NEW_MONITOR_EVENT_LAYOUT
00202 #endif
00203
00204 #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 1, 0)
00205 #define ZMQ_HAS_PROXY_STEERABLE
00206 /* Socket event data */
00207 typedef struct
00208 {
00209     uint16_t event; // id of the event as bitfield
00210     int32_t value; // value is either error code, fd or reconnect interval
00211 } zmq_event_t;
00212 #endif
00213
00214 // Avoid using deprecated message receive function when possible
00215 #if ZMQ_VERSION < ZMQ_MAKE_VERSION(3, 2, 0)
00216 #define zmq_msg_recv(msg, socket, flags) zmq_recvmg(socket, msg, flags)
00217 #endif
00218
00219
00220 // In order to prevent unused variable warnings when building in non-debug
00221 // mode use this macro to make assertions.
00222 #ifndef NDEBUG
00223 #define ZMQ_ASSERT(expression) assert(expression)
00224 #else
00225 #define ZMQ_ASSERT(expression) (void) (expression)
00226 #endif
00227
00228 namespace zmq
00229 {
00230 #ifdef ZMQ_CPP11
00231 namespace detail
00232 {
00233 namespace ranges
00234 {
00235 using std::begin;
00236 using std::end;
00237 template<class T> auto begin(T &r) -> decltype(begin(std::forward<T>(r)))
00238 {
00239     return begin(std::forward<T>(r));
00240 }
00241 template<class T> auto end(T &r) -> decltype(end(std::forward<T>(r)))
00242 {
00243     return end(std::forward<T>(r));
00244 }
00245 } // namespace ranges
00246
00247 template<class T> using void_t = void;
00248

```



```

00249 template<class Iter>
00250 using iter_value_t = typename std::iterator_traits<Iter>::value_type;
00251
00252 template<class Range>
00253 using range_iter_t = decltype(
00254     ranges::begin(std::declval<typename std::remove_reference<Range>::type &>()));
00255
00256 template<class Range> using range_value_t = iter_value_t<range_iter_t<Range>;
00257
00258 template<class T, class = void> struct is_range : std::false_type
00259 {
00260 };
00261
00262 template<class T>
00263 struct is_range<
00264     T,
00265     void_t<decltype(
00266         ranges::begin(std::declval<typename std::remove_reference<T>::type &>())
00267         == ranges::end(std::declval<typename std::remove_reference<T>::type &>()))>
00268     : std::true_type
00269 {
00270 };
00271
00272 } // namespace detail
00273 #endif
00274
00275 typedef zmq_free_fn free_fn;
00276 typedef zmq_pollitem_t pollitem_t;
00277
00278 // duplicate definition from libzmq 4.3.3
00279 #if defined _WIN32
00280 #if defined _WIN64
00281 typedef unsigned __int64 fd_t;
00282 #else
00283 typedef unsigned int fd_t;
00284 #endif
00285 #else
00286 typedef int fd_t;
00287 #endif
00288
00289 class error_t : public std::exception
00290 {
00291 public:
00292     error_t() ZMQ_NOTHROW : errno(zmq_errno()) {}
00293     explicit error_t(int err) ZMQ_NOTHROW : errno(err) {}
00294     virtual const char *what() const ZMQ_NOTHROW ZMQ_OVERRIDE
00295     {
00296         return zmq_strerror(errno);
00297     }
00298     int num() const ZMQ_NOTHROW { return errno; }
00299
00300 private:
00301     int errno;
00302 };
00303
00304 namespace detail {
00305 inline int poll(zmq_pollitem_t *items_, size_t nitems_, long timeout_)
00306 {
00307     int rc = zmq_poll(items_, static_cast<int>(nitems_), timeout_);
00308     if (rc < 0)
00309         throw error_t();
00310     return rc;
00311 }
00312 }
00313
00314 #ifndef ZMQ_CPP11
00315 ZMQ_DEPRECATED("from 4.8.0, use poll taking std::chrono::duration instead of long")
00316 inline int poll(zmq_pollitem_t *items_, size_t nitems_, long timeout_)
00317 #else
00318 inline int poll(zmq_pollitem_t *items_, size_t nitems_, long timeout_ = -1)
00319 #endif
00320 {
00321     return detail::poll(items_, nitems_, timeout_);
00322 }
00323
00324 ZMQ_DEPRECATED("from 4.3.1, use poll taking non-const items")
00325 inline int poll(zmq_pollitem_t const *items_, size_t nitems_, long timeout_ = -1)
00326 {
00327     return detail::poll(const_cast<zmq_pollitem_t *>(items_), nitems_, timeout_);
00328 }
00329
00330 #ifndef ZMQ_CPP11
00331 ZMQ_DEPRECATED("from 4.3.1, use poll taking non-const items")
00332 inline int
00333 poll(zmq_pollitem_t const *items, size_t nitems, std::chrono::milliseconds timeout)
00334 {
00335     return detail::poll(const_cast<zmq_pollitem_t *>(items), nitems,

```

```

00336         static_cast<long>(timeout.count()));
00337     }
00338
00339     ZMQ_DEPRECATED("from 4.3.1, use poll taking non-const items")
00340     inline int poll(std::vector<zmq_pollitem_t> const &items,
00341                   std::chrono::milliseconds timeout)
00342     {
00343         return detail::poll(const_cast<zmq_pollitem_t *>(items.data()), items.size(),
00344                           static_cast<long>(timeout.count()));
00345     }
00346
00347     ZMQ_DEPRECATED("from 4.3.1, use poll taking non-const items")
00348     inline int poll(std::vector<zmq_pollitem_t> const &items, long timeout_ = -1)
00349     {
00350         return detail::poll(const_cast<zmq_pollitem_t *>(items.data()), items.size(), timeout_);
00351     }
00352
00353     inline int
00354     poll(zmq_pollitem_t *items, size_t nitems, std::chrono::milliseconds timeout =
00355          std::chrono::milliseconds{-1})
00356     {
00357         return detail::poll(items, nitems, static_cast<long>(timeout.count()));
00358     }
00359
00360     inline int poll(std::vector<zmq_pollitem_t> &items,
00361                   std::chrono::milliseconds timeout = std::chrono::milliseconds{-1})
00362     {
00363         return detail::poll(items.data(), items.size(), static_cast<long>(timeout.count()));
00364     }
00365
00366     ZMQ_DEPRECATED("from 4.3.1, use poll taking std::chrono::duration instead of long")
00367     inline int poll(std::vector<zmq_pollitem_t> &items, long timeout_)
00368     {
00369         return detail::poll(items.data(), items.size(), timeout_);
00370     }
00371
00372     template<std::size_t SIZE>
00373     inline int poll(std::array<zmq_pollitem_t, SIZE> &items,
00374                   std::chrono::milliseconds timeout = std::chrono::milliseconds{-1})
00375     {
00376         return detail::poll(items.data(), items.size(), static_cast<long>(timeout.count()));
00377     }
00378 #endif
00379
00380     inline void version(int *major_, int *minor_, int *patch_)
00381     {
00382         zmq_version(major_, minor_, patch_);
00383     }
00384
00385 #ifdef ZMQ_CPP11
00386     inline std::tuple<int, int, int> version()
00387     {
00388         std::tuple<int, int, int> v;
00389         zmq_version(&std::get<0>(v), &std::get<1>(v), &std::get<2>(v));
00390         return v;
00391     }
00392
00393 #if !defined(ZMQ_CPP11_PARTIAL)
00394     namespace detail
00395     {
00396         template<class T> struct is_char_type
00397         {
00398             // true if character type for string literals in C++11
00399             static constexpr bool value =
00400                 std::is_same<T, char>::value || std::is_same<T, wchar_t>::value
00401                 || std::is_same<T, char16_t>::value || std::is_same<T, char32_t>::value;
00402         };
00403     }
00404 #endif
00405
00406 #endif
00407
00408     class message_t
00409     {
00410     public:
00411         message_t() ZMQ_NOTHROW
00412         {
00413             int rc = zmq_msg_init(&msg);
00414             ZMQ_ASSERT(rc == 0);
00415         }
00416
00417         explicit message_t(size_t size_)
00418         {
00419             int rc = zmq_msg_init_size(&msg, size_);
00420             if (rc != 0)
00421                 throw error_t();

```

```

00422     }
00423
00424     template<class ForwardIter> message_t(ForwardIter first, ForwardIter last)
00425     {
00426         typedef typename std::iterator_traits<ForwardIter>::value_type value_t;
00427
00428         assert(std::distance(first, last) >= 0);
00429         size_t const size_ =
00430             static_cast<size_t>(std::distance(first, last)) * sizeof(value_t);
00431         int const rc = zmq_msg_init_size(&msg, size_);
00432         if (rc != 0)
00433             throw error_t();
00434         std::copy(first, last, data<value_t>());
00435     }
00436
00437     message_t(const void *data_, size_t size_)
00438     {
00439         int rc = zmq_msg_init_size(&msg, size_);
00440         if (rc != 0)
00441             throw error_t();
00442         if (size_) {
00443             // this constructor allows (nullptr, 0),
00444             // memcpy with a null pointer is UB
00445             memcpy(data(), data_, size_);
00446         }
00447     }
00448
00449     message_t(void *data_, size_t size_, free_fn *ffn_, void *hint_ = ZMQ_NULLPTR)
00450     {
00451         int rc = zmq_msg_init_data(&msg, data_, size_, ffn_, hint_);
00452         if (rc != 0)
00453             throw error_t();
00454     }
00455
00456     // overload set of string-like types and generic containers
00457 #if defined(ZMQ_CPP11) && !defined(ZMQ_CPP11_PARTIAL)
00458     // NOTE this constructor will include the null terminator
00459     // when called with a string literal.
00460     // An overload taking const char* can not be added because
00461     // it would be preferred over this function and break compatibility.
00462     template<
00463         class Char,
00464         size_t N,
00465         typename = typename std::enable_if<detail::is_char_type<Char>::value::type>
00466         ZMQ_DEPRECATED("from 4.7.0, use constructors taking iterators, (pointer, size) "
00467             "or strings instead")
00468     explicit message_t(const Char (&data)[N]) :
00469         message_t(detail::ranges::begin(data), detail::ranges::end(data))
00470     {
00471     }
00472
00473     template<class Range,
00474         typename = typename std::enable_if<
00475             detail::is_range<Range>::value
00476             && ZMQ_IS_TRIVIALY_COPYABLE(detail::range_value_t<Range>)
00477             && !detail::is_char_type<detail::range_value_t<Range>::value>
00478             && !std::is_same<Range, message_t>::value>::type>
00479     explicit message_t(const Range &rng) :
00480         message_t(detail::ranges::begin(rng), detail::ranges::end(rng))
00481     {
00482     }
00483
00484     explicit message_t(const std::string &str) : message_t(str.data(), str.size()) {}
00485
00486 #if CPPZMQ_HAS_STRING_VIEW
00487     explicit message_t(std::string_view str) : message_t(str.data(), str.size()) {}
00488 #endif
00489
00490 #endif
00491
00492 #ifndef ZMQ_HAS_RVALUE_REFS
00493     message_t(message_t &&rhs) ZMQ_NOTHROW : msg(rhs.msg)
00494     {
00495         int rc = zmq_msg_init(&rhs.msg);
00496         ZMQ_ASSERT(rc == 0);
00497     }
00498
00499     message_t &operator=(message_t &&rhs) ZMQ_NOTHROW
00500     {
00501         std::swap(msg, rhs.msg);
00502         return *this;
00503     }
00504 #endif
00505
00506     ~message_t() ZMQ_NOTHROW
00507     {
00508         int rc = zmq_msg_close(&msg);

```

```

00509         ZMQ_ASSERT(rc == 0);
00510     }
00511
00512     void rebuild()
00513     {
00514         int rc = zmq_msg_close(&msg);
00515         if (rc != 0)
00516             throw error_t();
00517         rc = zmq_msg_init(&msg);
00518         ZMQ_ASSERT(rc == 0);
00519     }
00520
00521     void rebuild(size_t size_)
00522     {
00523         int rc = zmq_msg_close(&msg);
00524         if (rc != 0)
00525             throw error_t();
00526         rc = zmq_msg_init_size(&msg, size_);
00527         if (rc != 0)
00528             throw error_t();
00529     }
00530
00531     void rebuild(const void *data_, size_t size_)
00532     {
00533         int rc = zmq_msg_close(&msg);
00534         if (rc != 0)
00535             throw error_t();
00536         rc = zmq_msg_init_size(&msg, size_);
00537         if (rc != 0)
00538             throw error_t();
00539         memcpy(data(), data_, size_);
00540     }
00541
00542     void rebuild(const std::string &str)
00543     {
00544         rebuild(str.data(), str.size());
00545     }
00546
00547     void rebuild(void *data_, size_t size_, free_fn *ffn_, void *hint_ = ZMQ_NULLPTR)
00548     {
00549         int rc = zmq_msg_close(&msg);
00550         if (rc != 0)
00551             throw error_t();
00552         rc = zmq_msg_init_data(&msg, data_, size_, ffn_, hint_);
00553         if (rc != 0)
00554             throw error_t();
00555     }
00556
00557     ZMQ_DEPRECATED("from 4.3.1, use move taking non-const reference instead")
00558     void move(message_t const &msg_)
00559     {
00560         int rc = zmq_msg_move(&msg, const_cast<zmq_msg_t *>(msg_->handle()));
00561         if (rc != 0)
00562             throw error_t();
00563     }
00564
00565     void move(message_t &msg_)
00566     {
00567         int rc = zmq_msg_move(&msg, msg_.handle());
00568         if (rc != 0)
00569             throw error_t();
00570     }
00571
00572     ZMQ_DEPRECATED("from 4.3.1, use copy taking non-const reference instead")
00573     void copy(message_t const &msg_)
00574     {
00575         int rc = zmq_msg_copy(&msg, const_cast<zmq_msg_t *>(msg_->handle()));
00576         if (rc != 0)
00577             throw error_t();
00578     }
00579
00580     void copy(message_t &msg_)
00581     {
00582         int rc = zmq_msg_copy(&msg, msg_.handle());
00583         if (rc != 0)
00584             throw error_t();
00585     }
00586
00587     bool more() const ZMQ_NOTHROW
00588     {
00589         int rc = zmq_msg_more(const_cast<zmq_msg_t *>(&msg));
00590         return rc != 0;
00591     }
00592
00593     void *data() ZMQ_NOTHROW { return zmq_msg_data(&msg); }
00594
00595     const void *data() const ZMQ_NOTHROW

```

```

00596     {
00597         return zmq_msg_data(const_cast<zmq_msg_t *>(&msg));
00598     }
00599
00600     size_t size() const ZMQ_NOTHROW
00601     {
00602         return zmq_msg_size(const_cast<zmq_msg_t *>(&msg));
00603     }
00604
00605     ZMQ_NODISCARD bool empty() const ZMQ_NOTHROW { return size() == 0u; }
00606
00607     template<typename T> T *data() ZMQ_NOTHROW { return static_cast<T *>(data()); }
00608
00609     template<typename T> T const *data() const ZMQ_NOTHROW
00610     {
00611         return static_cast<T const *>(data());
00612     }
00613
00614     ZMQ_DEPRECATED("from 4.3.0, use operator== instead")
00615     bool equal(const message_t *other) const ZMQ_NOTHROW { return *this == *other; }
00616
00617     bool operator==(const message_t &other) const ZMQ_NOTHROW
00618     {
00619         const size_t my_size = size();
00620         return my_size == other.size() && 0 == memcmp(data(), other.data(), my_size);
00621     }
00622
00623     bool operator!=(const message_t &other) const ZMQ_NOTHROW
00624     {
00625         return !(*this == other);
00626     }
00627
00628     #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(3, 2, 0)
00629     int get(int property_)
00630     {
00631         int value = zmq_msg_get(&msg, property_);
00632         if (value == -1)
00633             throw error_t();
00634         return value;
00635     }
00636     #endif
00637
00638     #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 1, 0)
00639     const char *gets(const char *property_)
00640     {
00641         const char *value = zmq_msg_gets(&msg, property_);
00642         if (value == ZMQ_NULLPTR)
00643             throw error_t();
00644         return value;
00645     }
00646     #endif
00647
00648     #if defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 0)
00649     uint32_t routing_id() const
00650     {
00651         return zmq_msg_routing_id(const_cast<zmq_msg_t *>(&msg));
00652     }
00653
00654     void set_routing_id(uint32_t routing_id)
00655     {
00656         int rc = zmq_msg_set_routing_id(&msg, routing_id);
00657         if (rc != 0)
00658             throw error_t();
00659     }
00660
00661     const char *group() const
00662     {
00663         return zmq_msg_group(const_cast<zmq_msg_t *>(&msg));
00664     }
00665
00666     void set_group(const char *group)
00667     {
00668         int rc = zmq_msg_set_group(&msg, group);
00669         if (rc != 0)
00670             throw error_t();
00671     }
00672     #endif
00673
00674     // interpret message content as a string
00675     std::string to_string() const
00676     {
00677         return std::string(static_cast<const char *>(data()), size());
00678     }
00679     #if CPPZMQ_HAS_STRING_VIEW
00680     // interpret message content as a string
00681     std::string_view to_string_view() const noexcept
00682     {

```

```

00683         return std::string_view(static_cast<const char *>(data()), size());
00684     }
00685 #endif
00686
00693     std::string str() const
00694     {
00695         // Partly mutated from the same method in zmq::multipart_t
00696         std::stringstream os;
00697
00698         const unsigned char *msg_data = this->data<unsigned char>();
00699         unsigned char byte;
00700         size_t size = this->size();
00701         int is_ascii[2] = {0, 0};
00702
00703         os << "zmq::message_t [size " << std::dec << std::setw(3)
00704             << std::setfill('0') << size << "]" << " ";
00705         // Totally arbitrary
00706         if (size >= 1000) {
00707             os << "... too big to print");
00708         } else {
00709             while (size-- > 0) {
00710                 byte = *msg_data++;
00711
00712                 is_ascii[1] = (byte >= 32 && byte < 127);
00713                 if (is_ascii[1] != is_ascii[0])
00714                     os << " "; // Separate text/non text
00715
00716                 if (is_ascii[1]) {
00717                     os << byte;
00718                 } else {
00719                     os << std::hex << std::uppercase << std::setw(2)
00720                         << std::setfill('0') << static_cast<short>(byte);
00721                 }
00722                 is_ascii[0] = is_ascii[1];
00723             }
00724             os << " ";
00725         }
00726         return os.str();
00727     }
00728
00729     void swap(message_t &other) ZMQ_NOTHROW
00730     {
00731         // this assumes zmq::msg_t from libzmq is trivially relocatable
00732         std::swap(msg, other.msg);
00733     }
00734
00735     ZMQ_NODISCARD zmq_msg_t *handle() ZMQ_NOTHROW { return &msg; }
00736     ZMQ_NODISCARD const zmq_msg_t *handle() const ZMQ_NOTHROW { return &msg; }
00737
00738 private:
00739     // The underlying message
00740     zmq_msg_t msg;
00741
00742     // Disable implicit message copying, so that users won't use shared
00743     // messages (less efficient) without being aware of the fact.
00744     message_t(const message_t &) ZMQ_DELETED_FUNCTION;
00745     void operator=(const message_t &) ZMQ_DELETED_FUNCTION;
00746 };
00747
00748 inline void swap(message_t &a, message_t &b) ZMQ_NOTHROW
00749 {
00750     a.swap(b);
00751 }
00752
00753 #ifdef ZMQ_CPP11
00754 enum class ctxopt
00755 {
00756     #ifdef ZMQ_BLOCKY
00757         blocky = ZMQ_BLOCKY,
00758     #endif
00759     #ifdef ZMQ_IO_THREADS
00760         io_threads = ZMQ_IO_THREADS,
00761     #endif
00762     #ifdef ZMQ_THREAD_SCHED_POLICY
00763         thread_sched_policy = ZMQ_THREAD_SCHED_POLICY,
00764     #endif
00765     #ifdef ZMQ_THREAD_PRIORITY
00766         thread_priority = ZMQ_THREAD_PRIORITY,
00767     #endif
00768     #ifdef ZMQ_THREAD_AFFINITY_CPU_ADD
00769         thread_affinity_cpu_add = ZMQ_THREAD_AFFINITY_CPU_ADD,
00770     #endif
00771     #ifdef ZMQ_THREAD_AFFINITY_CPU_REMOVE
00772         thread_affinity_cpu_remove = ZMQ_THREAD_AFFINITY_CPU_REMOVE,
00773     #endif
00774     #ifdef ZMQ_THREAD_NAME_PREFIX
00775         thread_name_prefix = ZMQ_THREAD_NAME_PREFIX,

```

```

00776 #endif
00777 #ifdef ZMQ_MAX_MSGSZ
00778     max_msgsz = ZMQ_MAX_MSGSZ,
00779 #endif
00780 #ifdef ZMQ_ZERO_COPY_RECV
00781     zero_copy_recv = ZMQ_ZERO_COPY_RECV,
00782 #endif
00783 #ifdef ZMQ_MAX_SOCKETS
00784     max_sockets = ZMQ_MAX_SOCKETS,
00785 #endif
00786 #ifdef ZMQ_SOCKET_LIMIT
00787     socket_limit = ZMQ_SOCKET_LIMIT,
00788 #endif
00789 #ifdef ZMQ_IPV6
00790     ipv6 = ZMQ_IPV6,
00791 #endif
00792 #ifdef ZMQ_MSG_T_SIZE
00793     msg_t_size = ZMQ_MSG_T_SIZE
00794 #endif
00795 };
00796 #endif
00797
00798 class context_t
00799 {
00800 public:
00801     context_t()
00802     {
00803         ptr = zmq_ctx_new();
00804         if (ptr == ZMQ_NULLPTR)
00805             throw error_t();
00806     }
00807
00808
00809     explicit context_t(int io_threads_, int max_sockets_ = ZMQ_MAX_SOCKETS_DFLT)
00810     {
00811         ptr = zmq_ctx_new();
00812         if (ptr == ZMQ_NULLPTR)
00813             throw error_t();
00814
00815         int rc = zmq_ctx_set(ptr, ZMQ_IO_THREADS, io_threads_);
00816         ZMQ_ASSERT(rc == 0);
00817
00818         rc = zmq_ctx_set(ptr, ZMQ_MAX_SOCKETS, max_sockets_);
00819         ZMQ_ASSERT(rc == 0);
00820     }
00821
00822 #ifdef ZMQ_HAS_RVALUE_REFS
00823     context_t(context_t &&rhs) ZMQ_NOTHROW : ptr(rhs.ptr) { rhs.ptr = ZMQ_NULLPTR; }
00824     context_t &operator=(context_t &&rhs) ZMQ_NOTHROW
00825     {
00826         close();
00827         std::swap(ptr, rhs.ptr);
00828         return *this;
00829     }
00830 #endif
00831
00832     ~context_t() ZMQ_NOTHROW { close(); }
00833
00834     ZMQ_CPP11_DEPRECATED("from 4.7.0, use set taking zmq::ctxopt instead")
00835     int setctxopt(int option_, int optval_)
00836     {
00837         int rc = zmq_ctx_set(ptr, option_, optval_);
00838         ZMQ_ASSERT(rc == 0);
00839         return rc;
00840     }
00841
00842     ZMQ_CPP11_DEPRECATED("from 4.7.0, use get taking zmq::ctxopt instead")
00843     int getctxopt(int option_) { return zmq_ctx_get(ptr, option_); }
00844
00845 #ifdef ZMQ_CPP11
00846     void set(ctxopt option, int optval)
00847     {
00848         int rc = zmq_ctx_set(ptr, static_cast<int>(option), optval);
00849         if (rc == -1)
00850             throw error_t();
00851     }
00852
00853     ZMQ_NODISCARD int get(ctxopt option)
00854     {
00855         int rc = zmq_ctx_get(ptr, static_cast<int>(option));
00856         // some options have a default value of -1
00857         // which is unfortunate, and may result in errors
00858         // that don't make sense
00859         if (rc == -1)
00860             throw error_t();
00861         return rc;
00862     }

```

```

00863 #endif
00864
00865 // Terminates context (see also shutdown()).
00866 void close() ZMQ_NOTHROW
00867 {
00868     if (ptr == ZMQ_NULLPTR)
00869         return;
00870
00871     int rc;
00872     do {
00873         rc = zmq_ctx_term(ptr);
00874     } while (rc == -1 && errno == EINTR);
00875
00876     ZMQ_ASSERT(rc == 0);
00877     ptr = ZMQ_NULLPTR;
00878 }
00879
00880 // Shutdown context in preparation for termination (close()).
00881 // Causes all blocking socket operations and any further
00882 // socket operations to return with ETERM.
00883 void shutdown() ZMQ_NOTHROW
00884 {
00885     if (ptr == ZMQ_NULLPTR)
00886         return;
00887     int rc = zmq_ctx_shutdown(ptr);
00888     ZMQ_ASSERT(rc == 0);
00889 }
00890
00891 // Be careful with this, it's probably only useful for
00892 // using the C api together with an existing C++ api.
00893 // Normally you should never need to use this.
00894 ZMQ_EXPLICIT operator void *() ZMQ_NOTHROW { return ptr; }
00895
00896 ZMQ_EXPLICIT operator void const *() const ZMQ_NOTHROW { return ptr; }
00897
00898 ZMQ_NODISCARD void *handle() ZMQ_NOTHROW { return ptr; }
00899
00900 ZMQ_DEPRECATED("from 4.7.0, use handle() != nullptr instead")
00901 operator bool() const ZMQ_NOTHROW { return ptr != ZMQ_NULLPTR; }
00902
00903 void swap(context_t &other) ZMQ_NOTHROW { std::swap(ptr, other.ptr); }
00904
00905 private:
00906     void *ptr;
00907
00908     context_t(const context_t &) ZMQ_DELETED_FUNCTION;
00909     void operator=(const context_t &) ZMQ_DELETED_FUNCTION;
00910 };
00911
00912 inline void swap(context_t &a, context_t &b) ZMQ_NOTHROW
00913 {
00914     a.swap(b);
00915 }
00916
00917 #ifdef ZMQ_CPP11
00918
00919 struct recv_buffer_size
00920 {
00921     size_t size; // number of bytes written to buffer
00922     size_t untruncated_size; // untruncated message size in bytes
00923
00924     ZMQ_NODISCARD bool truncated() const noexcept
00925     {
00926         return size != untruncated_size;
00927     }
00928 };
00929
00930 #if CPPZMQ_HAS_OPTIONAL
00931
00932 using send_result_t = std::optional<size_t>;
00933 using recv_result_t = std::optional<size_t>;
00934 using recv_buffer_result_t = std::optional<recv_buffer_size>;
00935
00936 #else
00937
00938 namespace detail
00939 {
00940     // A C++11 type emulating the most basic
00941     // operations of std::optional for trivial types
00942     template<class T> class trivial_optional
00943     {
00944     public:
00945         static_assert(std::is_trivial<T>::value, "T must be trivial");
00946         using value_type = T;
00947
00948         trivial_optional() = default;
00949         trivial_optional(T value) noexcept : _value(value), _has_value(true) {}

```



```

00950
00951     const T *operator->() const noexcept
00952     {
00953         assert(_has_value);
00954         return &_amp;_value;
00955     }
00956     T *operator->() noexcept
00957     {
00958         assert(_has_value);
00959         return &_amp;_value;
00960     }
00961
00962     const T &operator*() const noexcept
00963     {
00964         assert(_has_value);
00965         return _amp;_value;
00966     }
00967     T &operator*() noexcept
00968     {
00969         assert(_has_value);
00970         return _amp;_value;
00971     }
00972
00973     T &value()
00974     {
00975         if (!_has_value)
00976             throw std::exception();
00977         return _amp;_value;
00978     }
00979     const T &value() const
00980     {
00981         if (!_has_value)
00982             throw std::exception();
00983         return _amp;_value;
00984     }
00985
00986     explicit operator bool() const noexcept { return _has_value; }
00987     bool has_value() const noexcept { return _has_value; }
00988
00989 private:
00990     T _value{};
00991     bool _has_value{false};
00992 };
00993 } // namespace detail
00994
00995 using send_result_t = detail::trivial_optional<size_t>;
00996 using rcv_result_t = detail::trivial_optional<size_t>;
00997 using rcv_buffer_result_t = detail::trivial_optional<rcv_buffer_size>;
00998
00999 #endif
01000
01001 namespace detail
01002 {
01003     template<class T> constexpr T enum_bit_or(T a, T b) noexcept
01004     {
01005         static_assert(std::is_enum<T>::value, "must be enum");
01006         using U = typename std::underlying_type<T>::type;
01007         return static_cast<T>(static_cast<U>(a) | static_cast<U>(b));
01008     }
01009     template<class T> constexpr T enum_bit_and(T a, T b) noexcept
01010     {
01011         static_assert(std::is_enum<T>::value, "must be enum");
01012         using U = typename std::underlying_type<T>::type;
01013         return static_cast<T>(static_cast<U>(a) & static_cast<U>(b));
01014     }
01015     template<class T> constexpr T enum_bit_xor(T a, T b) noexcept
01016     {
01017         static_assert(std::is_enum<T>::value, "must be enum");
01018         using U = typename std::underlying_type<T>::type;
01019         return static_cast<T>(static_cast<U>(a) ^ static_cast<U>(b));
01020     }
01021     template<class T> constexpr T enum_bit_not(T a) noexcept
01022     {
01023         static_assert(std::is_enum<T>::value, "must be enum");
01024         using U = typename std::underlying_type<T>::type;
01025         return static_cast<T>(~static_cast<U>(a));
01026     }
01027 } // namespace detail
01028
01029 // partially satisfies named requirement BitmaskType
01030 enum class send_flags : int
01031 {
01032     none = 0,
01033     dontwait = ZMQ_DONTWAIT,
01034     sndmore = ZMQ_SNDMORE
01035 };
01036

```

```

01037 constexpr send_flags operator|(send_flags a, send_flags b) noexcept
01038 {
01039     return detail::enum_bit_or(a, b);
01040 }
01041 constexpr send_flags operator&(send_flags a, send_flags b) noexcept
01042 {
01043     return detail::enum_bit_and(a, b);
01044 }
01045 constexpr send_flags operator^(send_flags a, send_flags b) noexcept
01046 {
01047     return detail::enum_bit_xor(a, b);
01048 }
01049 constexpr send_flags operator~(send_flags a) noexcept
01050 {
01051     return detail::enum_bit_not(a);
01052 }
01053
01054 // partially satisfies named requirement BitmaskType
01055 enum class recv_flags : int
01056 {
01057     none = 0,
01058     dontwait = ZMQ_DONTWAIT
01059 };
01060
01061 constexpr recv_flags operator|(recv_flags a, recv_flags b) noexcept
01062 {
01063     return detail::enum_bit_or(a, b);
01064 }
01065 constexpr recv_flags operator&(recv_flags a, recv_flags b) noexcept
01066 {
01067     return detail::enum_bit_and(a, b);
01068 }
01069 constexpr recv_flags operator^(recv_flags a, recv_flags b) noexcept
01070 {
01071     return detail::enum_bit_xor(a, b);
01072 }
01073 constexpr recv_flags operator~(recv_flags a) noexcept
01074 {
01075     return detail::enum_bit_not(a);
01076 }
01077
01078
01079 // mutable_buffer, const_buffer and buffer are based on
01080 // the Networking TS specification, draft:
01081 // http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4771.pdf
01082
01083 class mutable_buffer
01084 {
01085     public:
01086         constexpr mutable_buffer() noexcept : _data(nullptr), _size(0) {}
01087         constexpr mutable_buffer(void *p, size_t n) noexcept : _data(p), _size(n)
01088         {
01089             #ifdef ZMQ_EXTENDED_CONSTEXPR
01090                 assert(p != nullptr || n == 0);
01091             #endif
01092         }
01093
01094         constexpr void *data() const noexcept { return _data; }
01095         constexpr size_t size() const noexcept { return _size; }
01096         mutable_buffer &operator+=(size_t n) noexcept
01097         {
01098             // (std::min) is a workaround for when a min macro is defined
01099             const auto shift = (std::min)(n, _size);
01100             _data = static_cast<char *>(_data) + shift;
01101             _size -= shift;
01102             return *this;
01103         }
01104
01105     private:
01106         void *_data;
01107         size_t _size;
01108 };
01109
01110 inline mutable_buffer operator+(const mutable_buffer &mb, size_t n) noexcept
01111 {
01112     return mutable_buffer(static_cast<char *>(mb.data()) + (std::min)(n, mb.size()),
01113                           mb.size() - (std::min)(n, mb.size()));
01114 }
01115 inline mutable_buffer operator+(size_t n, const mutable_buffer &mb) noexcept
01116 {
01117     return mb + n;
01118 }
01119
01120 class const_buffer
01121 {
01122     public:
01123         constexpr const_buffer() noexcept : _data(nullptr), _size(0) {}

```

```

01124     constexpr const_buffer(const void *p, size_t n) noexcept : _data(p), _size(n)
01125     {
01126 #ifdef ZMQ_EXTENDED_CONSTEXPR
01127     assert(p != nullptr || n == 0);
01128 #endif
01129     }
01130     constexpr const_buffer(const mutable_buffer &mb) noexcept :
01131     _data(mb.data()), _size(mb.size())
01132     {
01133     }
01134
01135     constexpr const void *data() const noexcept { return _data; }
01136     constexpr size_t size() const noexcept { return _size; }
01137     const_buffer &operator+=(size_t n) noexcept
01138     {
01139         const auto shift = (std::min)(n, _size);
01140         _data = static_cast<const char *>(_data) + shift;
01141         _size -= shift;
01142         return *this;
01143     }
01144
01145 private:
01146     const void *_data;
01147     size_t _size;
01148 };
01149
01150 inline const_buffer operator+(const const_buffer &cb, size_t n) noexcept
01151 {
01152     return const_buffer(static_cast<const char *>(cb.data())
01153         + (std::min)(n, cb.size()),
01154         cb.size() - (std::min)(n, cb.size()));
01155 }
01156 inline const_buffer operator+(size_t n, const const_buffer &cb) noexcept
01157 {
01158     return cb + n;
01159 }
01160
01161 // buffer creation
01162
01163 constexpr mutable_buffer buffer(void *p, size_t n) noexcept
01164 {
01165     return mutable_buffer(p, n);
01166 }
01167 constexpr const_buffer buffer(const void *p, size_t n) noexcept
01168 {
01169     return const_buffer(p, n);
01170 }
01171 constexpr mutable_buffer buffer(const mutable_buffer &mb) noexcept
01172 {
01173     return mb;
01174 }
01175 inline mutable_buffer buffer(const mutable_buffer &mb, size_t n) noexcept
01176 {
01177     return mutable_buffer(mb.data(), (std::min)(mb.size(), n));
01178 }
01179 constexpr const_buffer buffer(const const_buffer &cb) noexcept
01180 {
01181     return cb;
01182 }
01183 inline const_buffer buffer(const const_buffer &cb, size_t n) noexcept
01184 {
01185     return const_buffer(cb.data(), (std::min)(cb.size(), n));
01186 }
01187
01188 namespace detail
01189 {
01190 template<class T> struct is_buffer
01191 {
01192     static constexpr bool value =
01193         std::is_same<T, const_buffer>::value || std::is_same<T, mutable_buffer>::value;
01194 };
01195
01196 template<class T> struct is_pod_like
01197 {
01198     // NOTE: The networking draft N4771 section 16.11 requires
01199     // T in the buffer functions below to be
01200     // trivially copyable OR standard layout.
01201     // Here we decide to be conservative and require both.
01202     static constexpr bool value =
01203         ZMQ_IS_TRIVIALLY_COPYABLE(T) && std::is_standard_layout<T>::value;
01204 };
01205
01206 template<class C> constexpr auto seq_size(const C &c) noexcept -> decltype(c.size())
01207 {
01208     return c.size();
01209 }
01210 template<class T, size_t N>

```

```

01211 constexpr size_t seq_size(const T (& /*array*/)[N]) noexcept
01212 {
01213     return N;
01214 }
01215
01216 template<class Seq>
01217 auto buffer_contiguous_sequence(Seq &&seq) noexcept
01218     -> decltype(buffer(std::addressof(*std::begin(seq)), size_t{}))
01219 {
01220     using T = typename std::remove_cv<
01221         typename std::remove_reference<decltype(*std::begin(seq))>::type>::type;
01222     static_assert(detail::is_pod_like<T>::value, "T must be POD");
01223
01224     const auto size = seq_size(seq);
01225     return buffer(size != 0u ? std::addressof(*std::begin(seq)) : nullptr,
01226         size * sizeof(T));
01227 }
01228 template<class Seq>
01229 auto buffer_contiguous_sequence(Seq &&seq, size_t n_bytes) noexcept
01230     -> decltype(buffer_contiguous_sequence(seq))
01231 {
01232     using T = typename std::remove_cv<
01233         typename std::remove_reference<decltype(*std::begin(seq))>::type>::type;
01234     static_assert(detail::is_pod_like<T>::value, "T must be POD");
01235
01236     const auto size = seq_size(seq);
01237     return buffer(size != 0u ? std::addressof(*std::begin(seq)) : nullptr,
01238         (std::min)(size * sizeof(T), n_bytes));
01239 }
01240
01241 } // namespace detail
01242
01243 // C array
01244 template<class T, size_t N> mutable_buffer buffer(T (&data)[N]) noexcept
01245 {
01246     return detail::buffer_contiguous_sequence(data);
01247 }
01248 template<class T, size_t N>
01249 mutable_buffer buffer(T (&data)[N], size_t n_bytes) noexcept
01250 {
01251     return detail::buffer_contiguous_sequence(data, n_bytes);
01252 }
01253 template<class T, size_t N> const_buffer buffer(const T (&data)[N]) noexcept
01254 {
01255     return detail::buffer_contiguous_sequence(data);
01256 }
01257 template<class T, size_t N>
01258 const_buffer buffer(const T (&data)[N], size_t n_bytes) noexcept
01259 {
01260     return detail::buffer_contiguous_sequence(data, n_bytes);
01261 }
01262 // std::array
01263 template<class T, size_t N> mutable_buffer buffer(std::array<T, N> &data) noexcept
01264 {
01265     return detail::buffer_contiguous_sequence(data);
01266 }
01267 template<class T, size_t N>
01268 mutable_buffer buffer(std::array<T, N> &data, size_t n_bytes) noexcept
01269 {
01270     return detail::buffer_contiguous_sequence(data, n_bytes);
01271 }
01272 template<class T, size_t N>
01273 const_buffer buffer(std::array<const T, N> &data) noexcept
01274 {
01275     return detail::buffer_contiguous_sequence(data);
01276 }
01277 template<class T, size_t N>
01278 const_buffer buffer(std::array<const T, N> &data, size_t n_bytes) noexcept
01279 {
01280     return detail::buffer_contiguous_sequence(data, n_bytes);
01281 }
01282 template<class T, size_t N>
01283 const_buffer buffer(const std::array<T, N> &data) noexcept
01284 {
01285     return detail::buffer_contiguous_sequence(data);
01286 }
01287 template<class T, size_t N>
01288 const_buffer buffer(const std::array<T, N> &data, size_t n_bytes) noexcept
01289 {
01290     return detail::buffer_contiguous_sequence(data, n_bytes);
01291 }
01292 // std::vector
01293 template<class T, class Allocator>
01294 mutable_buffer buffer(std::vector<T, Allocator> &data) noexcept
01295 {
01296     return detail::buffer_contiguous_sequence(data);
01297 }

```

```

01298 template<class T, class Allocator>
01299 mutable_buffer buffer(std::vector<T, Allocator> &data, size_t n_bytes) noexcept
01300 {
01301     return detail::buffer_contiguous_sequence(data, n_bytes);
01302 }
01303 template<class T, class Allocator>
01304 const_buffer buffer(const std::vector<T, Allocator> &data) noexcept
01305 {
01306     return detail::buffer_contiguous_sequence(data);
01307 }
01308 template<class T, class Allocator>
01309 const_buffer buffer(const std::vector<T, Allocator> &data, size_t n_bytes) noexcept
01310 {
01311     return detail::buffer_contiguous_sequence(data, n_bytes);
01312 }
01313 // std::basic_string
01314 template<class T, class Traits, class Allocator>
01315 mutable_buffer buffer(std::basic_string<T, Traits, Allocator> &data) noexcept
01316 {
01317     return detail::buffer_contiguous_sequence(data);
01318 }
01319 template<class T, class Traits, class Allocator>
01320 mutable_buffer buffer(std::basic_string<T, Traits, Allocator> &data,
01321     size_t n_bytes) noexcept
01322 {
01323     return detail::buffer_contiguous_sequence(data, n_bytes);
01324 }
01325 template<class T, class Traits, class Allocator>
01326 const_buffer buffer(const std::basic_string<T, Traits, Allocator> &data) noexcept
01327 {
01328     return detail::buffer_contiguous_sequence(data);
01329 }
01330 template<class T, class Traits, class Allocator>
01331 const_buffer buffer(const std::basic_string<T, Traits, Allocator> &data,
01332     size_t n_bytes) noexcept
01333 {
01334     return detail::buffer_contiguous_sequence(data, n_bytes);
01335 }
01336
01337 #if CPPZMQ_HAS_STRING_VIEW
01338 // std::basic_string_view
01339 template<class T, class Traits>
01340 const_buffer buffer(std::basic_string_view<T, Traits> data) noexcept
01341 {
01342     return detail::buffer_contiguous_sequence(data);
01343 }
01344 template<class T, class Traits>
01345 const_buffer buffer(std::basic_string_view<T, Traits> data, size_t n_bytes) noexcept
01346 {
01347     return detail::buffer_contiguous_sequence(data, n_bytes);
01348 }
01349 #endif
01350
01351 // Buffer for a string literal (null terminated)
01352 // where the buffer size excludes the terminating character.
01353 // Equivalent to zmq::buffer(std::string_view("...")).
01354 template<class Char, size_t N>
01355 constexpr const_buffer str_buffer(const Char (&data)[N]) noexcept
01356 {
01357     static_assert(detail::is_pod_like<Char>::value, "Char must be POD");
01358 #ifdef ZMQ_EXTENDED_CONSTEXPR
01359     assert(data[N - 1] == Char{0});
01360 #endif
01361     return const_buffer(static_cast<const Char *>(data), (N - 1) * sizeof(Char));
01362 }
01363
01364 namespace literals
01365 {
01366     constexpr const_buffer operator"" _zbuf(const char *str, size_t len) noexcept
01367     {
01368         return const_buffer(str, len * sizeof(char));
01369     }
01370     constexpr const_buffer operator"" _zbuf(const wchar_t *str, size_t len) noexcept
01371     {
01372         return const_buffer(str, len * sizeof(wchar_t));
01373     }
01374     constexpr const_buffer operator"" _zbuf(const char16_t *str, size_t len) noexcept
01375     {
01376         return const_buffer(str, len * sizeof(char16_t));
01377     }
01378     constexpr const_buffer operator"" _zbuf(const char32_t *str, size_t len) noexcept
01379     {
01380         return const_buffer(str, len * sizeof(char32_t));
01381     }
01382 }
01383
01384 #ifdef ZMQ_CPP11

```

```

01385 enum class socket_type : int
01386 {
01387     req = ZMQ_REQ,
01388     rep = ZMQ_REP,
01389     dealer = ZMQ_DEALER,
01390     router = ZMQ_ROUTER,
01391     pub = ZMQ_PUB,
01392     sub = ZMQ_SUB,
01393     xpub = ZMQ_XPUB,
01394     xsub = ZMQ_XSUB,
01395     push = ZMQ_PUSH,
01396     pull = ZMQ_PULL,
01397 #if defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 0)
01398     server = ZMQ_SERVER,
01399     client = ZMQ_CLIENT,
01400     radio = ZMQ_RADIO,
01401     dish = ZMQ_DISH,
01402     gather = ZMQ_GATHER,
01403     scatter = ZMQ_SCATTER,
01404     dgram = ZMQ_DGRAM,
01405 #endif
01406 #if defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 3, 3)
01407     peer = ZMQ_PEER,
01408     channel = ZMQ_CHANNEL,
01409 #endif
01410 #if ZMQ_VERSION_MAJOR >= 4
01411     stream = ZMQ_STREAM,
01412 #endif
01413     pair = ZMQ_PAIR
01414 };
01415 #endif
01416 namespace sockopt
01417 {
01418     // There are two types of options,
01419     // integral type with known compiler time size (int, bool, int64_t, uint64_t)
01420     // and arrays with dynamic size (strings, binary data).
01421     // BoolUnit: if true accepts values of type bool (but passed as T into libzmq)
01422     template<int Opt, class T, bool BoolUnit = false> struct integral_option
01423     {
01424     };
01425     // NullTerm:
01426     // 0: binary data
01427     // 1: null-terminated string ('getsockopt' size includes null)
01428     // 2: binary (size 32) or Z85 encoder string of size 41 (null included)
01429     template<int Opt, int NullTerm = 1> struct array_option
01430     {
01431     };
01432 #define ZMQ_DEFINE_INTEGRAL_OPT(OPT, NAME, TYPE) \
01433     using NAME##_t = integral_option<OPT, TYPE, false>; \
01434     ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {} \
01435 #define ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(OPT, NAME, TYPE) \
01436     using NAME##_t = integral_option<OPT, TYPE, true>; \
01437     ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {} \
01438 #define ZMQ_DEFINE_ARRAY_OPT(OPT, NAME) \
01439     using NAME##_t = array_option<OPT>; \
01440     ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {} \
01441 #define ZMQ_DEFINE_ARRAY_OPT_BINARY(OPT, NAME) \
01442     using NAME##_t = array_option<OPT, 0>; \
01443     ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {} \
01444 #define ZMQ_DEFINE_ARRAY_OPT_BIN_OR_Z85(OPT, NAME) \
01445     using NAME##_t = array_option<OPT, 2>; \
01446     ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {} \
01447 // deprecated, use zmq::fd_t
01448 using cppzmq_fd_t = ::zmq::fd_t;
01449 #ifndef ZMQ_AFFINITY
01450 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_AFFINITY, affinity, uint64_t);
01451 #endif
01452 #ifndef ZMQ_BACKLOG
01453 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_BACKLOG, backlog, int);
01454 #endif
01455 #ifndef ZMQ_BINDTODEVICE
01456 ZMQ_DEFINE_ARRAY_OPT_BINARY(ZMQ_BINDTODEVICE, bindtodevice);
01457 #endif
01458 #ifndef ZMQ_CONFLATE
01459 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_CONFLATE, conflate, int);
01460 #endif
01461 #ifndef ZMQ_CONNECT_ROUTING_ID
01462 ZMQ_DEFINE_ARRAY_OPT(ZMQ_CONNECT_ROUTING_ID, connect_routing_id);
01463 #endif
01464 #ifndef ZMQ_CONNECT_TIMEOUT
01465 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_CONNECT_TIMEOUT, connect_timeout, int);

```

```
01472 #endif
01473 #ifdef ZMQ_CURVE_PUBLICKEY
01474 ZMQ_DEFINE_ARRAY_OPT_BIN_OR_Z85(ZMQ_CURVE_PUBLICKEY, curve_publickey);
01475 #endif
01476 #ifdef ZMQ_CURVE_SECRETKEY
01477 ZMQ_DEFINE_ARRAY_OPT_BIN_OR_Z85(ZMQ_CURVE_SECRETKEY, curve_secretkey);
01478 #endif
01479 #ifdef ZMQ_CURVE_SERVER
01480 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_CURVE_SERVER, curve_server, int);
01481 #endif
01482 #ifdef ZMQ_CURVE_SERVERKEY
01483 ZMQ_DEFINE_ARRAY_OPT_BIN_OR_Z85(ZMQ_CURVE_SERVERKEY, curve_serverkey);
01484 #endif
01485 #ifdef ZMQ_DISCONNECT_MSG
01486 ZMQ_DEFINE_ARRAY_OPT_BINARY(ZMQ_DISCONNECT_MSG, disconnect_msg);
01487 #endif
01488 #ifdef ZMQ_EVENTS
01489 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_EVENTS, events, int);
01490 #endif
01491 #ifdef ZMQ_FD
01492 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_FD, fd, ::zmq::fd_t);
01493 #endif
01494 #ifdef ZMQ_GSSAPI_PLAINTEXT
01495 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_GSSAPI_PLAINTEXT, gssapi_plaintext, int);
01496 #endif
01497 #ifdef ZMQ_GSSAPI_SERVER
01498 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_GSSAPI_SERVER, gssapi_server, int);
01499 #endif
01500 #ifdef ZMQ_GSSAPI_SERVICE_PRINCIPAL
01501 ZMQ_DEFINE_ARRAY_OPT(ZMQ_GSSAPI_SERVICE_PRINCIPAL, gssapi_service_principal);
01502 #endif
01503 #ifdef ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAMETYPE
01504 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAMETYPE,
01505     gssapi_service_principal_nametype,
01506     int);
01507 #endif
01508 #ifdef ZMQ_GSSAPI_PRINCIPAL
01509 ZMQ_DEFINE_ARRAY_OPT(ZMQ_GSSAPI_PRINCIPAL, gssapi_principal);
01510 #endif
01511 #ifdef ZMQ_GSSAPI_PRINCIPAL_NAMETYPE
01512 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_GSSAPI_PRINCIPAL_NAMETYPE,
01513     gssapi_principal_nametype,
01514     int);
01515 #endif
01516 #ifdef ZMQ_HANDSHAKE_IVL
01517 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_HANDSHAKE_IVL, handshake_ivl, int);
01518 #endif
01519 #ifdef ZMQ_HEARTBEAT_IVL
01520 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_HEARTBEAT_IVL, heartbeat_ivl, int);
01521 #endif
01522 #ifdef ZMQ_HEARTBEAT_TIMEOUT
01523 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_HEARTBEAT_TIMEOUT, heartbeat_timeout, int);
01524 #endif
01525 #ifdef ZMQ_HEARTBEAT_TTL
01526 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_HEARTBEAT_TTL, heartbeat_ttl, int);
01527 #endif
01528 #ifdef ZMQ_HELLO_MSG
01529 ZMQ_DEFINE_ARRAY_OPT_BINARY(ZMQ_HELLO_MSG, hello_msg);
01530 #endif
01531 #ifdef ZMQ_IMMEDIATE
01532 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_IMMEDIATE, immediate, int);
01533 #endif
01534 #ifdef ZMQ_INVERT_MATCHING
01535 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_INVERT_MATCHING, invert_matching, int);
01536 #endif
01537 #ifdef ZMQ_IPV6
01538 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_IPV6, ipv6, int);
01539 #endif
01540 #ifdef ZMQ_LAST_ENDPOINT
01541 ZMQ_DEFINE_ARRAY_OPT(ZMQ_LAST_ENDPOINT, last_endpoint);
01542 #endif
01543 #ifdef ZMQ_LINGER
01544 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_LINGER, linger, int);
01545 #endif
01546 #ifdef ZMQ_MAXMSGSIZE
01547 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_MAXMSGSIZE, maxmsgsize, int64_t);
01548 #endif
01549 #ifdef ZMQ_MECHANISM
01550 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_MECHANISM, mechanism, int);
01551 #endif
01552 #ifdef ZMQ_METADATA
01553 ZMQ_DEFINE_ARRAY_OPT(ZMQ_METADATA, metadata);
01554 #endif
01555 #ifdef ZMQ_MULTICAST_HOPS
01556 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_MULTICAST_HOPS, multicast_hops, int);
01557 #endif
01558 #ifdef ZMQ_MULTICAST_LOOP
```

```
01559 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_MULTICAST_LOOP, multicast_loop, int);
01560 #endif
01561 #ifdef ZMQ_MULTICAST_MAXTPDU
01562 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_MULTICAST_MAXTPDU, multicast_maxtpdu, int);
01563 #endif
01564 #ifdef ZMQ_ONLY_FIRST_SUBSCRIBE
01565 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_ONLY_FIRST_SUBSCRIBE, only_first_subscribe, int);
01566 #endif
01567 #ifdef ZMQ_PLAIN_SERVER
01568 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_PLAIN_SERVER, plain_server, int);
01569 #endif
01570 #ifdef ZMQ_PLAIN_PASSWORD
01571 ZMQ_DEFINE_ARRAY_OPT(ZMQ_PLAIN_PASSWORD, plain_password);
01572 #endif
01573 #ifdef ZMQ_PLAIN_USERNAME
01574 ZMQ_DEFINE_ARRAY_OPT(ZMQ_PLAIN_USERNAME, plain_username);
01575 #endif
01576 #ifdef ZMQ_PRIORITY
01577 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_PRIORITY, priority, int);
01578 #endif
01579 #ifdef ZMQ_USE_FD
01580 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_USE_FD, use_fd, int);
01581 #endif
01582 #ifdef ZMQ_PROBE_ROUTER
01583 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_PROBE_ROUTER, probe_router, int);
01584 #endif
01585 #ifdef ZMQ_RATE
01586 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RATE, rate, int);
01587 #endif
01588 #ifdef ZMQ_RCVBUF
01589 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RCVBUF, rcvbuf, int);
01590 #endif
01591 #ifdef ZMQ_RCVHWM
01592 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RCVHWM, rcvhwm, int);
01593 #endif
01594 #ifdef ZMQ_RCVMORE
01595 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_RCVMORE, rcvmore, int);
01596 #endif
01597 #ifdef ZMQ_RCVTIMEO
01598 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RCVTIMEO, rcvtimeo, int);
01599 #endif
01600 #ifdef ZMQ_RECONNECT_IVL
01601 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RECONNECT_IVL, reconnect_ivl, int);
01602 #endif
01603 #ifdef ZMQ_RECONNECT_IVL_MAX
01604 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RECONNECT_IVL_MAX, reconnect_ivl_max, int);
01605 #endif
01606 #ifdef ZMQ_RECONNECT_STOP
01607 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RECONNECT_STOP, reconnect_stop, int);
01608 #endif
01609 #ifdef ZMQ_RECOVERY_IVL
01610 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RECOVERY_IVL, recovery_ivl, int);
01611 #endif
01612 #ifdef ZMQ_REQ_CORRELATE
01613 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_REQ_CORRELATE, req_correlate, int);
01614 #endif
01615 #ifdef ZMQ_REQ_RELAXED
01616 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_REQ_RELAXED, req_relaxed, int);
01617 #endif
01618 #ifdef ZMQ_ROUTER_HANDOVER
01619 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_ROUTER_HANDOVER, router_handover, int);
01620 #endif
01621 #ifdef ZMQ_ROUTER_MANDATORY
01622 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_ROUTER_MANDATORY, router_mandatory, int);
01623 #endif
01624 #ifdef ZMQ_ROUTER_NOTIFY
01625 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_ROUTER_NOTIFY, router_notify, int);
01626 #endif
01627 #ifdef ZMQ_ROUTING_ID
01628 ZMQ_DEFINE_ARRAY_OPT_BINARY(ZMQ_ROUTING_ID, routing_id);
01629 #endif
01630 #ifdef ZMQ_SNDBUF
01631 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_SNDBUF, sndbuf, int);
01632 #endif
01633 #ifdef ZMQ_SNDHWM
01634 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_SNDHWM, sndhwm, int);
01635 #endif
01636 #ifdef ZMQ_SNDTIMEO
01637 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_SNDTIMEO, sndtimeo, int);
01638 #endif
01639 #ifdef ZMQ SOCKS_PASSWORD
01640 ZMQ_DEFINE_ARRAY_OPT(ZMQ SOCKS_PASSWORD, socks_password);
01641 #endif
01642 #ifdef ZMQ SOCKS_PROXY
01643 ZMQ_DEFINE_ARRAY_OPT(ZMQ SOCKS_PROXY, socks_proxy);
01644 #endif
01645 #ifdef ZMQ SOCKS_USERNAME
```



```

01646 ZMQ_DEFINE_ARRAY_OPT(ZMQ SOCKS_USERNAME, socks_username);
01647 #endif
01648 #ifdef ZMQ_STREAM_NOTIFY
01649 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_STREAM_NOTIFY, stream_notify, int);
01650 #endif
01651 #ifdef ZMQ_SUBSCRIBE
01652 ZMQ_DEFINE_ARRAY_OPT(ZMQ_SUBSCRIBE, subscribe);
01653 #endif
01654 #ifdef ZMQ_TCP_KEEPAIVE
01655 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_KEEPAIVE, tcp_keepalive, int);
01656 #endif
01657 #ifdef ZMQ_TCP_KEEPAIVE_CNT
01658 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_KEEPAIVE_CNT, tcp_keepalive_cnt, int);
01659 #endif
01660 #ifdef ZMQ_TCP_KEEPAIVE_IDLE
01661 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_KEEPAIVE_IDLE, tcp_keepalive_idle, int);
01662 #endif
01663 #ifdef ZMQ_TCP_KEEPAIVE_INTVL
01664 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_KEEPAIVE_INTVL, tcp_keepalive_intvl, int);
01665 #endif
01666 #ifdef ZMQ_TCP_MAXRT
01667 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_MAXRT, tcp_maxrt, int);
01668 #endif
01669 #ifdef ZMQ_THREAD_SAFE
01670 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_THREAD_SAFE, thread_safe, int);
01671 #endif
01672 #ifdef ZMQ_TOS
01673 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TOS, tos, int);
01674 #endif
01675 #ifdef ZMQ_TYPE
01676 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TYPE, type, int);
01677 #ifdef ZMQ_CPP11
01678 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TYPE, socket_type, socket_type);
01679 #endif // ZMQ_CPP11
01680 #endif // ZMQ_TYPE
01681 #ifdef ZMQ_UNSUBSCRIBE
01682 ZMQ_DEFINE_ARRAY_OPT(ZMQ_UNSUBSCRIBE, unsubscribe);
01683 #endif
01684 #ifdef ZMQ_VMCI_BUFFER_SIZE
01685 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_VMCI_BUFFER_SIZE, vmci_buffer_size, uint64_t);
01686 #endif
01687 #ifdef ZMQ_VMCI_BUFFER_MIN_SIZE
01688 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_VMCI_BUFFER_MIN_SIZE, vmci_buffer_min_size, uint64_t);
01689 #endif
01690 #ifdef ZMQ_VMCI_BUFFER_MAX_SIZE
01691 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_VMCI_BUFFER_MAX_SIZE, vmci_buffer_max_size, uint64_t);
01692 #endif
01693 #ifdef ZMQ_VMCI_CONNECT_TIMEOUT
01694 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_VMCI_CONNECT_TIMEOUT, vmci_connect_timeout, int);
01695 #endif
01696 #ifdef ZMQ_XPUB_VERBOSE
01697 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_VERBOSE, xpub_verbose, int);
01698 #endif
01699 #ifdef ZMQ_XPUB_VERBOSE
01700 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_VERBOSE, xpub_verbose, int);
01701 #endif
01702 #ifdef ZMQ_XPUB_MANUAL
01703 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_MANUAL, xpub_manual, int);
01704 #endif
01705 #ifdef ZMQ_XPUB_MANUAL_LAST_VALUE
01706 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_MANUAL_LAST_VALUE, xpub_manual_last_value, int);
01707 #endif
01708 #ifdef ZMQ_XPUB_NODROP
01709 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_NODROP, xpub_nodrop, int);
01710 #endif
01711 #ifdef ZMQ_XPUB_WELCOME_MSG
01712 ZMQ_DEFINE_ARRAY_OPT(ZMQ_XPUB_WELCOME_MSG, xpub_welcome_msg);
01713 #endif
01714 #ifdef ZMQ_ZAP_ENFORCE_DOMAIN
01715 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_ZAP_ENFORCE_DOMAIN, zap_enforce_domain, int);
01716 #endif
01717 #ifdef ZMQ_ZAP_DOMAIN
01718 ZMQ_DEFINE_ARRAY_OPT(ZMQ_ZAP_DOMAIN, zap_domain);
01719 #endif
01720
01721 } // namespace sockopt
01722 #endif // ZMQ_CPP11
01723
01724 namespace detail
01725 {
01726     class socket_base
01727     {
01728     public:
01729         socket_base() ZMQ_NOTHROW : _handle(ZMQ_NULLPTR) {}
01730         ZMQ_EXPLICIT socket_base(void *handle) ZMQ_NOTHROW : _handle(handle) {}
01731     }
01732 }

```

```

01733     template<typename T>
01734     ZMQ_CPP11_DEPRECATED("from 4.7.0, use `set` taking option from zmq::socketopt")
01735     void setsockopt(int option_, T const &optval)
01736     {
01737         setsockopt(option_, &optval, sizeof(T));
01738     }
01739
01740     ZMQ_CPP11_DEPRECATED("from 4.7.0, use `set` taking option from zmq::socketopt")
01741     void setsockopt(int option_, const void *optval_, size_t optvallen_)
01742     {
01743         int rc = zmq_setsockopt(_handle, option_, optval_, optvallen_);
01744         if (rc != 0)
01745             throw error_t();
01746     }
01747
01748     ZMQ_CPP11_DEPRECATED("from 4.7.0, use `get` taking option from zmq::socketopt")
01749     void getsockopt(int option_, void *optval_, size_t *optvallen_) const
01750     {
01751         int rc = zmq_getsockopt(_handle, option_, optval_, optvallen_);
01752         if (rc != 0)
01753             throw error_t();
01754     }
01755
01756     template<typename T>
01757     ZMQ_CPP11_DEPRECATED("from 4.7.0, use `get` taking option from zmq::socketopt")
01758     T getsockopt(int option_) const
01759     {
01760         T optval;
01761         size_t optlen = sizeof(T);
01762         getsockopt(option_, &optval, &optlen);
01763         return optval;
01764     }
01765
01766 #ifndef ZMQ_CPP11
01767     // Set integral socket option, e.g.
01768     // `socket.set(zmq::socketopt::linger, 0)`
01769     template<int Opt, class T, bool BoolUnit>
01770     void set(socketopt::integral_option<Opt, T, BoolUnit>, const T &val)
01771     {
01772         static_assert(std::is_integral<T>::value, "T must be integral");
01773         set_option(Opt, &val, sizeof val);
01774     }
01775
01776     // Set integral socket option from boolean, e.g.
01777     // `socket.set(zmq::socketopt::immediate, false)`
01778     template<int Opt, class T>
01779     void set(socketopt::integral_option<Opt, T, true>, bool val)
01780     {
01781         static_assert(std::is_integral<T>::value, "T must be integral");
01782         T rep_val = val;
01783         set_option(Opt, &rep_val, sizeof rep_val);
01784     }
01785
01786     // Set array socket option, e.g.
01787     // `socket.set(zmq::socketopt::plain_username, "fool23")`
01788     template<int Opt, int NullTerm>
01789     void set(socketopt::array_option<Opt, NullTerm>, const char *buf)
01790     {
01791         set_option(Opt, buf, std::strlen(buf));
01792     }
01793
01794     // Set array socket option, e.g.
01795     // `socket.set(zmq::socketopt::routing_id, zmq::buffer(id))`
01796     template<int Opt, int NullTerm>
01797     void set(socketopt::array_option<Opt, NullTerm>, const_buffer buf)
01798     {
01799         set_option(Opt, buf.data(), buf.size());
01800     }
01801
01802     // Set array socket option, e.g.
01803     // `socket.set(zmq::socketopt::routing_id, id_str)`
01804     template<int Opt, int NullTerm>
01805     void set(socketopt::array_option<Opt, NullTerm>, const std::string &buf)
01806     {
01807         set_option(Opt, buf.data(), buf.size());
01808     }
01809
01810 #if CPPZMQ_HAS_STRING_VIEW
01811     // Set array socket option, e.g.
01812     // `socket.set(zmq::socketopt::routing_id, id_str)`
01813     template<int Opt, int NullTerm>
01814     void set(socketopt::array_option<Opt, NullTerm>, std::string_view buf)
01815     {
01816         set_option(Opt, buf.data(), buf.size());
01817     }
01818 #endif
01819

```

```

01820 // Get scalar socket option, e.g.
01821 // `auto opt = socket.get(zmq::sockopt::linger)`
01822 template<int Opt, class T, bool BoolUnit>
01823 ZMQ_NODISCARD T get(sockopt::integral_option<Opt, T, BoolUnit>) const
01824 {
01825     static_assert(std::is_scalar<T>::value, "T must be scalar");
01826     T val;
01827     size_t size = sizeof val;
01828     get_option(Opt, &val, &size);
01829     assert(size == sizeof val);
01830     return val;
01831 }
01832
01833 // Get array socket option, writes to buf, returns option size in bytes, e.g.
01834 // `size_t optsize = socket.get(zmq::sockopt::routing_id, zmq::buffer(id))`
01835 template<int Opt, int NullTerm>
01836 ZMQ_NODISCARD size_t get(sockopt::array_option<Opt, NullTerm>,
01837     mutable_buffer buf) const
01838 {
01839     size_t size = buf.size();
01840     get_option(Opt, buf.data(), &size);
01841     return size;
01842 }
01843
01844 // Get array socket option as string (initializes the string buffer size to init_size) e.g.
01845 // `auto s = socket.get(zmq::sockopt::routing_id)`
01846 // Note: removes the null character from null-terminated string options,
01847 // i.e. the string size excludes the null character.
01848 template<int Opt, int NullTerm>
01849 ZMQ_NODISCARD std::string get(sockopt::array_option<Opt, NullTerm>,
01850     size_t init_size = 1024) const
01851 {
01852     if ZMQ_CONSTEXPR_IF (NullTerm == 2) {
01853         if (init_size == 1024) {
01854             init_size = 41; // get as Z85 string
01855         }
01856     }
01857     std::string str(init_size, '\0');
01858     size_t size = get(sockopt::array_option<Opt>{}, buffer(str));
01859     if ZMQ_CONSTEXPR_IF (NullTerm == 1) {
01860         if (size > 0) {
01861             assert(str[size - 1] == '\0');
01862             --size;
01863         }
01864     } else if ZMQ_CONSTEXPR_IF (NullTerm == 2) {
01865         assert(size == 32 || size == 41);
01866         if (size == 41) {
01867             assert(str[size - 1] == '\0');
01868             --size;
01869         }
01870     }
01871     str.resize(size);
01872     return str;
01873 }
01874 #endif
01875
01876 void bind(std::string const &addr) { bind(addr.c_str()); }
01877
01878 void bind(const char *addr_)
01879 {
01880     int rc = zmq_bind(_handle, addr_);
01881     if (rc != 0)
01882         throw error_t();
01883 }
01884
01885 void unbind(std::string const &addr) { unbind(addr.c_str()); }
01886
01887 void unbind(const char *addr_)
01888 {
01889     int rc = zmq_unbind(_handle, addr_);
01890     if (rc != 0)
01891         throw error_t();
01892 }
01893
01894 void connect(std::string const &addr) { connect(addr.c_str()); }
01895
01896 void connect(const char *addr_)
01897 {
01898     int rc = zmq_connect(_handle, addr_);
01899     if (rc != 0)
01900         throw error_t();
01901 }
01902
01903 void disconnect(std::string const &addr) { disconnect(addr.c_str()); }
01904
01905 void disconnect(const char *addr_)
01906 {

```

```

01907         int rc = zmq_disconnect(_handle, addr_);
01908         if (rc != 0)
01909             throw error_t();
01910     }
01911
01912     ZMQ_DEPRECATED("from 4.7.1, use handle() != nullptr or operator bool")
01913     bool connected() const ZMQ_NOTHROW { return (_handle != ZMQ_NULLPTR); }
01914
01915     ZMQ_CPP11_DEPRECATED("from 4.3.1, use send taking a const_buffer and send_flags")
01916     size_t send(const void *buf_, size_t len_, int flags_ = 0)
01917     {
01918         int nbytes = zmq_send(_handle, buf_, len_, flags_);
01919         if (nbytes >= 0)
01920             return static_cast<size_t>(nbytes);
01921         if (zmq_errno() == EAGAIN)
01922             return 0;
01923         throw error_t();
01924     }
01925
01926     ZMQ_CPP11_DEPRECATED("from 4.3.1, use send taking message_t and send_flags")
01927     bool send(message_t &msg_,
01928              int flags_ = 0) // default until removed
01929     {
01930         int nbytes = zmq_msg_send(msg_.handle(), _handle, flags_);
01931         if (nbytes >= 0)
01932             return true;
01933         if (zmq_errno() == EAGAIN)
01934             return false;
01935         throw error_t();
01936     }
01937
01938     template<typename T>
01939     ZMQ_CPP11_DEPRECATED(
01940         "from 4.4.1, use send taking message_t or buffer (for contiguous "
01941         "ranges), and send_flags")
01942     bool send(T first, T last, int flags_ = 0)
01943     {
01944         zmq::message_t msg(first, last);
01945         int nbytes = zmq_msg_send(msg.handle(), _handle, flags_);
01946         if (nbytes >= 0)
01947             return true;
01948         if (zmq_errno() == EAGAIN)
01949             return false;
01950         throw error_t();
01951     }
01952
01953     #ifndef ZMQ_HAS_RVALUE_REFS
01954     ZMQ_CPP11_DEPRECATED("from 4.3.1, use send taking message_t and send_flags")
01955     bool send(message_t &&msg_,
01956              int flags_ = 0) // default until removed
01957     {
01958     #ifdef ZMQ_CPP11
01959         return send(msg_, static_cast<send_flags>(flags_)).has_value();
01960     #else
01961         return send(msg_, flags_);
01962     #endif
01963     }
01964     #endif
01965
01966     #ifdef ZMQ_CPP11
01967     send_result_t send(const_buffer buf, send_flags flags = send_flags::none)
01968     {
01969         const int nbytes =
01970             zmq_send(_handle, buf.data(), buf.size(), static_cast<int>(flags));
01971         if (nbytes >= 0)
01972             return static_cast<size_t>(nbytes);
01973         if (zmq_errno() == EAGAIN)
01974             return {};
01975         throw error_t();
01976     }
01977
01978     send_result_t send(message_t &msg, send_flags flags)
01979     {
01980         int nbytes = zmq_msg_send(msg.handle(), _handle, static_cast<int>(flags));
01981         if (nbytes >= 0)
01982             return static_cast<size_t>(nbytes);
01983         if (zmq_errno() == EAGAIN)
01984             return {};
01985         throw error_t();
01986     }
01987
01988     send_result_t send(message_t &&msg, send_flags flags)
01989     {
01990         return send(msg, flags);
01991     }
01992     #endif
01993

```

```

01994     ZMQ_CPP11_DEPRECATED(
01995         "from 4.3.1, use recv taking a mutable_buffer and recv_flags")
01996     size_t recv(void *buf_, size_t len_, int flags_ = 0)
01997     {
01998         int nbytes = zmq_recv(_handle, buf_, len_, flags_);
01999         if (nbytes >= 0)
02000             return static_cast<size_t>(nbytes);
02001         if (zmq_errno() == EAGAIN)
02002             return 0;
02003         throw error_t();
02004     }
02005
02006     ZMQ_CPP11_DEPRECATED(
02007         "from 4.3.1, use recv taking a reference to message_t and recv_flags")
02008     bool recv(message_t *msg_, int flags_ = 0)
02009     {
02010         int nbytes = zmq_msg_recv(msg_->handle(), _handle, flags_);
02011         if (nbytes >= 0)
02012             return true;
02013         if (zmq_errno() == EAGAIN)
02014             return false;
02015         throw error_t();
02016     }
02017
02018 #ifdef ZMQ_CPP11
02019     ZMQ_NODISCARD
02020     recv_buffer_result_t recv(mutable_buffer buf,
02021                               recv_flags flags = recv_flags::none)
02022     {
02023         const int nbytes =
02024             zmq_recv(_handle, buf.data(), buf.size(), static_cast<int>(flags));
02025         if (nbytes >= 0) {
02026             return recv_buffer_size{
02027                 (std::min)(static_cast<size_t>(nbytes), buf.size()),
02028                 static_cast<size_t>(nbytes)};
02029         }
02030         if (zmq_errno() == EAGAIN)
02031             return {};
02032         throw error_t();
02033     }
02034
02035     ZMQ_NODISCARD
02036     recv_result_t recv(message_t &msg, recv_flags flags = recv_flags::none)
02037     {
02038         const int nbytes =
02039             zmq_msg_recv(msg.handle(), _handle, static_cast<int>(flags));
02040         if (nbytes >= 0) {
02041             assert(msg.size() == static_cast<size_t>(nbytes));
02042             return static_cast<size_t>(nbytes);
02043         }
02044         if (zmq_errno() == EAGAIN)
02045             return {};
02046         throw error_t();
02047     }
02048 #endif
02049
02050 #if defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 0)
02051     void join(const char *group)
02052     {
02053         int rc = zmq_join(_handle, group);
02054         if (rc != 0)
02055             throw error_t();
02056     }
02057
02058     void leave(const char *group)
02059     {
02060         int rc = zmq_leave(_handle, group);
02061         if (rc != 0)
02062             throw error_t();
02063     }
02064 #endif
02065
02066     ZMQ_NODISCARD void *handle() ZMQ_NOTHROW { return _handle; }
02067     ZMQ_NODISCARD const void *handle() const ZMQ_NOTHROW { return _handle; }
02068
02069     ZMQ_EXPLICIT operator bool() const ZMQ_NOTHROW { return _handle != ZMQ_NULLPTR; }
02070     // note: non-const operator bool can be removed once
02071     // operator void* is removed from socket_t
02072     ZMQ_EXPLICIT operator bool() ZMQ_NOTHROW { return _handle != ZMQ_NULLPTR; }
02073
02074 protected:
02075     void *_handle;
02076
02077 private:
02078     void set_option(int option_, const void *optval_, size_t optvallen_)
02079     {
02080         int rc = zmq_setsockopt(_handle, option_, optval_, optvallen_);

```

```

02081         if (rc != 0)
02082             throw error_t();
02083     }
02084
02085     void get_option(int option_, void *optval_, size_t *optvallen_) const
02086     {
02087         int rc = zmq_getsockopt(_handle, option_, optval_, optvallen_);
02088         if (rc != 0)
02089             throw error_t();
02090     }
02091 };
02092 } // namespace detail
02093
02094 struct from_handle_t
02095 {
02096     struct _private
02097     {
02098     }; // disabling use other than with from_handle
02099     ZMQ_CONSTEXPR_FN ZMQ_EXPLICIT from_handle_t(_private /*p*/) ZMQ_NOTHROW {}
02100 };
02101
02102 ZMQ_CONSTEXPR_VAR from_handle_t from_handle =
02103     from_handle_t(from_handle_t::_private());
02104
02105 // A non-owning nullable reference to a socket.
02106 // The reference is invalidated on socket close or destruction.
02107 class socket_ref : public detail::socket_base
02108 {
02109 public:
02110     socket_ref() ZMQ_NOTHROW : detail::socket_base() {}
02111 #ifdef ZMQ_CPP11
02112     socket_ref(std::nullptr_t) ZMQ_NOTHROW : detail::socket_base() {}
02113 #endif
02114     socket_ref(from_handle_t /*fh*/, void *handle) ZMQ_NOTHROW
02115         : detail::socket_base(handle)
02116     {
02117     }
02118 };
02119
02120 #ifdef ZMQ_CPP11
02121 inline bool operator==(socket_ref sr, std::nullptr_t /*p*/) ZMQ_NOTHROW
02122 {
02123     return sr.handle() == nullptr;
02124 }
02125 inline bool operator==(std::nullptr_t /*p*/, socket_ref sr) ZMQ_NOTHROW
02126 {
02127     return sr.handle() == nullptr;
02128 }
02129 inline bool operator!=(socket_ref sr, std::nullptr_t /*p*/) ZMQ_NOTHROW
02130 {
02131     return !(sr == nullptr);
02132 }
02133 inline bool operator!=(std::nullptr_t /*p*/, socket_ref sr) ZMQ_NOTHROW
02134 {
02135     return !(sr == nullptr);
02136 }
02137 #endif
02138
02139 inline bool operator==(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
02140 {
02141     return std::equal_to<const void *>()(a.handle(), b.handle());
02142 }
02143 inline bool operator!=(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
02144 {
02145     return !(a == b);
02146 }
02147 inline bool operator<(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
02148 {
02149     return std::less<const void *>()(a.handle(), b.handle());
02150 }
02151 inline bool operator>(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
02152 {
02153     return b < a;
02154 }
02155 inline bool operator<=(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
02156 {
02157     return !(a > b);
02158 }
02159 inline bool operator>=(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
02160 {
02161     return !(a < b);
02162 }
02163
02164 } // namespace zmq
02165
02166 #ifdef ZMQ_CPP11
02167 namespace std

```

```

02168 {
02169 template<> struct hash<zmq::socket_ref>
02170 {
02171     size_t operator()(zmq::socket_ref sr) const ZMQ_NOTHROW
02172     {
02173         return hash<void *>()(sr.handle());
02174     }
02175 };
02176 } // namespace std
02177 #endif
02178
02179 namespace zmq
02180 {
02181     class socket_t : public detail::socket_base
02182     {
02183     public:
02184         friend class monitor_t;
02185
02186         socket_t() ZMQ_NOTHROW : detail::socket_base(ZMQ_NULLPTR), ctxptr(ZMQ_NULLPTR) {}
02187
02188         socket_t(context_t &context_, int type_) :
02189             detail::socket_base(zmq_socket(context_.handle(), type_)),
02190             ctxptr(context_.handle())
02191         {
02192             if (_handle == ZMQ_NULLPTR)
02193                 throw error_t();
02194         }
02195
02196 #ifdef ZMQ_CPP11
02197         socket_t(context_t &context_, socket_type type_) :
02198             socket_t(context_, static_cast<int>(type_))
02199         {
02200         }
02201 #endif
02202
02203 #ifdef ZMQ_HAS_RVALUE_REFS
02204         socket_t(socket_t &&rhs) ZMQ_NOTHROW : detail::socket_base(rhs._handle),
02205             ctxptr(rhs.ctxptr)
02206         {
02207             rhs._handle = ZMQ_NULLPTR;
02208             rhs.ctxptr = ZMQ_NULLPTR;
02209         }
02210         socket_t &operator=(socket_t &&rhs) ZMQ_NOTHROW
02211         {
02212             close();
02213             std::swap(_handle, rhs._handle);
02214             std::swap(ctxptr, rhs.ctxptr);
02215             return *this;
02216         }
02217 #endif
02218
02219         ~socket_t() ZMQ_NOTHROW { close(); }
02220
02221         operator void *() ZMQ_NOTHROW { return _handle; }
02222
02223         operator void const *() const ZMQ_NOTHROW { return _handle; }
02224
02225         void close() ZMQ_NOTHROW
02226         {
02227             if (_handle == ZMQ_NULLPTR)
02228                 // already closed
02229                 return;
02230             int rc = zmq_close(_handle);
02231             ZMQ_ASSERT(rc == 0);
02232             _handle = ZMQ_NULLPTR;
02233             ctxptr = ZMQ_NULLPTR;
02234         }
02235
02236         void swap(socket_t &other) ZMQ_NOTHROW
02237         {
02238             std::swap(_handle, other._handle);
02239             std::swap(ctxptr, other.ctxptr);
02240         }
02241
02242         operator socket_ref() ZMQ_NOTHROW { return socket_ref(from_handle, _handle); }
02243
02244     private:
02245         void *ctxptr;
02246
02247         socket_t(const socket_t &) ZMQ_DELETED_FUNCTION;
02248         void operator=(const socket_t &) ZMQ_DELETED_FUNCTION;
02249
02250         // used by monitor_t
02251         socket_t(void *context_, int type_) :
02252             detail::socket_base(zmq_socket(context_, type_)), ctxptr(context_)
02253         {
02254             if (_handle == ZMQ_NULLPTR)

```

```

02255         throw error_t();
02256         if (ctxptr == ZMQ_NULLPTR)
02257             throw error_t();
02258     }
02259 };
02260
02261 inline void swap(socket_t &a, socket_t &b) ZMQ_NOTHROW
02262 {
02263     a.swap(b);
02264 }
02265
02266 ZMQ_DEPRECATED("from 4.3.1, use proxy taking socket_t objects")
02267 inline void proxy(void *frontend, void *backend, void *capture)
02268 {
02269     int rc = zmq_proxy(frontend, backend, capture);
02270     if (rc != 0)
02271         throw error_t();
02272 }
02273
02274 inline void
02275 proxy(socket_ref frontend, socket_ref backend, socket_ref capture = socket_ref())
02276 {
02277     int rc = zmq_proxy(frontend.handle(), backend.handle(), capture.handle());
02278     if (rc != 0)
02279         throw error_t();
02280 }
02281
02282 #ifdef ZMQ_HAS_PROXY_STEERABLE
02283 ZMQ_DEPRECATED("from 4.3.1, use proxy_steerable taking socket_t objects")
02284 inline void
02285 proxy_steerable(void *frontend, void *backend, void *capture, void *control)
02286 {
02287     int rc = zmq_proxy_steerable(frontend, backend, capture, control);
02288     if (rc != 0)
02289         throw error_t();
02290 }
02291
02292 inline void proxy_steerable(socket_ref frontend,
02293                             socket_ref backend,
02294                             socket_ref capture,
02295                             socket_ref control)
02296 {
02297     int rc = zmq_proxy_steerable(frontend.handle(), backend.handle(),
02298                                 capture.handle(), control.handle());
02299     if (rc != 0)
02300         throw error_t();
02301 }
02302 #endif
02303
02304 class monitor_t
02305 {
02306 public:
02307     monitor_t() : _socket(), _monitor_socket() {}
02308
02309     virtual ~monitor_t() { close(); }
02310
02311 #ifdef ZMQ_HAS_RVALUE_REFS
02312     monitor_t(monitor_t &&rhs) ZMQ_NOTHROW : _socket(), _monitor_socket()
02313     {
02314         std::swap(_socket, rhs._socket);
02315         std::swap(_monitor_socket, rhs._monitor_socket);
02316     }
02317
02318     monitor_t &operator=(monitor_t &&rhs) ZMQ_NOTHROW
02319     {
02320         close();
02321         _socket = socket_ref();
02322         std::swap(_socket, rhs._socket);
02323         std::swap(_monitor_socket, rhs._monitor_socket);
02324         return *this;
02325     }
02326 #endif
02327
02328     void
02329     monitor(socket_t &socket, std::string const &addr, int events = ZMQ_EVENT_ALL)
02330     {
02331         monitor(socket, addr.c_str(), events);
02332     }
02333
02334     void monitor(socket_t &socket, const char *addr_, int events = ZMQ_EVENT_ALL)
02335     {
02336         init(socket, addr_, events);
02337         while (true) {
02338             check_event(-1);
02339         }
02340     }
02341 }

```



```

02342
02343 void init(socket_t &socket, std::string const &addr, int events = ZMQ_EVENT_ALL)
02344 {
02345     init(socket, addr.c_str(), events);
02346 }
02347
02348 void init(socket_t &socket, const char *addr_, int events = ZMQ_EVENT_ALL)
02349 {
02350     int rc = zmq_socket_monitor(socket.handle(), addr_, events);
02351     if (rc != 0)
02352         throw error_t();
02353
02354     _socket = socket;
02355     _monitor_socket = socket_t(socket.ctxptr, ZMQ_PAIR);
02356     _monitor_socket.connect(addr_);
02357
02358     on_monitor_started();
02359 }
02360
02361 bool check_event(int timeout = 0)
02362 {
02363     assert(_monitor_socket);
02364
02365     zmq::message_t eventMsg;
02366
02367     zmq::pollitem_t items[] = {
02368         {_monitor_socket.handle(), 0, ZMQ_POLLIN, 0},
02369     };
02370
02371     #ifdef ZMQ_CPP11
02372     zmq::poll(&items[0], 1, std::chrono::milliseconds(timeout));
02373     #else
02374     zmq::poll(&items[0], 1, timeout);
02375     #endif
02376
02377     if (items[0].revents & ZMQ_POLLIN) {
02378         int rc = zmq_msg_recv(eventMsg.handle(), _monitor_socket.handle(), 0);
02379         if (rc == -1 && zmq_errno() == ETERM)
02380             return false;
02381         assert(rc != -1);
02382     } else {
02383         return false;
02384     }
02385
02386     #if ZMQ_VERSION_MAJOR >= 4
02387     const char *data = static_cast<const char *>(eventMsg.data());
02388     zmq_event_t msgEvent;
02389     memcpy(&msgEvent.event, data, sizeof(uint16_t));
02390     data += sizeof(uint16_t);
02391     memcpy(&msgEvent.value, data, sizeof(int32_t));
02392     zmq_event_t *event = &msgEvent;
02393     #else
02394     zmq_event_t *event = static_cast<zmq_event_t *>(eventMsg.data());
02395     #endif
02396
02397     #ifdef ZMQ_NEW_MONITOR_EVENT_LAYOUT
02398     zmq::message_t addrMsg;
02399     int rc = zmq_msg_recv(addrMsg.handle(), _monitor_socket.handle(), 0);
02400     if (rc == -1 && zmq_errno() == ETERM) {
02401         return false;
02402     }
02403
02404     assert(rc != -1);
02405     std::string address = addrMsg.to_string();
02406     #else
02407     // Bit of a hack, but all events in the zmq_event_t union have the same layout so this will
02408     // work for all event types.
02409     std::string address = event->data.connected.addr;
02410     #endif
02411
02412     #ifdef ZMQ_EVENT_MONITOR_STOPPED
02413     if (event->event == ZMQ_EVENT_MONITOR_STOPPED) {
02414         return false;
02415     }
02416     #endif
02417
02418     switch (event->event) {
02419     case ZMQ_EVENT_CONNECTED:
02420         on_event_connected(*event, address.c_str());
02421         break;
02422     case ZMQ_EVENT_CONNECT_DELAYED:
02423         on_event_connect_delayed(*event, address.c_str());
02424         break;
02425     case ZMQ_EVENT_CONNECT_RETRIED:
02426         on_event_connect_retried(*event, address.c_str());
02427

```

```

02428         break;
02429     case ZMQ_EVENT_LISTENING:
02430         on_event_listening(*event, address.c_str());
02431         break;
02432     case ZMQ_EVENT_BIND_FAILED:
02433         on_event_bind_failed(*event, address.c_str());
02434         break;
02435     case ZMQ_EVENT_ACCEPTED:
02436         on_event_accepted(*event, address.c_str());
02437         break;
02438     case ZMQ_EVENT_ACCEPT_FAILED:
02439         on_event_accept_failed(*event, address.c_str());
02440         break;
02441     case ZMQ_EVENT_CLOSED:
02442         on_event_closed(*event, address.c_str());
02443         break;
02444     case ZMQ_EVENT_CLOSE_FAILED:
02445         on_event_close_failed(*event, address.c_str());
02446         break;
02447     case ZMQ_EVENT_DISCONNECTED:
02448         on_event_disconnected(*event, address.c_str());
02449         break;
02450     #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 3, 0) || (defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >=
ZMQ_MAKE_VERSION(4, 2, 3))
02451     case ZMQ_EVENT_HANDSHAKE_FAILED_NO_DETAIL:
02452         on_event_handshake_failed_no_detail(*event, address.c_str());
02453         break;
02454     case ZMQ_EVENT_HANDSHAKE_FAILED_PROTOCOL:
02455         on_event_handshake_failed_protocol(*event, address.c_str());
02456         break;
02457     case ZMQ_EVENT_HANDSHAKE_FAILED_AUTH:
02458         on_event_handshake_failed_auth(*event, address.c_str());
02459         break;
02460     case ZMQ_EVENT_HANDSHAKE_SUCCEEDED:
02461         on_event_handshake_succeeded(*event, address.c_str());
02462         break;
02463     #elif defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 1)
02464     case ZMQ_EVENT_HANDSHAKE_FAILED:
02465         on_event_handshake_failed(*event, address.c_str());
02466         break;
02467     case ZMQ_EVENT_HANDSHAKE_SUCCEED:
02468         on_event_handshake_succeed(*event, address.c_str());
02469         break;
02470     #endif
02471     default:
02472         on_event_unknown(*event, address.c_str());
02473         break;
02474 }
02475
02476     return true;
02477 }
02478
02479 #ifndef ZMQ_EVENT_MONITOR_STOPPED
02480 void abort()
02481 {
02482     if (_socket)
02483         zmq_socket_monitor(_socket.handle(), ZMQ_NULLPTR, 0);
02484
02485     _socket = socket_ref();
02486 }
02487 #endif
02488 virtual void on_monitor_started() {}
02489 virtual void on_event_connected(const zmq_event_t &event_, const char *addr_)
02490 {
02491     (void) event_;
02492     (void) addr_;
02493 }
02494 virtual void on_event_connect_delayed(const zmq_event_t &event_,
const char *addr_)
02495 {
02496     (void) event_;
02497     (void) addr_;
02498 }
02499 }
02500 virtual void on_event_connect_retried(const zmq_event_t &event_,
const char *addr_)
02501 {
02502     (void) event_;
02503     (void) addr_;
02504 }
02505 }
02506 virtual void on_event_listening(const zmq_event_t &event_, const char *addr_)
02507 {
02508     (void) event_;
02509     (void) addr_;
02510 }
02511 virtual void on_event_bind_failed(const zmq_event_t &event_, const char *addr_)
02512 {
02513     (void) event_;

```

```

02514         (void) addr_;
02515     }
02516     virtual void on_event_accepted(const zmq_event_t &event_, const char *addr_)
02517     {
02518         (void) event_;
02519         (void) addr_;
02520     }
02521     virtual void on_event_accept_failed(const zmq_event_t &event_, const char *addr_)
02522     {
02523         (void) event_;
02524         (void) addr_;
02525     }
02526     virtual void on_event_closed(const zmq_event_t &event_, const char *addr_)
02527     {
02528         (void) event_;
02529         (void) addr_;
02530     }
02531     virtual void on_event_close_failed(const zmq_event_t &event_, const char *addr_)
02532     {
02533         (void) event_;
02534         (void) addr_;
02535     }
02536     virtual void on_event_disconnected(const zmq_event_t &event_, const char *addr_)
02537     {
02538         (void) event_;
02539         (void) addr_;
02540     }
02541     #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 3)
02542     virtual void on_event_handshake_failed_no_detail(const zmq_event_t &event_,
02543                                                     const char *addr_)
02544     {
02545         (void) event_;
02546         (void) addr_;
02547     }
02548     virtual void on_event_handshake_failed_protocol(const zmq_event_t &event_,
02549                                                     const char *addr_)
02550     {
02551         (void) event_;
02552         (void) addr_;
02553     }
02554     virtual void on_event_handshake_failed_auth(const zmq_event_t &event_,
02555                                                  const char *addr_)
02556     {
02557         (void) event_;
02558         (void) addr_;
02559     }
02560     virtual void on_event_handshake_succeeded(const zmq_event_t &event_,
02561                                                const char *addr_)
02562     {
02563         (void) event_;
02564         (void) addr_;
02565     }
02566     #elif ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 1)
02567     virtual void on_event_handshake_failed(const zmq_event_t &event_,
02568                                           const char *addr_)
02569     {
02570         (void) event_;
02571         (void) addr_;
02572     }
02573     virtual void on_event_handshake_succeed(const zmq_event_t &event_,
02574                                             const char *addr_)
02575     {
02576         (void) event_;
02577         (void) addr_;
02578     }
02579     #endif
02580     virtual void on_event_unknown(const zmq_event_t &event_, const char *addr_)
02581     {
02582         (void) event_;
02583         (void) addr_;
02584     }
02585 private:
02586     monitor_t(const monitor_t &) ZMQ_DELETED_FUNCTION;
02587     void operator=(const monitor_t &) ZMQ_DELETED_FUNCTION;
02588     socket_ref _socket;
02589     socket_t _monitor_socket;
02590     void close() ZMQ_NOTHROW
02591     {
02592         if (_socket)
02593             zmq_socket_monitor(_socket.handle(), ZMQ_NULLPTR, 0);
02594         _monitor_socket.close();
02595     }
02596 };
02597
02600

```

```

02601 #if defined(ZMQ_BUILD_DRAFT_API) && defined(ZMQ_CPP11) && defined(ZMQ_HAVE_POLLER)
02602
02603 // polling events
02604 enum class event_flags : short
02605 {
02606     none = 0,
02607     pollin = ZMQ_POLLIN,
02608     pollout = ZMQ_POLLOUT,
02609     pollerr = ZMQ_POLLERR,
02610     pollpri = ZMQ_POLLPRI
02611 };
02612
02613 constexpr event_flags operator|(event_flags a, event_flags b) noexcept
02614 {
02615     return detail::enum_bit_or(a, b);
02616 }
02617 constexpr event_flags operator&(event_flags a, event_flags b) noexcept
02618 {
02619     return detail::enum_bit_and(a, b);
02620 }
02621 constexpr event_flags operator^(event_flags a, event_flags b) noexcept
02622 {
02623     return detail::enum_bit_xor(a, b);
02624 }
02625 constexpr event_flags operator~(event_flags a) noexcept
02626 {
02627     return detail::enum_bit_not(a);
02628 }
02629
02630 struct no_user_data;
02631
02632 // layout compatible with zmq_poller_event_t
02633 template<class T = no_user_data> struct poller_event
02634 {
02635     socket_ref socket;
02636     ::zmq::fd_t fd;
02637     T *user_data;
02638     event_flags events;
02639 };
02640
02641 template<typename T = no_user_data> class poller_t
02642 {
02643 public:
02644     using event_type = poller_event<T>;
02645
02646     poller_t() : poller_ptr(zmq_poller_new())
02647     {
02648         if (!poller_ptr)
02649             throw error_t();
02650     }
02651
02652     template<
02653         typename Dummy = void,
02654         typename =
02655             typename std::enable_if<!std::is_same<T, no_user_data>::value, Dummy>::type>
02656     void add(zmq::socket_ref socket, event_flags events, T *user_data)
02657     {
02658         add_impl(socket, events, user_data);
02659     }
02660
02661     void add(zmq::socket_ref socket, event_flags events)
02662     {
02663         add_impl(socket, events, nullptr);
02664     }
02665
02666     template<
02667         typename Dummy = void,
02668         typename =
02669             typename std::enable_if<!std::is_same<T, no_user_data>::value, Dummy>::type>
02670     void add(fd_t fd, event_flags events, T *user_data)
02671     {
02672         add_impl(fd, events, user_data);
02673     }
02674
02675     void add(fd_t fd, event_flags events) { add_impl(fd, events, nullptr); }
02676
02677     void remove(zmq::socket_ref socket)
02678     {
02679         if (0 != zmq_poller_remove(poller_ptr.get(), socket.handle())) {
02680             throw error_t();
02681         }
02682     }
02683
02684     void modify(zmq::socket_ref socket, event_flags events)
02685     {
02686         if (0
02687             != zmq_poller_modify(poller_ptr.get(), socket.handle(),

```

```

02688                                     static_cast<short>(events))) {
02689     throw error_t();
02690 }
02691 }
02692
02693 size_t wait_all(std::vector<event_type> &poller_events,
02694               const std::chrono::milliseconds timeout)
02695 {
02696     int rc = zmq_poller_wait_all(
02697         poller_ptr.get(),
02698         reinterpret_cast<zmq_poller_event_t *>(poller_events.data()),
02699         static_cast<int>(poller_events.size()),
02700         static_cast<long>(timeout.count()));
02701     if (rc > 0)
02702         return static_cast<size_t>(rc);
02703
02704 #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 3)
02705     if (zmq_errno() == EAGAIN)
02706 #else
02707     if (zmq_errno() == ETIMEDOUT)
02708 #endif
02709         return 0;
02710
02711     throw error_t();
02712 }
02713
02714 #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 3, 3)
02715 size_t size() const noexcept
02716 {
02717     int rc = zmq_poller_size(const_cast<void *>(poller_ptr.get()));
02718     ZMQ_ASSERT(rc >= 0);
02719     return static_cast<size_t>(std::max(rc, 0));
02720 }
02721 #endif
02722
02723 private:
02724     struct destroy_poller_t
02725     {
02726         void operator()(void *ptr) noexcept
02727         {
02728             int rc = zmq_poller_destroy(&ptr);
02729             ZMQ_ASSERT(rc == 0);
02730         }
02731     };
02732
02733     std::unique_ptr<void, destroy_poller_t> poller_ptr;
02734
02735     void add_impl(zmq::socket_ref socket, event_flags events, T *user_data)
02736     {
02737         if (0
02738             != zmq_poller_add(poller_ptr.get(), socket.handle(), user_data,
02739                               static_cast<short>(events))) {
02740             throw error_t();
02741         }
02742     }
02743
02744     void add_impl(fd_t fd, event_flags events, T *user_data)
02745     {
02746         if (0
02747             != zmq_poller_add_fd(poller_ptr.get(), fd, user_data,
02748                                   static_cast<short>(events))) {
02749             throw error_t();
02750         }
02751     }
02752 };
02753 #endif // defined(ZMQ_BUILD_DRAFT_API) && defined(ZMQ_CPP11) && defined(ZMQ_HAVE_POLLER)
02754
02755 inline std::ostream &operator<<(std::ostream &os, const message_t &msg)
02756 {
02757     return os << msg.str();
02758 }
02759
02760 } // namespace zmq
02761
02762 #endif // __ZMQ_HPP_INCLUDED__

```

7.29 external/zmq/includes/zmq/zmq_addon.hpp File Reference

```

#include "zmq.hpp"
#include <deque>
#include <iomanip>
#include <sstream>

```

```
#include <stdexcept>
```

Namespaces

- namespace `zmq`

7.30 zmq_addon.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002     Copyright (c) 2016-2017 ZeroMQ community
00003     Copyright (c) 2016 VOCA AS / Harald Nøkland
00004
00005     Permission is hereby granted, free of charge, to any person obtaining a copy
00006     of this software and associated documentation files (the "Software"), to
00007     deal in the Software without restriction, including without limitation the
00008     rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
00009     sell copies of the Software, and to permit persons to whom the Software is
00010     furnished to do so, subject to the following conditions:
00011
00012     The above copyright notice and this permission notice shall be included in
00013     all copies or substantial portions of the Software.
00014
00015     THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00016     IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00017     FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00018     AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00019     LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
00020     FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
00021     IN THE SOFTWARE.
00022 */
00023
00024 #ifndef __ZMQ_ADDON_HPP_INCLUDED__
00025 #define __ZMQ_ADDON_HPP_INCLUDED__
00026
00027 #include "zmq.hpp"
00028
00029 #include <deque>
00030 #include <iomanip>
00031 #include <sstream>
00032 #include <stdexcept>
00033 #ifdef ZMQ_CPP11
00034 #include <limits>
00035 #include <functional>
00036 #include <unordered_map>
00037 #endif
00038
00039 namespace zmq
00040 {
00041 #ifdef ZMQ_CPP11
00042     namespace detail
00043     {
00044         template<bool CheckN, class OutputIt>
00045         recv_result_t
00046         recv_multipart_n(socket_ref s, OutputIt out, size_t n, recv_flags flags)
00047         {
00048             size_t msg_count = 0;
00049             message_t msg;
00050             while (true) {
00051                 if ZMQ_CONSTEXPR_IF (CheckN) {
00052                     if (msg_count >= n)
00053                         throw std::runtime_error(
00054                             "Too many message parts in recv_multipart_n");
00055                 }
00056                 if (!s.recv(msg, flags)) {
00057                     // zmq ensures atomic delivery of messages
00058                     assert(msg_count == 0);
00059                     return {};
00060                 }
00061                 ++msg_count;
00062                 const bool more = msg.more();
00063                 *out++ = std::move(msg);
00064                 if (!more)
00065                     break;
00066             }
00067             return msg_count;
00068         }
00069     }
00070
00071     inline bool is_little_endian()
00072     {
```

```

00073     const uint16_t i = 0x01;
00074     return *reinterpret_cast<const uint8_t *>(&i) == 0x01;
00075 }
00076
00077 inline void write_network_order(unsigned char *buf, const uint32_t value)
00078 {
00079     if (is_little_endian()) {
00080         ZMQ_CONTEXPR_VAR uint32_t mask = (std::numeric_limits<std::uint8_t>::max)();
00081         *buf++ = static_cast<unsigned char>((value >> 24) & mask);
00082         *buf++ = static_cast<unsigned char>((value >> 16) & mask);
00083         *buf++ = static_cast<unsigned char>((value >> 8) & mask);
00084         *buf++ = static_cast<unsigned char>(value & mask);
00085     } else {
00086         std::memcpy(buf, &value, sizeof(value));
00087     }
00088 }
00089
00090 inline uint32_t read_u32_network_order(const unsigned char *buf)
00091 {
00092     if (is_little_endian()) {
00093         return (static_cast<uint32_t>(buf[0]) << 24)
00094             + (static_cast<uint32_t>(buf[1]) << 16)
00095             + (static_cast<uint32_t>(buf[2]) << 8)
00096             + static_cast<uint32_t>(buf[3]);
00097     } else {
00098         uint32_t value;
00099         std::memcpy(&value, buf, sizeof(value));
00100         return value;
00101     }
00102 }
00103 } // namespace detail
00104
00105 /* Receive a multipart message.
00106
00107 Writes the zmq::message_t objects to OutputIterator out.
00108 The out iterator must handle an unspecified number of writes,
00109 e.g. by using std::back_inserter.
00110
00111 Returns: the number of messages received or nullopt (on EAGAIN).
00112 Throws: if recv throws. Any exceptions thrown
00113 by the out iterator will be propagated and the message
00114 may have been only partially received with pending
00115 message parts. It is advised to close this socket in that event.
00116 */
00117 template<class OutputIt>
00118 ZMQ_NODISCARD recv_result_t recv_multipart(socket_ref s,
00119     OutputIt out,
00120     recv_flags flags = recv_flags::none)
00121 {
00122     return detail::recv_multipart_n<false>(s, std::move(out), 0, flags);
00123 }
00124
00125 /* Receive a multipart message.
00126
00127 Writes at most n zmq::message_t objects to OutputIterator out.
00128 If the number of message parts of the incoming message exceeds n
00129 then an exception will be thrown.
00130
00131 Returns: the number of messages received or nullopt (on EAGAIN).
00132 Throws: if recv throws. Throws std::runtime_error if the number
00133 of message parts exceeds n (exactly n messages will have been written
00134 to out). Any exceptions thrown
00135 by the out iterator will be propagated and the message
00136 may have been only partially received with pending
00137 message parts. It is advised to close this socket in that event.
00138 */
00139 template<class OutputIt>
00140 ZMQ_NODISCARD recv_result_t recv_multipart_n(socket_ref s,
00141     OutputIt out,
00142     size_t n,
00143     recv_flags flags = recv_flags::none)
00144 {
00145     return detail::recv_multipart_n<true>(s, std::move(out), n, flags);
00146 }
00147
00148 /* Send a multipart message.
00149
00150 The range must be a ForwardRange of zmq::message_t,
00151 zmq::const_buffer or zmq::mutable_buffer.
00152 The flags may be zmq::send_flags::sndmore if there are
00153 more message parts to be sent after the call to this function.
00154
00155 Returns: the number of messages sent (exactly msgs.size()) or nullopt (on EAGAIN).
00156 Throws: if send throws. Any exceptions thrown
00157 by the msgs range will be propagated and the message
00158 may have been only partially sent. It is advised to close this socket in that event.
00159 */

```

```

00160 template<class Range
00161 #ifndef ZMQ_CPP11_PARTIAL
00162     ,
00163     typename = typename std::enable_if<
00164         detail::is_range<Range>::value
00165         && (std::is_same<detail::range_value_t<Range>, message_t>::value
00166             || detail::is_buffer<detail::range_value_t<Range>::value)>::type
00167 #endif
00168     >
00169 send_result_t
00170 send_multipart(socket_ref s, Range &msgs, send_flags flags = send_flags::none)
00171 {
00172     using std::begin;
00173     using std::end;
00174     auto it = begin(msgs);
00175     const auto end_it = end(msgs);
00176     size_t msg_count = 0;
00177     while (it != end_it) {
00178         const auto next = std::next(it);
00179         const auto msg_flags =
00180             flags | (next == end_it ? send_flags::none : send_flags::sndmore);
00181         if (!s.send(*it, msg_flags)) {
00182             // zmq ensures atomic delivery of messages
00183             assert(it == begin(msgs));
00184             return {};
00185         }
00186         ++msg_count;
00187         it = next;
00188     }
00189     return msg_count;
00190 }
00191
00192 /* Encode a multipart message.
00193
00194     The range must be a ForwardRange of zmq::message_t. A
00195     zmq::multipart_t or STL container may be passed for encoding.
00196
00197     Returns: a zmq::message_t holding the encoded multipart data.
00198
00199     Throws: std::range_error is thrown if the size of any single part
00200     can not fit in an unsigned 32 bit integer.
00201
00202     The encoding is compatible with that used by the CZMQ function
00203     zmqmsg_encode(), see https://rfc.zeromq.org/spec/50/.
00204     Each part consists of a size followed by the data.
00205     These are placed contiguously into the output message. A part of
00206     size less than 255 bytes will have a single byte size value.
00207     Larger parts will have a five byte size value with the first byte
00208     set to 0xFF and the remaining four bytes holding the size of the
00209     part's data.
00210 */
00211 template<class Range
00212 #ifndef ZMQ_CPP11_PARTIAL
00213     ,
00214     typename = typename std::enable_if<
00215         detail::is_range<Range>::value
00216         && (std::is_same<detail::range_value_t<Range>, message_t>::value
00217             || detail::is_buffer<detail::range_value_t<Range>::value)>::type
00218 #endif
00219     >
00220 message_t encode(const Range &parts)
00221 {
00222     size_t mmsg_size = 0;
00223
00224     // First pass check sizes
00225     for (const auto &part : parts) {
00226         const size_t part_size = part.size();
00227         if (part_size > (std::numeric_limits<std::uint32_t>::max)()) {
00228             // Size value must fit into uint32_t.
00229             throw std::range_error("Invalid size, message part too large");
00230         }
00231         const size_t count_size =
00232             part_size < (std::numeric_limits<std::uint8_t>::max)() ? 1 : 5;
00233         mmsg_size += part_size + count_size;
00234     }
00235
00236     message_t encoded(mmsg_size);
00237     unsigned char *buf = encoded.data<unsigned char>();
00238     for (const auto &part : parts) {
00239         const uint32_t part_size = static_cast<uint32_t>(part.size());
00240         const unsigned char *part_data =
00241             static_cast<const unsigned char *>(part.data());
00242
00243         if (part_size < (std::numeric_limits<std::uint8_t>::max)()) {
00244             // small part
00245             *buf++ = (unsigned char) part_size;
00246         } else {

```



```

00247         // big part
00248         *buf++ = (std::numeric_limits<uint8_t>::max)();
00249         detail::write_network_order(buf, part_size);
00250         buf += sizeof(part_size);
00251     }
00252     std::memcpy(buf, part_data, part_size);
00253     buf += part_size;
00254 }
00255
00256 assert(static_cast<size_t>(buf - encoded.data<unsigned char>()) == mmsg_size);
00257 return encoded;
00258 }
00259
00260 /* Decode an encoded message to multiple parts.
00261
00262 The given output iterator must be a ForwardIterator to a container
00263 holding zmq::message_t such as a zmq::multipart_t or various STL
00264 containers.
00265
00266 Returns the ForwardIterator advanced once past the last decoded
00267 part.
00268
00269 Throws: a std::out_of_range is thrown if the encoded part sizes
00270 lead to exceeding the message data bounds.
00271
00272 The decoding assumes the message is encoded in the manner
00273 performed by zmq::encode(), see https://rfc.zeromq.org/spec/50/.
00274 */
00275 template<class OutputIt> OutputIt decode(const message_t &encoded, OutputIt out)
00276 {
00277     const unsigned char *source = encoded.data<unsigned char>();
00278     const unsigned char *const limit = source + encoded.size();
00279
00280     while (source < limit) {
00281         size_t part_size = *source++;
00282         if (part_size == (std::numeric_limits<std::uint8_t>::max)()) {
00283             if (static_cast<size_t>(limit - source) < sizeof(uint32_t)) {
00284                 throw std::out_of_range(
00285                     "Malformed encoding, overflow in reading size");
00286             }
00287             part_size = detail::read_u32_network_order(source);
00288             // the part size is allowed to be less than 0xFF
00289             source += sizeof(uint32_t);
00290         }
00291
00292         if (static_cast<size_t>(limit - source) < part_size) {
00293             throw std::out_of_range("Malformed encoding, overflow in reading part");
00294         }
00295         *out = message_t(source, part_size);
00296         ++out;
00297         source += part_size;
00298     }
00299
00300     assert(source == limit);
00301     return out;
00302 }
00303
00304 #endif
00305
00306 #ifndef ZMQ_HAS_RVALUE_REFS
00307
00308 /*
00309 This class handles multipart messaging. It is the C++ equivalent of zmqmsg.h,
00310 which is part of CZMQ (the high-level C binding). Furthermore, it is a major
00311 improvement compared to zmqmsg.hpp, which is part of the examples in the ØMQ
00312 Guide. Unnecessary copying is avoided by using move semantics to efficiently
00313 add/remove parts.
00314 */
00315
00316 class multipart_t
00317 {
00318 private:
00319     std::deque<message_t> m_parts;
00320
00321 public:
00322     typedef std::deque<message_t>::value_type value_type;
00323
00324     typedef std::deque<message_t>::iterator iterator;
00325     typedef std::deque<message_t>::const_iterator const_iterator;
00326
00327     typedef std::deque<message_t>::reverse_iterator reverse_iterator;
00328     typedef std::deque<message_t>::const_reverse_iterator const_reverse_iterator;
00329
00330     // Default constructor
00331     multipart_t() {}
00332
00333     // Construct from socket receive

```

```

00334     multipart_t(socket_ref socket) { recv(socket); }
00335
00336     // Construct from memory block
00337     multipart_t(const void *src, size_t size) { addmem(src, size); }
00338
00339     // Construct from string
00340     multipart_t(const std::string &string) { addstr(string); }
00341
00342     // Construct from message part
00343     multipart_t(message_t &&message) { add(std::move(message)); }
00344
00345     // Move constructor
00346     multipart_t(multipart_t &&other) ZMQ_NOTHROW { m_parts = std::move(other.m_parts); }
00347
00348     // Move assignment operator
00349     multipart_t &operator=(multipart_t &&other) ZMQ_NOTHROW
00350     {
00351         m_parts = std::move(other.m_parts);
00352         return *this;
00353     }
00354
00355     // Destructor
00356     virtual ~multipart_t() { clear(); }
00357
00358     message_t &operator[](size_t n) { return m_parts[n]; }
00359
00360     const message_t &operator[](size_t n) const { return m_parts[n]; }
00361
00362     message_t &at(size_t n) { return m_parts.at(n); }
00363
00364     const message_t &at(size_t n) const { return m_parts.at(n); }
00365
00366     iterator begin() { return m_parts.begin(); }
00367
00368     const_iterator begin() const { return m_parts.begin(); }
00369
00370     const_iterator cbegin() const { return m_parts.cbegin(); }
00371
00372     reverse_iterator rbegin() { return m_parts.rbegin(); }
00373
00374     const_reverse_iterator rbegin() const { return m_parts.rbegin(); }
00375
00376     iterator end() { return m_parts.end(); }
00377
00378     const_iterator end() const { return m_parts.end(); }
00379
00380     const_iterator cend() const { return m_parts.cend(); }
00381
00382     reverse_iterator rend() { return m_parts.rend(); }
00383
00384     const_reverse_iterator rend() const { return m_parts.rend(); }
00385
00386     // Delete all parts
00387     void clear() { m_parts.clear(); }
00388
00389     // Get number of parts
00390     size_t size() const { return m_parts.size(); }
00391
00392     // Check if number of parts is zero
00393     bool empty() const { return m_parts.empty(); }
00394
00395     // Receive multipart message from socket
00396     bool recv(socket_ref socket, int flags = 0)
00397     {
00398         clear();
00399         bool more = true;
00400         while (more) {
00401             message_t message;
00402 #ifdef ZMQ_CPP11
00403             if (!socket.recv(message, static_cast<recv_flags>(flags)))
00404                 return false;
00405 #else
00406             if (!socket.recv(&message, flags))
00407                 return false;
00408 #endif
00409             more = message.more();
00410             add(std::move(message));
00411         }
00412         return true;
00413     }
00414
00415     // Send multipart message to socket
00416     bool send(socket_ref socket, int flags = 0)
00417     {
00418         flags &= ~(ZMQ_SNDMORE);
00419         bool more = size() > 0;
00420         while (more) {

```

```

00421         message_t message = pop();
00422         more = size() > 0;
00423 #ifdef ZMQ_CPP11
00424         if (!socket.send(message, static_cast<send_flags>((
00425             more ? ZMQ_SNDMORE : 0) | flags)))
00426             return false;
00427 #else
00428         if (!socket.send(message, (more ? ZMQ_SNDMORE : 0) | flags))
00429             return false;
00430 #endif
00431     }
00432     clear();
00433     return true;
00434 }
00435
00436 // Concatenate other multipart to front
00437 void prepend(multipart_t &&other)
00438 {
00439     while (!other.empty())
00440         push(other.remove());
00441 }
00442
00443 // Concatenate other multipart to back
00444 void append(multipart_t &&other)
00445 {
00446     while (!other.empty())
00447         add(other.pop());
00448 }
00449
00450 // Push memory block to front
00451 void pushmem(const void *src, size_t size)
00452 {
00453     m_parts.push_front(message_t(src, size));
00454 }
00455
00456 // Push memory block to back
00457 void addmem(const void *src, size_t size)
00458 {
00459     m_parts.push_back(message_t(src, size));
00460 }
00461
00462 // Push string to front
00463 void pushstr(const std::string &string)
00464 {
00465     m_parts.push_front(message_t(string.data(), string.size()));
00466 }
00467
00468 // Push string to back
00469 void addstr(const std::string &string)
00470 {
00471     m_parts.push_back(message_t(string.data(), string.size()));
00472 }
00473
00474 // Push type (fixed-size) to front
00475 template<typename T> void pushtyp(const T &type)
00476 {
00477     static_assert(!std::is_same<T, std::string>::value,
00478         "Use pushstr() instead of pushtyp<std::string>()");
00479     m_parts.push_front(message_t(&type, sizeof(type)));
00480 }
00481
00482 // Push type (fixed-size) to back
00483 template<typename T> void addtyp(const T &type)
00484 {
00485     static_assert(!std::is_same<T, std::string>::value,
00486         "Use addstr() instead of addtyp<std::string>()");
00487     m_parts.push_back(message_t(&type, sizeof(type)));
00488 }
00489
00490 // Push message part to front
00491 void push(message_t &&message) { m_parts.push_front(std::move(message)); }
00492
00493 // Push message part to back
00494 void add(message_t &&message) { m_parts.push_back(std::move(message)); }
00495
00496 // Alias to allow std::back_inserter()
00497 void push_back(message_t &&message) { m_parts.push_back(std::move(message)); }
00498
00499 // Pop string from front
00500 std::string popstr()
00501 {
00502     std::string string(m_parts.front().data<char>(), m_parts.front().size());
00503     m_parts.pop_front();
00504     return string;
00505 }
00506
00507 // Pop type (fixed-size) from front

```

```

00508     template<typename T> T poptyp()
00509     {
00510         static_assert(!std::is_same<T, std::string>::value,
00511             "Use popstr() instead of poptyp<std::string>()");
00512         if (sizeof(T) != m_parts.front().size())
00513             throw std::runtime_error(
00514                 "Invalid type, size does not match the message size");
00515         T type = *m_parts.front().data<T>();
00516         m_parts.pop_front();
00517         return type;
00518     }
00519
00520     // Pop message part from front
00521     message_t pop()
00522     {
00523         message_t message = std::move(m_parts.front());
00524         m_parts.pop_front();
00525         return message;
00526     }
00527
00528     // Pop message part from back
00529     message_t remove()
00530     {
00531         message_t message = std::move(m_parts.back());
00532         m_parts.pop_back();
00533         return message;
00534     }
00535
00536     // get message part from front
00537     const message_t &front() { return m_parts.front(); }
00538
00539     // get message part from back
00540     const message_t &back() { return m_parts.back(); }
00541
00542     // Get pointer to a specific message part
00543     const message_t *peek(size_t index) const { return &m_parts[index]; }
00544
00545     // Get a string copy of a specific message part
00546     std::string peekstr(size_t index) const
00547     {
00548         std::string string(m_parts[index].data<char>(), m_parts[index].size());
00549         return string;
00550     }
00551
00552     // Peek type (fixed-size) from front
00553     template<typename T> T peektyp(size_t index) const
00554     {
00555         static_assert(!std::is_same<T, std::string>::value,
00556             "Use peekstr() instead of peektyp<std::string>()");
00557         if (sizeof(T) != m_parts[index].size())
00558             throw std::runtime_error(
00559                 "Invalid type, size does not match the message size");
00560         T type = *m_parts[index].data<T>();
00561         return type;
00562     }
00563
00564     // Create multipart from type (fixed-size)
00565     template<typename T> static multipart_t create(const T &type)
00566     {
00567         multipart_t multipart;
00568         multipart.addtyp(type);
00569         return multipart;
00570     }
00571
00572     // Copy multipart
00573     multipart_t clone() const
00574     {
00575         multipart_t multipart;
00576         for (size_t i = 0; i < size(); i++)
00577             multipart.addmem(m_parts[i].data(), m_parts[i].size());
00578         return multipart;
00579     }
00580
00581     // Dump content to string
00582     std::string str() const
00583     {
00584         std::stringstream ss;
00585         for (size_t i = 0; i < m_parts.size(); i++) {
00586             const unsigned char *data = m_parts[i].data<unsigned char>();
00587             size_t size = m_parts[i].size();
00588
00589             // Dump the message as text or binary
00590             bool isText = true;
00591             for (size_t j = 0; j < size; j++) {
00592                 if (data[j] < 32 || data[j] > 127) {
00593                     isText = false;
00594                     break;

```

```

00595     }
00596     }
00597     ss << "\n[" << std::dec << std::setw(3) << std::setfill('0') << size
00598     << "] ";
00599     if (size >= 1000) {
00600         ss << "... (too big to print)";
00601         continue;
00602     }
00603     for (size_t j = 0; j < size; j++) {
00604         if (isText)
00605             ss << static_cast<char>(data[j]);
00606         else
00607             ss << std::hex << std::setw(2) << std::setfill('0')
00608             << static_cast<short>(data[j]);
00609     }
00610 }
00611 return ss.str();
00612 }
00613
00614 // Check if equal to other multipart
00615 bool equal(const multipart_t &other) const ZMQ_NOTHROW
00616 {
00617     return *this == *other;
00618 }
00619
00620 bool operator==(const multipart_t &other) const ZMQ_NOTHROW
00621 {
00622     if (size() != other.size())
00623         return false;
00624     for (size_t i = 0; i < size(); i++)
00625         if (at(i) != other.at(i))
00626             return false;
00627     return true;
00628 }
00629
00630 bool operator!=(const multipart_t &other) const ZMQ_NOTHROW
00631 {
00632     return !(*this == other);
00633 }
00634
00635 #ifdef ZMQ_CPP11
00636
00637 // Return single part message_t encoded from this multipart_t.
00638 message_t encode() const { return zmq::encode(*this); }
00639
00640 // Decode encoded message into multiple parts and append to self.
00641 void decode_append(const message_t &encoded)
00642 {
00643     zmq::decode(encoded, std::back_inserter(*this));
00644 }
00645
00646 // Return a new multipart_t containing the decoded message_t.
00647 static multipart_t decode(const message_t &encoded)
00648 {
00649     multipart_t tmp;
00650     zmq::decode(encoded, std::back_inserter(tmp));
00651     return tmp;
00652 }
00653
00654 #endif
00655
00656 private:
00657     // Disable implicit copying (moving is more efficient)
00658     multipart_t(const multipart_t &other) ZMQ_DELETED_FUNCTION;
00659     void operator=(const multipart_t &other) ZMQ_DELETED_FUNCTION;
00660 }; // class multipart_t
00661
00662 inline std::ostream &operator<<(std::ostream &os, const multipart_t &msg)
00663 {
00664     return os << msg.str();
00665 }
00666
00667 #endif // ZMQ_HAS_RVALUE_REFS
00668
00669 #if defined(ZMQ_BUILD_DRAFT_API) && defined(ZMQ_CPP11) && defined(ZMQ_HAVE_POLLER)
00670 class active_poller_t
00671 {
00672 public:
00673     active_poller_t() = default;
00674     ~active_poller_t() = default;
00675
00676     active_poller_t(const active_poller_t &) = delete;
00677     active_poller_t &operator=(const active_poller_t &) = delete;
00678
00679     active_poller_t(active_poller_t &&src) = default;
00680     active_poller_t &operator=(active_poller_t &&src) = default;
00681

```

```

00682     using handler_type = std::function<void(event_flags)>;
00683
00684     void add(zmq::socket_ref socket, event_flags events, handler_type handler)
00685     {
00686         if (!handler)
00687             throw std::invalid_argument("null handler in active_poller_t::add");
00688         auto ret = handlers.emplace(
00689             socket, std::make_shared<handler_type>(std::move(handler)));
00690         if (!ret.second)
00691             throw error_t(EINVAL); // already added
00692         try {
00693             base_poller.add(socket, events, ret.first->second.get());
00694             need_rebuild = true;
00695         }
00696         catch (...) {
00697             // rollback
00698             handlers.erase(socket);
00699             throw;
00700         }
00701     }
00702
00703     void remove(zmq::socket_ref socket)
00704     {
00705         base_poller.remove(socket);
00706         handlers.erase(socket);
00707         need_rebuild = true;
00708     }
00709
00710     void modify(zmq::socket_ref socket, event_flags events)
00711     {
00712         base_poller.modify(socket, events);
00713     }
00714
00715     size_t wait(std::chrono::milliseconds timeout)
00716     {
00717         if (need_rebuild) {
00718             poller_events.resize(handlers.size());
00719             poller_handlers.clear();
00720             poller_handlers.reserve(handlers.size());
00721             for (const auto &handler : handlers) {
00722                 poller_handlers.push_back(handler.second);
00723             }
00724             need_rebuild = false;
00725         }
00726         const auto count = base_poller.wait_all(poller_events, timeout);
00727         std::for_each(poller_events.begin(),
00728             poller_events.begin() + static_cast<ptrdiff_t>(count),
00729             [](decltype(base_poller)::event_type &event) {
00730                 assert(event.user_data != nullptr);
00731                 (*event.user_data)(event.events);
00732             });
00733         return count;
00734     }
00735
00736     ZMQ_NODISCARD bool empty() const noexcept { return handlers.empty(); }
00737
00738     size_t size() const noexcept { return handlers.size(); }
00739
00740 private:
00741     bool need_rebuild{false};
00742
00743     poller_t<handler_type> base_poller{};
00744     std::unordered_map<socket_ref, std::shared_ptr<handler_type>> handlers{};
00745     std::vector<decltype(base_poller)::event_type> poller_events{};
00746     std::vector<std::shared_ptr<handler_type>> poller_handlers{};
00747 }; // class active_poller_t
00748 #endif // defined(ZMQ_BUILD_DRAFT_API) && defined(ZMQ_CPP11) && defined(ZMQ_HAVE_POLLER)
00749
00750
00751 } // namespace zmq
00752
00753 #endif // __ZMQ_ADDON_HPP_INCLUDED__

```

7.31 includes/LibZMQUtils/CommandServerClient/command_client.h

File Reference

This file contains the declaration of the CommandClientBase class and related.

```

#include <future>
#include <map>
#include "LibZMQUtils/libzmqutils_global.h"
#include "LibZMQUtils/CommandServerClient/common.h"

```

Classes

- class [zmqutils::CommandClientBase](#)

Namespaces

- namespace [zmq](#)
- namespace [zmqutils](#)

7.31.1 Detailed Description

This file contains the declaration of the `CommandClientBase` class and related.

Author

Degoras Project Team

Copyright

EUPL License

Version

2307.1

Definition in file [command_client.h](#).

7.32 command_client.h

[Go to the documentation of this file.](#)

```

00001
00002  /*****
00003  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
00004  *
00005  *   Copyright (C) 2023 Degoras Project Team
00006  *
00007  *   < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
00008  *   < Jesús Relinque Madroñal >
00009  *
00010  *   This file is part of LibZMQUtils.
00011  *
00012  *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
00013  *   the EUPL license
00014  *   as soon they will be approved by the European Commission (IDABC).
00015  *
00016  *   This project is free software: you can redistribute it and/or modify it under the terms of the
00017  *   EUPL license as
00018  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
00019  *
00020  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
00021  *   or agreed to in
00022  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
00023  *   without even the
00024  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
00025  *   check specific
00026  *   language governing permissions and limitations and more details.
00027  *
00028  *   You should use this project in compliance with the EUPL license. You should have received a copy
00029  *   of the license
00030  *
00031  */

```

```

00022 *   along with this project. If not, see the license at < https://eupl.eu/ >.
00023 *
00023 *****/
00024
00033 //
00034 #pragma once
00035 //
00036
00037 // C++ INCLUDES
00038 //
00039 #include <future>
00040 #include <map>
00041 //
00042
00043 // ZMQUTILS INCLUDES
00044 //
00045 #include "LibZMQUtils/libzmqutils_global.h"
00046 #include "LibZMQUtils/CommandServerClient/common.h"
00047 //
00048
00049 // ZMQ DECLARATIONS
00050 //
00051 namespace zmq
00052 {
00053     class context_t;
00054     class socket_t;
00055 }
00056 //
00057
00058 // ZMQUTILS NAMESPACES
00059 //
00060 namespace zmqutils{
00061 //
00062
00063 //
00064 using common::ServerCommand;
00065 using common::ServerResult;
00066 using common::ClientResult;
00067 using common::CommandReply;
00068 using common::CommandType;
00069 using common::RequestData;
00070 //
00071
00072 class LIBZMQUTILS_EXPORT CommandClientBase
00073 {
00074 public:
00075     CommandClientBase(const std::string &server_endpoint);
00076     virtual ~CommandClientBase();
00077     bool startClient(const std::string& interface_name);
00078     void stopClient();
00079     void resetClient();
00080     void startAutoAlive();
00081     void stopAutoAlive();
00082     void setClientHostIP(const std::string& interf);
00083     void setClientId(const std::string &id);
00084     ClientResult sendCommand(const RequestData&, CommandReply&);
00085 protected:
00086     virtual void onSendCommand(const RequestData&, const zmq::multipart_t&) = 0;
00087 private:
00088     ClientResult recvFromSocket(CommandReply&);
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102

```



```

00103     void sendAliveCallback();
00104     zmq::multipart_t prepareMessage(const RequestData &msg);
00105
00106     // Internal client identification.
00107     common::HostClient client_info_;
00108
00109     // Server endpoint.
00110     std::string server_endpoint_;
00111
00112     // ZMQ context and socket.
00113     zmq::context_t *context_;
00114     zmq::socket_t *client_socket_;
00115
00116     // Mutex.
00117     std::mutex mtx_;
00118
00119     std::future<void> auto_alive_future_;
00120     std::condition_variable auto_alive_cv_;
00121     std::atomic_bool auto_alive_working_;
00122 };
00123 };
00124
00125 } // END NAMESPACES.
00126 //
=====

```

7.33 includes/LibZMQUtils/CommandServerClient/command_server.h File Reference

This file contains the declaration of the CommandServerBase class and related.

```

#include <future>
#include <map>
#include <zmq/zmq.hpp>
#include <zmq/zmq_addon.hpp>
#include "LibZMQUtils/libzmqutils_global.h"
#include "LibZMQUtils/CommandServerClient/common.h"
#include "LibZMQUtils/utils.h"

```

Classes

- class [zmqutils::CommandServerBase](#)

This class provides the base structure for a ZeroMQ based command server.

Namespaces

- namespace [zmq](#)
- namespace [zmqutils](#)

7.33.1 Detailed Description

This file contains the declaration of the CommandServerBase class and related.

Author

Degoras Project Team

Copyright

EUPL License

Version

2307.1

Definition in file [command_server.h](#).

7.34 command_server.h

[Go to the documentation of this file.](#)

```

00001  /*****
00002  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
00003  *
00004  *   Copyright (C) 2023 Degoras Project Team
00005  *
00006  *   < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
00007  *
00008  *   < Jesús Relinque Madroñal >
00009  *
00010  *   This file is part of LibZMQUtils.
00011  *
00012  *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
00013  *   the EUPL license
00014  *   as soon they will be approved by the European Commission (IDABC).
00015  *
00016  *   This project is free software: you can redistribute it and/or modify it under the terms of the
00017  *   EUPL license as
00018  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
00019  *
00020  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
00021  *   or agreed to in
00022  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
00023  *   without even the
00024  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
00025  *   check specific
00026  *   language governing permissions and limitations and more details.
00027  *
00028  *   You should use this project in compliance with the EUPL license. You should have received a copy
00029  *   of the license
00030  *   along with this project. If not, see the license at < https://eupl.eu/ >.
00031  *
00032  *****/
00033  //
00034  #pragma once
00035  //
00036  // C++ INCLUDES
00037  //
00038  #include <future>
00039  #include <map>
00040  #include <zmq/zmq.hpp>
00041  #include <zmq/zmq_addon.hpp>
00042  //
00043  // ZMQUTILS INCLUDES
00044  //
00045  #include "LibZMQUtils/libzmqutils_global.h"
00046  #include "LibZMQUtils/CommandServerClient/common.h"
00047  #include "LibZMQUtils/Utils.h"
00048  //
00049  // ZMQ DECLARATIONS
00050  //
00051  namespace zmq
00052  {
00053      class context_t;
00054      class socket_t;
00055  }
00056  //
00057  *****/
00058  
```

```

00061 // ZMQUTILS NAMESPACE
00062 //
00063 namespace zmqutils{
00064 //
00065 //
00066 //
00067 using common::ServerResultStr;
00068 using common::CommandReply;
00069 using common::CommandRequest;
00070 using common::ServerCommand;
00071 using common::ServerResult;
00072 using common::HostClient;
00073 using utils::NetworkAdapterInfo;
00074 //
00075
00141 class LIBZMQUTILS_EXPORT CommandServerBase
00142 {
00143 public:
00145     CommandServerBase(unsigned port, const std::string &local_addr = "");
00167     const unsigned& getServerPort() const;
00173     const std::vector<NetworkAdapterInfo> &getServerAddresses() const;
00183     const std::string& getServerEndpoint() const;
00193     const std::future<void>& getServerWorkerFuture() const;
00204     const std::map<std::string, HostClient>& getConnectedClients() const;
00215     bool isWorking() const{return this->server_working_;}
00226     void setClientStatusCheck(bool);
00240     void startServer();
00248     void stopServer();
00256     virtual ~CommandServerBase();
00263 protected:
00266     virtual void onServerStop() = 0;
00276     virtual void onServerStart() = 0;
00287     virtual void onWaitingCommand() = 0;
00302     virtual void onConnected(const HostClient&) = 0;
00315     virtual void onDisconnected(const HostClient&) = 0;
00328     virtual void onDeadClient(const HostClient&) = 0;
00341     virtual void onInvalidMsgReceived(const CommandRequest&) = 0;
00354     virtual void onCommandReceived(const CommandRequest&) = 0;
00370     virtual void onCustomCommandReceived(const CommandRequest&, CommandReply&);
00388     virtual void onServerError(const zmq::error_t &error, const std::string& ext_info = "") = 0;
00409     virtual void onSendingResponse(const CommandReply&) = 0;
00422 private:
00425     // Helper for prepare the result message.
00426     static void prepareCommandResult(ServerResult, std::unique_ptr<uint8_t>& data_out);
00428     // Helper for check if the base command is valid.
00429     static bool validateCommand(int raw_command);
00430     // Server worker. Will be execute asynchronously.
00431     void serverWorker();
00433     // Process command class.
00434     void processCommand(const CommandRequest&, CommandReply&);
00436     // Client status checker.
00437     void checkClientsAliveStatus();
00438
00440

```

```

00441 // Update client last connection.
00442 void updateClientLastConnection(const std::string& id);
00443
00444 // Update the server timeout.
00445 void updateServerTimeout();
00446
00447 // Internal connect execution process.
00448 ServerResult execReqConnect(const CommandRequest&);
00449
00450 // Internal disconnect execution process.
00451 ServerResult execReqDisconnect(const CommandRequest&);
00452
00453 // Function for receive data from the client.
00454 ServerResult recvFromSocket(CommandRequest&);
00455
00456 // Function for reset the socket.
00457 void resetSocket();
00458
00459 // ZMQ socket and context.
00460 zmq::context_t *context_;
00461 zmq::socket_t* server_socket_;
00462
00463 // Endpoint data.
00464 std::string server_endpoint_;
00465 std::vector<utils::NetworkAdapterInfo> server_listen_adapters_;
00466 unsigned server_port_;
00467
00468 // Mutex.
00469 std::mutex mtx_;
00470
00471 // Future for the server worker.
00472 std::future<void> server_worker_future_;
00473
00474 // Clients container.
00475 std::map<std::string, HostClient> connected_clients_;
00476
00477 // Usefull flags.
00478 std::atomic_bool server_working_;
00479 std::atomic_bool check_clients_alive_;
00480 };
00481
00482 } // END NAMESPACES.
00483 //
=====

```

7.35 includes/LibZMQUtils/libzmqutils_global.h File Reference

Macros

- `#define LIBZMQUTILS_EXPORT`

7.35.1 Macro Definition Documentation

7.35.1.1 LIBZMQUTILS_EXPORT

`#define LIBZMQUTILS_EXPORT`

Definition at line 38 of file [libzmqutils_global.h](#).

7.36 libzmqutils_global.h

[Go to the documentation of this file.](#)

```

00001
00002 /*****
00003  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
00004  *
00005  *   Copyright (C) 2023 Degoras Project Team
00006  *
00007  *   < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
00008  *
00009  *   < Jesús Relinque Madroñal >
00010  *
00011  *   This file is part of LibZMQUtils.
00012  *
00013  *****/

```

```

00009  *
00010  *   Licensed under the European Union Public License (EURL), Version 1.2 or subsequent versions of
the EURL license  *
00011  *   as soon they will be approved by the European Commission (IDABC).
00012  *
00013  *   This project is free software: you can redistribute it and/or modify it under the terms of the
EURL license as  *
00014  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
00015  *
00016  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
or agreed to in *
00017  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
without even the *
00018  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EURL license to
check specific  *
00019  *   language governing permissions and limitations and more details.
00020  *
00021  *   You should use this project in compliance with the EURL license. You should have received a copy
of the license  *
00022  *   along with this project. If not, see the license at < https://eupl.eu/ >.
00023  *
00024  *
00025  //
=====
00026 #pragma once
00027 //
=====
00028
00029 //
=====
00030 #if ((defined __WIN32__) || (defined _WIN32)) && (!defined LIBZMQUTILS_STATIC)
00031 #ifndef LIBZMQUTILS_LIBRARY
00032 #define LIBZMQUTILS_EXPORT __declspec(dllexport)
00033 #else
00034 #define LIBZMQUTILS_EXPORT __declspec(dllimport)
00035 #endif
00036 #else
00037 /* Static libraries or non-Windows needs no special declaration. */
00038 # define LIBZMQUTILS_EXPORT
00039 #endif
00040 //
=====

```

7.37 sources/CommandServerClient/command_client.cpp File Reference

```

#include <zmq/zmq.hpp>
#include <zmq/zmq_addon.hpp>
#include <iostream>
#include <string>
#include <cstring>
#include <cstdlib>
#include <algorithm>
#include "LibZMQUtils/CommandServerClient/command_client.h"
#include "LibZMQUtils/Utils.h"

```

Namespaces

- namespace [zmqutils](#)

7.38 command_client.cpp

[Go to the documentation of this file.](#)

```

00001
00002 #include <zmq/zmq.hpp>
00003 #include <zmq/zmq_addon.hpp>

```

```

00004
00005 #include <iostream>
00006
00007 #include <iostream>
00008 #include <string>
00009 #include <cstring>
00010 #include <cstdlib>
00011 #include <algorithm>
00012
00013 #include "LibZMQUtils/CommandServerClient/command_client.h"
00014 #include "LibZMQUtils/Utils.h"
00015
00016 // ZMQUTILS NAMESPACE
00017 //
=====
00018 namespace zmqutils{
00019 //
=====
00020
00021
00022
00023
00024 CommandClientBase::CommandClientBase(const std::string &server_endpoint) :
00025     server_endpoint_(server_endpoint),
00026     context_(nullptr),
00027     client_socket_(nullptr),
00028     auto_alive_working_(false)
00029 {
00030
00031 }
00032
00033 CommandClientBase::~CommandClientBase()
00034 {
00035     if (this->auto_alive_working_)
00036         this->stopAutoAlive();
00037     this->stopClient();
00038 }
00039
00040 bool CommandClientBase::startClient(const std::string& interface_name)
00041 {
00042     // Auxiliar variables.
00043     std::string ip, name, pid;
00044
00045     // Get the client ip.
00046     std::vector<utils::NetworkAdapterInfo> interfcs = utils::getHostIPsWithInterfaces();
00047     auto it = std::find_if(interfcs.begin(), interfcs.end(), [&interface_name](const
utils::NetworkAdapterInfo& info)
00048     {return info.name == interface_name;});
00049     if (it == interfcs.end())
00050         return false;
00051     ip = it->ip;
00052
00053     // Get the host name.
00054     name = utils::getHostname();
00055
00056     // Get the current pid.
00057     pid = std::to_string(utils::getCurrentPID());
00058
00059     // Store the info.
00060     this->client_info_ = common::HostClient(ip, name, pid);
00061
00062     std::cout<<client_info_.id<<std::endl;
00063
00064     // If server is already started, do nothing
00065     if (this->client_socket_)
00066         return false;
00067
00068     // Create the ZMQ context.
00069     if (!this->context_)
00070         this->context_ = new zmq::context_t(1);
00071
00072     try
00073     {
00074         this->client_socket_ = new zmq::socket_t(*this->context_, zmq::socket_type::req);
00075         this->client_socket_->connect(this->server_endpoint_);
00076         // Set timeout so socket will not wait for answer more than client alive timeout.
00077         this->client_socket_->set(zmq::sockopt::rcvtimeo, common::kDefaultServerAliveTimeoutMsec);
00078         this->client_socket_->set(zmq::sockopt::linger, 0);
00079     }
00080     catch (const zmq::error_t &error)
00081     {
00082         delete this->client_socket_;
00083         this->client_socket_ = nullptr;
00084
00085         std::cerr << "Error at socket creation: " << error.num();
00086         // TODO: handle error
00087         return false;

```

```

00088     }
00089
00090     // All ok.
00091     return true;
00092 }
00093
00094 void CommandClientBase::stopClient()
00095 {
00096     // If server is already stopped, do nothing.
00097     if (!this->client_socket_)
00098         return;
00099
00100     // Destroy the socket.
00101     delete this->client_socket_;
00102     this->client_socket_ = nullptr;
00103
00104     std::this_thread::sleep_for(std::chrono::milliseconds(1050));
00105
00106     // Delete context
00107     if (this->context_)
00108     {
00109         delete this->context_;
00110         this->context_ = nullptr;
00111     }
00112 }
00113
00114 }
00115
00116 void CommandClientBase::resetClient()
00117 {
00118     if (this->client_socket_)
00119     {
00120         // Destroy the socket and create again to flush.
00121         delete this->client_socket_;
00122
00123         std::this_thread::sleep_for(std::chrono::milliseconds(1050));
00124
00125         try
00126         {
00127             this->client_socket_ = new zmq::socket_t(*this->context_, zmq::socket_type::req);
00128             this->client_socket_->connect(this->server_endpoint_);
00129             // Set timeout so socket will not wait for answer more than server alive timeout.
00130             this->client_socket_->set(zmq::sockopt::rcvtimeo, common::kDefaultServerAliveTimeoutMsec);
00131             this->client_socket_->set(zmq::sockopt::linger, 0);
00132         }
00133         catch (const zmq::error_t &error)
00134         {
00135             delete this->client_socket_;
00136             this->client_socket_ = nullptr;
00137
00138             std::cerr << "Error at socket creation: " << error.num();
00139             // TODO: handle error
00140         }
00141     }
00142 }
00143
00144 void CommandClientBase::startAutoAlive()
00145 {
00146     this->auto_alive_working_ = true;
00147     this->auto_alive_future_ = std::async(std::launch::async, [this]{this->sendAliveCallback();});
00148 }
00149
00150 void CommandClientBase::stopAutoAlive()
00151 {
00152     if (this->auto_alive_working_)
00153     {
00154         this->auto_alive_working_ = false;
00155         this->auto_alive_cv_.notify_all();
00156         this->auto_alive_future_.wait();
00157     }
00158 }
00159
00160 void CommandClientBase::setClientHostIP(const std::string&){}
00161
00162 void CommandClientBase::setClientId(const std::string &){}
00163
00164 ClientResult CommandClientBase::sendCommand(const RequestData& msg, CommandReply& reply)
00165 {
00166     // Result.
00167     ClientResult result;
00168
00169     // Check if we start the client.
00170     if (!this->client_socket_)
00171         return ClientResult::CLIENT_STOPPED;
00172
00173     // Send the command.
00174     try

```

```

00175     {
00176
00177         zmq::multipart_t multipart_msg(this->prepareMessage(msg));
00178
00179         // Internal send callback.
00180         this->onSendCommand(msg, multipart_msg);
00181
00182         // Send the multiple messages.
00183         multipart_msg.send(*this->client_socket_);
00184     } catch (const zmq::error_t &error)
00185     {
00186         // TODO: handle error
00187         std::cout<<error.what()<<std::endl;
00188         return ClientResult::INTERNAL_ZMQ_ERROR;
00189     }
00190
00191     std::cout<<"Waiting response"<<std::endl;
00192
00193     // TODO multipart.
00194
00195     result = this->recvFromSocket(reply);
00196
00197     if (this->auto_alive_working_)
00198         this->auto_alive_cv_.notify_one();
00199
00200     return result;
00201 }
00202
00203 ClientResult CommandClientBase::recvFromSocket(CommandReply& reply)
00204 {
00205     // Result variable.
00206     ClientResult result = ClientResult::COMMAND_OK;
00207
00208     // Containers.
00209     bool rcv_result;
00210     zmq::multipart_t multipart_msg;
00211
00212     // Try to receive data. If an exception is thrown, receiving fails and an error code is generated.
00213     try
00214     {
00215         // Call to the internal waiting command callback. TODO
00216         //this->onWaitingCommand();
00217
00218         // Wait the command.
00219         rcv_result = multipart_msg.recv(*this->client_socket_);
00220
00221         // Store the raw data.
00222         reply.raw_msg = multipart_msg.clone();
00223     }
00224     catch(zmq::error_t& error)
00225     {
00226         // Call to error callback. TODO
00227         //this->onServerError(error, "Error while receiving a request.");
00228
00229         std::cout<<"INTERNAL ERROR "<<error.what()<<std::endl;
00230
00231         return ClientResult::INTERNAL_ZMQ_ERROR;
00232     }
00233
00234     // Check for empty msg or timeout reached.
00235     if (multipart_msg.empty() && !rcv_result)
00236         return ClientResult::TIMEOUT_REACHED;
00237     else if (multipart_msg.empty())
00238         return ClientResult::EMPTY_MSG;
00239
00240     // Check the multipart msg size.
00241     if (multipart_msg.size() == 1 || multipart_msg.size() == 2)
00242     {
00243         // Get the multipart data.
00244         zmq::message_t message_result = multipart_msg.pop();
00245
00246         // Get the sizes.
00247         size_t result_size_bytes = message_result.size();
00248
00249         // Get the command.
00250         if (result_size_bytes == sizeof(ServerCommand))
00251         {
00252             int raw_result;
00253             utils::binarySerializeDeserialize(message_result.data(), sizeof(ServerCommand),
00254             &raw_result);

```



```

00261         reply.result = static_cast<common::ServerResult>(raw_result);
00262     }
00263     else
00264         return ClientResult::INVALID_MSG;
00265
00266     // If there is still one more part, they are the parameters.
00267     if (multipart_msg.size() == 1)
00268     {
00269         // Get the message and the size.
00270         zmq::message_t message_params = multipart_msg.pop();
00271         size_t params_size_bytes = message_params.size();
00272
00273         std::cout<<multipart_msg.str()<<std::endl;
00274         std::cout<<params_size_bytes<<std::endl;
00275
00276         // Check the parameters.
00277         if (params_size_bytes > 0)
00278         {
00279             // Get and store the parameters data.
00280             std::unique_ptr<std::uint8_t> params =
00281                 std::unique_ptr<std::uint8_t>(new std::uint8_t[params_size_bytes]);
00282             auto *params_pointer = static_cast<std::uint8_t*>(message_params.data());
00283             std::copy(params_pointer, params_pointer + params_size_bytes, params.get());
00284             reply.params = std::move(params);
00285             reply.params_size = params_size_bytes;
00286         }
00287         else
00288             return ClientResult::EMPTY_PARAMS;
00289     }
00290     else
00291         return ClientResult::INVALID_PARTS;
00292
00293     // Return the result.
00294     return result;
00295 }
00296
00297 void CommandClientBase::sendAliveCallback()
00298 {
00299     /*
00300     std::mutex m;
00301     std::unique_lock<std::mutex> lk(m);
00302     bool send_success = true;
00303     bool rcv_success = true;
00304     void *data_out;
00305     size_t out_size;
00306     zmq::multipart_t msg;
00307     zmq::socket_t *alive_socket = new zmq::socket_t(*this->context_, zmq::socket_type::req);
00308     alive_socket->connect(this->server_endpoint_);
00309     // Set timeout so socket will not wait for answer more than client alive timeout.
00310     alive_socket->set(zmq::sockopt::rcvtimeo, common::kDefaultServerAliveTimeoutMsec);
00311     alive_socket->set(zmq::sockopt::linger, 0);
00312
00313     while(this->auto_alive_working_)
00314     {
00315         auto res =
00316             this->auto_alive_cv_.wait_for(lk,
00317             std::chrono::milliseconds(common::kClientAlivePeriodMsec));
00318
00319         if (std::cv_status::timeout == res)
00320         {
00321             msg = this->prepareMessage(
00322                 RequestData(static_cast<common::CommandType>(ServerCommand::REQ_ALIVE)));
00323             try
00324             {
00325                 msg.send(*alive_socket);
00326             } catch (const zmq::error_t &error)
00327             {
00328                 // TODO: handle error
00329                 std::cerr << "Failed to automatically send alive command with error: " << error.num() <<
00330                 std::endl;
00331                 send_success = false;
00332             }
00333             if (send_success)
00334             {
00335                 auto rcv_result = this->recvFromSocket(alive_socket, data_out, out_size);
00336                 auto *data_bytes = static_cast<std::uint8_t*>(data_out);
00337
00338                 if (0 == rcv_result && out_size == sizeof(common::ServerResult))
00339                 {
00340                     common::ServerResult result;
00341
00342                     zmqutils::utils::binarySerializeDeserialize(
00343                         data_bytes, sizeof(common::CommandReply), &result);
00344                 }
00345             }
00346         }
00347     }

```

```

00346         recv_success = result == common::ServerResult::COMMAND_OK;
00347     }
00348     }
00349     else
00350     {
00351         std::cerr << "Auto alive message answer receive failed" << std::endl;
00352         recv_success = false;
00353     }
00354     delete[] data_bytes;
00355 }
00356 }
00357
00358 if (!send_success || !recv_success)
00359 {
00360     std::cerr << "Failed auto sending alive message. Process will be stopped." << std::endl;
00361     this->auto_alive_working_ = false;
00362 }
00363 }
00364 }
00365 }
00366 }
00367
00368 delete alive_socket;
00369 */
00370 }
00371
00372 zmq::multipart_t CommandClientBase::prepareMessage(const RequestData &msg)
00373 {
00374     // Prepare the ip data.
00375     zmq::message_t message_ip(this->client_info_.ip.begin(), this->client_info_.ip.end());
00376     // Prepare the hostname data.
00377     zmq::message_t message_host(this->client_info_.hostname.begin(),
00378         this->client_info_.hostname.end());
00379     // Prepare the pid data.
00380     zmq::message_t message_pid(this->client_info_.pid.begin(), this->client_info_.pid.end());
00381     // Prepare the command data.
00382     std::uint8_t command_buffer[sizeof(common::CommandType)];
00383     zmqutils::utils::binarySerializeDeserialize(&msg.command, sizeof(common::CommandType),
00384         command_buffer);
00385     zmq::message_t message_command(&command_buffer, sizeof(common::CommandType));
00386
00387     // Prepare the multipart msg.
00388     zmq::multipart_t multipart_msg;
00389     multipart_msg.add(std::move(message_ip));
00390     multipart_msg.add(std::move(message_host));
00391     multipart_msg.add(std::move(message_pid));
00392     multipart_msg.add(std::move(message_command));
00393
00394     // Add command parameters if they exist
00395     if (msg.params_size > 0)
00396     {
00397         // Prepare the command parameters
00398         zmq::message_t message_params(msg.params.get(), msg.params_size);
00399         multipart_msg.add(std::move(message_params));
00400     }
00401     return multipart_msg;
00402 }
00403
00404 } // END NAMESPACES.
00405 //
=====

```

7.39 sources/CommandServerClient/command_server.cpp File Reference

This file contains the implementation of the CommandServerBase class and related.

```

#include <iostream>
#include <stdio.h>
#include <zmq/zmq_addon.hpp>
#include <zmq/zmq.h>
#include "LibZMQUtils/CommandServerClient/command_server.h"
#include "LibZMQUtils/utils.h"

```

Namespaces

- namespace [zmqutils](#)

7.39.1 Detailed Description

This file contains the implementation of the `CommandServerBase` class and related.

Author

Degoras Project Team

Copyright

EUPL License

Version

2307.1

Definition in file [command_server.cpp](#).

7.40 command_server.cpp

[Go to the documentation of this file.](#)

```
00001 /*****
00002  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
00003  *
00004  *   Copyright (C) 2023 Degoras Project Team
00005  *
00006  *   < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
00007  *
00008  *   < Jesús Relinque Madroñal >
00009  *
00010  *   This file is part of LibZMQUtils.
00011  *
00012  *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
00013  *   the EUPL license
00014  *   as soon they will be approved by the European Commission (IDABC).
00015  *
00016  *   This project is free software: you can redistribute it and/or modify it under the terms of the
00017  *   EUPL license as
00018  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
00019  *
00020  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
00021  *   or agreed to in
00022  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
00023  *   without even the
00024  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
00025  *   check specific
00026  *   language governing permissions and limitations and more details.
00027  *
00028  *   You should use this project in compliance with the EUPL license. You should have received a copy
00029  *   of the license
00030  *   along with this project. If not, see the license at < https://eupl.eu/ >.
00031  *
00032  *****/
00033 // C++ INCLUDES
00034 //
00035 #include <iostream>
00036 #include <stdio.h>
```

```

00037 #include <zmq/zmq_addon.hpp>
00038 #include <zmq/zmq.h>
00039 //
=====
00040
00041 // ZMQUTILS INCLUDES
00042 //
=====
00043 #include "LibZMQUtils/CommandServerClient/command_server.h"
00044 #include "LibZMQUtils/Utils.h"
00045 //
=====
00046
00047 // ZMQUTILS NAMESPACES
00048 //
=====
00049 namespace zmqutils{
00050 //
=====
00051
00052 CommandServerBase::CommandServerBase(unsigned int port, const std::string& local_addr) :
00053     context_(nullptr),
00054     server_socket_(nullptr),
00055     server_endpoint_("tcp://" + local_addr + ":" + std::to_string(port)),
00056     server_port_(port),
00057     server_working_(false),
00058     check_clients_alive_(true)
00059 {
00060     // Get the adapters.
00061     std::vector<utils::NetworkAdapterInfo> interfcs = utils::getHostIPsWithInterfaces();
00062     // Store the adapters.
00063     if(local_addr == "*")
00064         this->server_listen_adapters_ = interfcs;
00065     else
00066     {
00067         for(const auto& intrfc : interfcs)
00068         {
00069             if(intrfc.ip == local_addr)
00070                 this->server_listen_adapters_.push_back(intrfc);
00071         }
00072     }
00073 }
00074
00075 const std::future<void> &CommandServerBase::getServerWorkerFuture() const {return
    this->server_worker_future_;}
00076
00077 const std::map<std::string, HostClient> &CommandServerBase::getConnectedClients() const
00078 {return this->connected_clients_;}
00079
00080 void CommandServerBase::setClientStatusCheck(bool)
00081 {
00082     // Safe mutex lock
00083     std::unique_lock<std::mutex> lock(this->mtx_);
00084     // Disable the client alive checking.
00085     this->check_clients_alive_ = false;
00086     if(this->server_socket_)
00087         this->server_socket_->set(zmq::sockopt::rcvtimeo, -1);
00088 }
00089
00090 const unsigned& CommandServerBase::getServerPort() const {return this->server_port_;}
00091
00092 const std::vector<utils::NetworkAdapterInfo>& CommandServerBase::getServerAddresses() const
00093 {return this->server_listen_adapters_;}
00094
00095 const std::string& CommandServerBase::getServerEndpoint() const {return this->server_endpoint_;}
00096
00097 void CommandServerBase::startServer()
00098 {
00099     // Safe mutex lock
00100     std::unique_lock<std::mutex> lock(this->mtx_);
00101
00102     // If server is already started, do nothing
00103     if (this->server_working_)
00104         return;
00105
00106     // Create the ZMQ context.
00107     if (!this->context_)
00108         this->context_ = new zmq::context_t(1);
00109
00110     // Launch server worker in other thread.
00111     this->server_worker_future_ = std::async(std::launch::async, &CommandServerBase::serverWorker,
    this);
00112 }
00113
00114 void CommandServerBase::stopServer()
00115 {
00116     // Safe mutex lock

```

```

00117     std::unique_lock<std::mutex> lock(this->mtx_);
00118
00119     // If server is already stopped, do nothing.
00120     if (!this->server_working_)
00121         return;
00122
00123     // Set the shared working flag to false (is atomic).
00124     this->server_working_ = false;
00125
00126     // Delete the context.
00127     if (this->context_)
00128     {
00129         delete this->context_;
00130         context_ = nullptr;
00131     }
00132
00133     // Clean the clients.
00134     this->connected_clients_.clear();
00135 }
00136
00137 CommandServerBase::~CommandServerBase()
00138 {
00139     // Stop the server (this function also deletes the pointers).
00140     this->stopServer();
00141 }
00142
00143 ServerResult CommandServerBase::execReqConnect(const CommandRequest& cmd_req)
00144 {
00145     // Safe mutex lock.
00146     std::unique_lock<std::mutex> lock(this->mtx_);
00147
00148     // Check if the client is already connected.
00149     auto it = this->connected_clients_.find(cmd_req.client.id);
00150     if(it != this->connected_clients_.end())
00151         return ServerResult::ALREADY_CONNECTED;
00152
00153     // Add the new client.
00154     this->connected_clients_[cmd_req.client.id] = cmd_req.client;
00155
00156     // Update the timeout of the main socket.
00157     if(this->check_clients_alive_)
00158         this->updateServerTimeout();
00159
00160     // Call to the internal callback.
00161     this->onConnected(cmd_req.client);
00162
00163     // All ok.
00164     return ServerResult::COMMAND_OK;
00165 }
00166
00167 ServerResult CommandServerBase::execReqDisconnect(const CommandRequest& cmd_req)
00168 {
00169     // Safe mutex lock.
00170     std::unique_lock<std::mutex> lock(this->mtx_);
00171
00172     // Get the client.
00173     auto it = this->connected_clients_.find(cmd_req.client.id);
00174
00175     // Remove the client from the map of connected clients.
00176     this->connected_clients_.erase(it);
00177
00178     // Call to the internal callback.
00179     this->onDisconnected(cmd_req.client);
00180
00181     // Update the timeout of the main socket.
00182     if(this->check_clients_alive_)
00183         this->updateServerTimeout();
00184
00185     // All ok.
00186     return ServerResult::COMMAND_OK;
00187 }
00188
00189 void CommandServerBase::serverWorker()
00190 {
00191     // Auxiliar variables.
00192     ServerResult result;
00193
00194     // Set the working flag to true.
00195     this->server_working_ = true;
00196
00197     // Start server socket
00198     this->resetSocket();
00199
00200     // Server worker loop.
00201     // If there is no client connected wait for a client to connect or for an exit message. If there
00202     // is a client connected set timeout, so if no command comes in time, check the last time
    connection

```

```

00203 // for each client. The loop can be stopped (in a safe way) if using the stopServer() function.
00204 while(this->server_socket_ && this->server_working_)
00205 {
00206     // Message container.
00207     CommandRequest cmd_request;
00208
00209     // Result container.
00210     CommandReply cmd_reply;
00211
00212     // Receive the data.
00213     result = this->recvFromSocket(cmd_request);
00214
00215     // Check all the clients status.
00216     if(this->check_clients_alive_)
00217         this->checkClientsAliveStatus();
00218
00219     // Process the data.
00220     if(result == ServerResult::COMMAND_OK && !this->server_working_)
00221     {
00222         // In this case, we will close the server. Call to the internal callback.
00223         this->onServerStop();
00224     }
00225     else if(result == ServerResult::TIMEOUT_REACHED)
00226     {
00227         // DO NOTHING.
00228     }
00229     else if (result != ServerResult::COMMAND_OK)
00230     {
00231         // Internal callback.
00232         this->onInvalidMsgReceived(cmd_request);
00233
00234         // Prepare the message.
00235         std::uint8_t res_buff[sizeof(ServerResult)];
00236         utils::binarySerializeDeserialize(&result, sizeof(ServerResult), res_buff);
00237         zmq::message_t message_res(res_buff, sizeof(ServerResult));
00238
00239         // Send response callback.
00240         cmd_reply.result = result;
00241         this->onSendingResponse(cmd_reply);
00242
00243         // Send the response.
00244         try
00245         {
00246             this->server_socket_->send(message_res, zmq::send_flags::none);
00247         }
00248         catch (const zmq::error_t &error)
00249         {
00250             // Check if we want to close the server.
00251             // The error code is for ZMQ EFSM error.
00252             if(! (error.num() == common::kZmqEFSMError && !this->server_working_))
00253                 this->onServerError(error, "Error while sending a response.");
00254         }
00255     }
00256     else if (result == ServerResult::COMMAND_OK)
00257     {
00258         // Reply id buffer.
00259         std::unique_ptr<std::uint8_t> rep_id_buff;
00260
00261         // Execute the command.
00262         this->processCommand(cmd_request, cmd_reply);
00263
00264         // Prepare the command result.
00265         CommandServerBase::prepareCommandResult(cmd_reply.result, rep_id_buff);
00266         zmq::message_t message_rep_id(rep_id_buff.get(), sizeof(ServerResult));
00267
00268         // Prepare the multipart msg.
00269         zmq::multipart_t multipart_msg;
00270         multipart_msg.add(std::move(message_rep_id));
00271
00272         // Specific data.
00273         if(cmd_reply.result == ServerResult::COMMAND_OK && cmd_reply.params_size != 0)
00274         {
00275             // Prepare the custom response.
00276             zmq::message_t message_rep_custom(cmd_reply.params.get(), cmd_reply.params_size);
00277             multipart_msg.add(std::move(message_rep_custom));
00278         }
00279
00280         // Sending callback.
00281         this->onSendingResponse(cmd_reply);
00282
00283         // Send the message.
00284         try
00285         {
00286             multipart_msg.send(*this->server_socket_);
00287         }
00288         catch (const zmq::error_t &error)
00289         {

```

```

00290         // Check if we want to close the server.
00291         // The error code is for ZMQ EFSM error.
00292         if(!error.num() == common::kZmqEFSMError && !this->server_working_)
00293             this->onServerError(error, "Error while sending a response.");
00294     }
00295 }
00296 }
00297
00298 // Delete pointers for clean finish the worker.
00299 if (this->server_socket_)
00300 {
00301     delete this->server_socket_;
00302     this->server_socket_ = nullptr;
00303 }
00304 }
00305
00306 ServerResult CommandServerBase::recvFromSocket(CommandRequest& request)
00307 {
00308     // Result variable.
00309     ServerResult result = ServerResult::COMMAND_OK;
00310
00311     // Containers.
00312     bool rcv_result;
00313     zmq::multipart_t multipart_msg;
00314
00315     // Try to receive data. If an exception is thrown, receiving fails and an error code is generated.
00316     try
00317     {
00318         // Call to the internal waiting command callback.
00319         this->onWaitingCommand();
00320
00321         // Wait the command.
00322         rcv_result = multipart_msg.recv(*(this->server_socket_));
00323
00324         // Store the raw data.
00325         request.raw_msg = multipart_msg.clone();
00326     }
00327     catch(zmq::error_t& error)
00328     {
00329         // Check if we want to close the server.
00330         // The error code is for ZMQ EFSM error.
00331         if(error.num() == common::kZmqEFSMError && !this->server_working_)
00332             return ServerResult::COMMAND_OK;
00333
00334         // Else, call to error callback.
00335         this->onServerError(error, "Error while receiving a request.");
00336         return ServerResult::INTERNAL_ZMQ_ERROR;
00337     }
00338
00339     // Check for empty msg or timeout reached.
00340     if (multipart_msg.empty() && !rcv_result)
00341         return ServerResult::TIMEOUT_REACHED;
00342     else if (multipart_msg.empty())
00343         return ServerResult::EMPTY_MSG;
00344
00345     // Check the multipart msg size.
00346     if (multipart_msg.size() == 4 || multipart_msg.size() == 5)
00347     {
00348         // Auxiliar containers.
00349         std::string ip;
00350         std::string hostname;
00351         std::string pid;
00352
00353         // Get the multipart data.
00354         zmq::message_t message_ip = multipart_msg.pop();
00355         zmq::message_t message_hostname = multipart_msg.pop();
00356         zmq::message_t message_pid = multipart_msg.pop();
00357         zmq::message_t message_command = multipart_msg.pop();
00358
00359         // Get the sizes.
00360         size_t ip_size_bytes = message_ip.size();
00361         size_t host_size_bytes = message_hostname.size();
00362         size_t pid_size_bytes = message_pid.size();
00363         size_t command_size_bytes = message_command.size();
00364
00365         // First get the ip data.
00366         if (ip_size_bytes > 0)
00367             ip = std::string(static_cast<char*>(message_ip.data()), ip_size_bytes);
00368         else
00369             return ServerResult::EMPTY_CLIENT_IP;
00370
00371         // Get the hostname data.
00372         if (host_size_bytes > 0)
00373             hostname = std::string(static_cast<char*>(message_hostname.data()), host_size_bytes);
00374         else
00375             return ServerResult::EMPTY_CLIENT_NAME;
00376

```

```

00377         // Get the pid data.
00378         if (host_size_bytes > 0)
00379             pid = std::string(static_cast<char*>(message_pid.data()), pid_size_bytes);
00380         else
00381             return ServerResult::EMPTY_CLIENT_PID;
00382
00383         // Update the client info.
00384         request.client = HostClient(ip, hostname, pid);
00385         request.client.last_connection = std::chrono::steady_clock::now();
00386
00387         // Update the last connection if the client is connected.
00388         this->updateClientLastConnection(request.client.id);
00389
00390         // Get the command.
00391         if (command_size_bytes == sizeof(ServerCommand))
00392         {
00393             int raw_command;
00394             utils::binarySerializeDeserialize(message_command.data(), sizeof(ServerCommand),
&raw_command);
00395             // Validate the command.
00396             if (CommandServerBase::validateCommand(raw_command))
00397                 request.command = static_cast<ServerCommand>(raw_command);
00398             else
00399             {
00400                 request.command = ServerCommand::INVALID_COMMAND;
00401                 return ServerResult::INVALID_MSG;
00402             }
00403         }
00404         else
00405         {
00406             request.command = ServerCommand::INVALID_COMMAND;
00407             return ServerResult::INVALID_MSG;
00408         }
00409
00410         // If there is still one more part, they are the parameters.
00411         if (multipart_msg.size() == 1)
00412         {
00413             // Get the message and the size.
00414             zmq::message_t message_params = multipart_msg.pop();
00415             size_t params_size_bytes = message_params.size();
00416
00417             // Check the parameters.
00418             if (params_size_bytes > 0)
00419             {
00420                 // Get and store the parameters data.
00421                 std::unique_ptr<std::uint8_t> params =
00422                     std::unique_ptr<std::uint8_t>(new std::uint8_t[params_size_bytes]);
00423                 auto *params_pointer = static_cast<std::uint8_t*>(message_params.data());
00424                 std::copy(params_pointer, params_pointer + params_size_bytes, params.get());
00425                 request.params = std::move(params);
00426                 request.params_size = params_size_bytes;
00427             }
00428             else
00429                 return ServerResult::EMPTY_PARAMS;
00430         }
00431     }
00432     else
00433         return ServerResult::INVALID_PARTS;
00434
00435     // Return the result.
00436     return result;
00437 }
00438
00439 void CommandServerBase::prepareCommandResult(ServerResult result, std::unique_ptr<std::uint8_t>&
data_out)
00440 {
00441     data_out = std::unique_ptr<std::uint8_t>(new std::uint8_t[sizeof(ServerResult)]);
00442     utils::binarySerializeDeserialize(&result, sizeof(ServerResult), data_out.get());
00443 }
00444
00445 bool CommandServerBase::validateCommand(int raw_command)
00446 {
00447     // Auxiliar variables.
00448     bool result = false;
00449     int reserved_cmd = static_cast<int>(common::ServerCommand::RESERVED_COMMANDS);
00450     int end_base_cmd = static_cast<int>(common::ServerCommand::END_BASE_COMMANDS);
00451     // Check if the command is valid.
00452     if (raw_command >= common::kMinBaseCmdId && raw_command < reserved_cmd)
00453         result = true;
00454     else if (raw_command > end_base_cmd)
00455         result = true;
00456     return result;
00457 }
00458
00459 void CommandServerBase::processCommand(const CommandRequest& request, CommandReply& reply)
00460 {
00461     // First of all, call to the internal callback.

```



```

00462     this->onCommandReceived(request);
00463
00464     // Process the different commands.
00465     // 1 - Process is the connect request.
00466     // 2 - If the command is other, check if the client is connected to the server.
00467     // 3 - If it is, check if the command is valid.
00468     // 4 - If valid, process the rest of the base commands or the custom command.
00469     if (ServerCommand::REQ_CONNECT == request.command)
00470     {
00471         reply.result = this->execReqConnect(request);
00472     }
00473     else if(this->connected_clients_.find(request.client.id) == this->connected_clients_.end())
00474     {
00475         reply.result = ServerResult::CLIENT_NOT_CONNECTED;
00476     }
00477     else if (ServerCommand::REQ_DISCONNECT == request.command)
00478     {
00479         reply.result = this->execReqDisconnect(request);
00480     }
00481     else if (ServerCommand::REQ_ALIVE == request.command)
00482     {
00483         reply.result = ServerResult::COMMAND_OK;
00484     }
00485     else
00486     {
00487         // Custom command, so call the internal callback.
00488         this->onCustomCommandReceived(request, reply);
00489
00490         // Chek for an invalid msg.
00491         if(reply.result == ServerResult::INVALID_MSG)
00492             this->onInvalidMsgReceived(request);
00493     }
00494 }
00495
00496 void CommandServerBase::checkClientsAliveStatus()
00497 {
00498     // Safe mutex lock
00499     std::unique_lock<std::mutex> lock(this->mtx_);
00500
00501     // Auxiliar containers.
00502     std::vector<std::string> dead_clients;
00503     std::chrono::milliseconds timeout(common::kDefaultClientAliveTimeoutMsec);
00504     std::chrono::milliseconds min_remaining_time = timeout;
00505
00506     // Get the current time.
00507     utils::SCTimePointStd now = std::chrono::steady_clock::now();
00508
00509     // Check each connection.
00510     for(auto& client : this->connected_clients_)
00511     {
00512         // Get the last connection time.
00513         const auto& last_conn = client.second.last_connection;
00514         // Check if the client reaches the timeout checking the last connection time.
00515         auto since_last_conn = std::chrono::duration_cast<std::chrono::milliseconds>(now - last_conn);
00516         if(since_last_conn >= timeout)
00517         {
00518             // If dead, call the onDead callback and quit the client from the map.
00519             this->onDeadClient(client.second);
00520             dead_clients.push_back(client.first);
00521         }
00522         else
00523         {
00524             // If the client is not dead, check the minor timeout of the client to set
00525             // with the remain time to reach the timeout.
00526             min_remaining_time = std::min(min_remaining_time, timeout - since_last_conn);
00527         }
00528     }
00529
00530     // Remove dead clients from the map.
00531     for(auto& client : dead_clients)
00532     {
00533         this->connected_clients_.erase(client);
00534     }
00535
00536     // Disable the timeout if no clients remains or set the socket timeout to the
00537     // minimum remaining time to the timeout among all clients.
00538     if(this->connected_clients_.empty())
00539     {
00540         this->server_socket_->set(zmq::sockopt::rcvtimeo, -1);
00541     }
00542     else
00543     {
00544         this->server_socket_->set(zmq::sockopt::rcvtimeo,
00545             static_cast<int>(min_remaining_time.count()));
00546     }
00547 }

```

```

00548 void CommandServerBase::updateClientLastConnection(const std::string &id)
00549 {
00550     // Safe mutex lock.
00551     std::unique_lock<std::mutex> lock(this->mtx_);
00552     // Update the client last connection.
00553     auto client_itr = this->connected_clients_.find(id);
00554     if(client_itr != this->connected_clients_.end())
00555         client_itr->second.last_connection = std::chrono::steady_clock::now();
00556 }
00557
00558 void CommandServerBase::updateServerTimeout()
00559 {
00560     // Calculate the minor timeout to set it into the socket.
00561     auto min_timeout = std::min_element(this->connected_clients_.begin(),
    this->connected_clients_.end(),
00562         [](const auto& a, const auto& b)
00563         {
00564             auto diff_a = std::chrono::duration_cast<std::chrono::milliseconds>(
00565                 std::chrono::steady_clock::now() - a.second.last_connection);
00566             auto diff_b = std::chrono::duration_cast<std::chrono::milliseconds>(
00567                 std::chrono::steady_clock::now() - b.second.last_connection);
00568             return diff_a.count() < diff_b.count();
00569         });
00570
00571     if (min_timeout != this->connected_clients_.end())
00572     {
00573         auto remain_time = common::kDefaultClientAliveTimeoutMsec -
00574             std::chrono::duration_cast<std::chrono::milliseconds>(
00575                 std::chrono::steady_clock::now() -
00576                 min_timeout->second.last_connection).count();
00577         this->server_socket_->set(zmq::sockopt::rcvtimeo, std::max(0, static_cast<int>(remain_time)));
00578     }
00579     else
00580     {
00581         this->server_socket_->set(zmq::sockopt::rcvtimeo, -1);
00582     }
00583 }
00584 void CommandServerBase::resetSocket()
00585 {
00586     // Auxiliar variables.
00587     int res = 0;
00588     const zmq::error_t* last_error;
00589     unsigned reconnect_count = common::kServerReconnTimes;
00590
00591     // Delete the previous socket.
00592     if (this->server_socket_)
00593     {
00594         delete this->server_socket_;
00595         this->server_socket_ = nullptr;
00596     }
00597     // Try creating a new socket.
00598     do
00599     {
00600         try
00601         {
00602             // Create the ZMQ rep socket.
00603             std::this_thread::sleep_for(std::chrono::microseconds(500));
00604             this->server_socket_ = new zmq::socket_t(*this->context_, zmq::socket_type::rep);
00605             this->server_socket_->bind(this->server_endpoint_);
00606             this->server_socket_->set(zmq::sockopt::linger, 0);
00607         }
00608         catch (const zmq::error_t& error)
00609         {
00610             // Delete the socket and store the last error.
00611             delete this->server_socket_;
00612             this->server_socket_ = nullptr;
00613             last_error = &error;
00614         }
00615         reconnect_count--;
00616     } while (res == EADDRINUSE && reconnect_count > 0);
00617
00618     if (!this->server_socket_)
00619     {
00620         // Update the working flag and calls to the callback.
00621         this->server_working_ = false;
00622         this->onServerError(*last_error, "Error during socket creation.");
00623     }
00624     else
00625     {
00626         // Call to the internal callback.
00627         this->onServerStart();
00628     }
00629 }
00630 void CommandServerBase::onCustomCommandReceived(const CommandRequest&, CommandReply& rep)
00631 {

```

```

00632     rep.result = ServerResult::NOT_IMPLEMENTED;
00633 }
00634
00635 } // END NAMESPACES.
00636 //
=====

```

7.41 sources/CommandServerClient/common.cpp File Reference

```
#include "LibZMQUtils/CommandServerClient/common.h"
```

7.42 common.cpp

[Go to the documentation of this file.](#)

```

00001
00002  /*****
00003   *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
00004   *
00005   *   Copyright (C) 2023 Degoras Project Team
00006   *
00007   *   < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
00008   *
00009   *   < Jesús Relinque Madroñal >
00010   *
00011   *   This file is part of LibZMQUtils.
00012   *
00013   *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
00014   *   the EUPL license
00015   *   as soon they will be approved by the European Commission (IDABC).
00016   *
00017   *   This project is free software: you can redistribute it and/or modify it under the terms of the
00018   *   EUPL license as
00019   *   published by the IDABC, either Version 1.2 or, at your option, any later version.
00020   *
00021   *   This project is distributed in the hope that it will be useful. Unless required by applicable law
00022   *   or agreed to in
00023   *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
00024   *   without even the
00025   *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
00026   *   check specific
00027   *   language governing permissions and limitations and more details.
00028   *
00029   *   You should use this project in compliance with the EUPL license. You should have received a copy
00030   *   of the license
00031   *   along with this project. If not, see the license at < https://eupl.eu/ >.
00032   *
00033   *****/
00034
00035 #include "LibZMQUtils/CommandServerClient/common.h"
00036
00037 zmqutils::common::HostClient::HostClient(const std::string &ip, const std::string &name,
00038                                           const std::string &pid, const std::string &info) :
00039     ip(ip),
00040     hostname(name),
00041     pid(pid),
00042     info(info)
00043 {
00044     // Create the host client internal identification.
00045     this->id = ip + "/" + name + "/" + pid;
00046 }

```

7.43 sources/utils.cpp File Reference

```
#include <iomanip>
#include <sys/socket.h>
#include <ifaddrs.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sstream>
#include <iphlpapi.h>
#include "LibZMQUtils/utils.h"
```

Namespaces

- namespace [zmqutils](#)
- namespace [zmqutils::utils](#)

Macros

- `#define _WIN32_WINNT 0x0600`

Functions

- [LIBZMQUTILS_EXPORT](#) `std::vector< NetworkAdapterInfo > zmqutils::utils::getHostIPsWithInterfaces ()`
- [LIBZMQUTILS_EXPORT](#) `std::string zmqutils::utils::getHostname ()`
- [LIBZMQUTILS_EXPORT](#) `void zmqutils::utils::binarySerializeDeserialize (const void *data, size_t data_size, void *_bytes, void *dest)`
Binary serialization and deserialization.
- [LIBZMQUTILS_EXPORT](#) `std::string zmqutils::utils::timePointToString (const HRTIMEPOINTSTD &tp, const std::string &format="%Y-%m-%dT%H:%M:%S", bool add_ms=true, bool add_ns=false, bool utc=true)`
- [LIBZMQUTILS_EXPORT](#) `std::string zmqutils::utils::timePointToIso8601 (const HRTIMEPOINTSTD &tp, bool add_ms=true, bool add_ns=false)`
- [LIBZMQUTILS_EXPORT](#) `std::string zmqutils::utils::currentISO8601Date (bool add_ms=true)`
- [LIBZMQUTILS_EXPORT](#) `unsigned zmqutils::utils::getCurrentPID ()`

7.43.1 Macro Definition Documentation

7.43.1.1 _WIN32_WINNT

```
#define _WIN32_WINNT 0x0600
Definition at line 46 of file utils.cpp.
```

7.44 utils.cpp

[Go to the documentation of this file.](#)

```
00001  /*****
00002  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
00003  *
00004  *   Copyright (C) 2023 Degoras Project Team
00005  *
00006  *   < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
00007  *
00008  *   < Jesús Relinque Madroñal >
00009  *
00010  *   This file is part of LibZMQUtils.
00011  *
00012  *   *****/
```

```

00010 *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
the EUPL license *
00011 *   as soon they will be approved by the European Commission (IDABC).
*
00012 *
*
00013 *   This project is free software: you can redistribute it and/or modify it under the terms of the
EUPL license as *
00014 *   published by the IDABC, either Version 1.2 or, at your option, any later version.
*
00015 *
*
00016 *   This project is distributed in the hope that it will be useful. Unless required by applicable law
or agreed to in *
00017 *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
without even the *
00018 *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
check specific *
00019 *   language governing permissions and limitations and more details.
*
00020 *
*
00021 *   You should use this project in compliance with the EUPL license. You should have received a copy
of the license *
00022 *   along with this project. If not, see the license at < https://eupl.eu/ >.
*
00023 *****/
00024
00025
00026
00027 #include <iomanip>
00028 #ifdef _WIN32
00029 #include <winsock2.h>
00030 #include <ws2tcpip.h>
00031 #include <Windows.h>
00032 #else
00033 #include <sys/socket.h>
00034 #include <ifaddrs.h>
00035 #include <netinet/in.h>
00036 #include <arpa/inet.h>
00037 #include <unistd.h>
00038 #endif
00039
00040 #include <sstream>
00041 #include <iphlpapi.h>
00042
00043 #include "LibZMQUtils/utils.h"
00044
00045 #ifndef _WIN32_WINNT
00046 #define _WIN32_WINNT 0x0600
00047 #elif _WIN32_WINNT < 0x0600
00048 #undef _WIN32_WINNT
00049 #define _WIN32_WINNT 0x0600
00050 #endif
00051
00052 // ZMQUTILS NAMESPACES
00053 //
=====
00054 namespace zmqutils{
00055 namespace utils{
00056 //
=====
00057
00058 //
=====
00059 using std::chrono::duration;
00060 using std::chrono::duration_cast;
00061 using std::chrono::high_resolution_clock;
00062 using std::chrono::time_point_cast;
00063 //
=====
00064
00065 std::vector<NetworkAdapterInfo> getHostIPsWithInterfaces()
00066 {
00067     // Result container.
00068     std::vector<NetworkAdapterInfo> adapters;
00069
00070 #ifdef _WIN32
00071
00072     // Buffer size.
00073     ULONG buff_size = 0;
00074
00075     if (GetAdaptersAddresses(AF_INET, GAA_FLAG_SKIP_ANYCAST | GAA_FLAG_SKIP_MULTICAST |
GAA_FLAG_SKIP_DNS_SERVER,
00076                             nullptr, nullptr, &buff_size) != ERROR_BUFFER_OVERFLOW)
00077     {

```

```

00078         return adapters;
00079     }
00080
00081     std::vector<char> buffer(buff_size);
00082
00083     PIP_ADAPTER_ADDRESSES adapter_addrs = reinterpret_cast<PIP_ADAPTER_ADDRESSES>(&buffer[0]);
00084
00085     if (GetAdaptersAddresses(AF_INET, GAA_FLAG_SKIP_ANYCAST | GAA_FLAG_SKIP_MULTICAST |
00086 GAA_FLAG_SKIP_DNS_SERVER,
00087         nullptr, adapter_addrs, &buff_size) != NO_ERROR)
00088     {
00089         return adapters;
00090     }
00091
00092     while (adapter_addrs != nullptr)
00093     {
00094         if (adapter_addrs->OperStatus == IfOperStatusUp)
00095         {
00096             PIP_ADAPTER_UNICAST_ADDRESS unicast_addrs = adapter_addrs->FirstUnicastAddress;
00097             while (unicast_addrs != nullptr)
00098             {
00099                 sockaddr_in* sockaddr =
00100 reinterpret_cast<sockaddr_in*>(unicast_addrs->Address.lpSockaddr);
00101                 char* ip = inet_ntoa(sockaddr->sin_addr);
00102                 char f_name_ch[260];
00103                 char desc_ch[260];
00104                 char df_char = ' ';
00105
00106                 WideCharToMultiByte(CP_ACP, 0, adapter_addrs->FriendlyName, -1, f_name_ch, 260, &df_char,
00107 NULL);
00108                 WideCharToMultiByte(CP_ACP, 0, adapter_addrs->Description, -1, desc_ch, 260, &df_char,
00109 NULL);
00110
00111                 NetworkAdapterInfo adaptr;
00112                 adaptr.id = std::string(adapter_addrs->AdapterName);
00113                 adaptr.name = std::string(f_name_ch);
00114                 adaptr.descr = std::string(desc_ch);
00115                 adaptr.ip = std::string(ip);
00116                 adapters.push_back(adaptr);
00117                 unicast_addrs = unicast_addrs->Next;
00118             }
00119         }
00120         adapter_addrs = adapter_addrs->Next;
00121     }
00122
00123     #else
00124     // TODO
00125     #endif
00126
00127     // Return the ip interface maps.
00128     return adapters;
00129 }
00130
00131 std::string getHostname()
00132 {
00133     std::string name;
00134
00135     #ifdef _WIN32
00136     WSADATA wsaData;
00137
00138     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
00139         return "";
00140
00141     char buffer[256];
00142     if (gethostname(buffer, sizeof(buffer)) != 0)
00143         WSACleanup();
00144
00145     // Clear.
00146     WSACleanup();
00147
00148     // Store the data.
00149     name = std::string(buffer);
00150
00151     #else
00152     // TODO
00153     #endif
00154
00155     // Return the hostname.
00156     return name;
00157 }
00158
00159 void binarySerializeDeserialize(const void *data, size_t data_size_bytes, void *dest)
00160 {
00161     const std::uint8_t* data_bytes = reinterpret_cast<const std::uint8_t*>(data);
00162     std::uint8_t* dest_bytes = reinterpret_cast<std::uint8_t*>(dest);
00163     std::reverse_copy(data_bytes, data_bytes + data_size_bytes, dest_bytes);

```

```

00161 }
00162
00163 std::string timePointToString(const HRTimePointStd &tp, const std::string &format, bool add_ms, bool
add_ns, bool utc)
00164 {
00165     // Stream to hold the formatted string and the return container.
00166     std::ostringstream ss;
00167     // Convert the time point to a duration and get the different time fractions.
00168     HRTimePointStd::duration dur = tp.time_since_epoch();
00169     const time_t secs = duration_cast<std::chrono::seconds>(dur).count();
00170     const long long mill = duration_cast<std::chrono::milliseconds>(dur).count();
00171     const unsigned long long ns = duration_cast<std::chrono::nanoseconds>(dur).count();
00172     const unsigned long long s_ns = secs * 1e9;
00173     const unsigned long long t_ns = (ns - s_ns);
00174     // Format the duration.
00175     if (const std::tm *tm = (utc ? std::gmtime(&secs) : std::localtime(&secs)))
00176     {
00177         ss << std::put_time(tm, format.c_str());
00178         if (add_ms && !add_ns)
00179             ss << '.' << std::setw(3) << std::setfill('0') << (mill - secs * 1e3);
00180         else if (add_ns)
00181             ss << '.' << std::setw(9) << std::setfill('0') << t_ns;
00182     }
00183     else
00184     {
00185         // If error, return an empty string.
00186         return std::string();
00187     }
00188     // Return the container.
00189     return ss.str();
00190 }
00191
00192 std::string timePointToIso8601(const HRTimePointStd &tp, bool add_ms, bool add_ns)
00193 {
00194     // Return the ISO 8601 datetime.
00195     return timePointToString(tp, "%Y-%m-%dT%H:%M:%S", add_ms, add_ns) + 'Z';
00196 }
00197
00198 std::string currentISO8601Date(bool add_ms)
00199 {
00200     auto now = high_resolution_clock::now();
00201     return timePointToIso8601(now, add_ms);
00202 }
00203
00204 unsigned getCurrentPID()
00205 {
00206     #if defined(_WIN32)
00207         return GetCurrentProcessId();
00208     #elif defined(__unix__) || defined(__APPLE__) || defined(__linux__)
00209         return getpid();
00210     #else
00211         // Unsupported or unknown platform
00212         return 0;
00213     #endif
00214 }
00215
00216 }} // END NAMESPACES.
00217 //
=====

```


Index

- - zmq_msg_t, 93
- _WIN32_WINNT
 - utils.cpp, 230
- ~CommandClientBase
 - zmquils::CommandClientBase, 44
- ~CommandServerBase
 - zmquils::CommandServerBase, 52
- ~context_t
 - zmq::context_t, 60
- ~message_t
 - zmq::message_t, 67
- ~monitor_t
 - zmq::monitor_t, 72
- ~socket_t
 - zmq::socket_t, 88
- abort
 - zmq::monitor_t, 72
- ALREADY_CONNECTED
 - zmquils::common, 20
- AltAzPos
 - amelas::common::AltAzPos, 28
- amelas, 9
- amelas::AmelasClient, 28
 - AmelasClient, 29
 - onSendCommand, 29
 - resetClient, 29
 - sendCommand, 30
 - setClientHostIP, 30
 - setClientId, 30
 - startAutoAlive, 30
 - startClient, 30
 - stopAutoAlive, 30
 - stopClient, 30
- amelas::AmelasController, 31
 - AmelasController, 31
 - getDatetime, 31
 - getHomePosition, 31
 - setHomePosition, 32
- amelas::AmelasServer, 32
 - AmelasServer, 34
 - clearCallbacks, 34
 - getCallbackMap, 34
 - getConnectedClients, 34
 - getServerAddresses, 34
 - getServerEndpoint, 35
 - getServerPort, 35
 - getServerWorkerFuture, 35
 - isCallbackSet, 36
 - isWorking, 36
 - onCommandReceived, 36
 - onConnected, 37
 - onCustomCommandReceived, 37, 38
 - onDeadClient, 38
 - onDisconnected, 39
 - onInvalidMsgReceived, 39
 - onSendingResponse, 40
 - onServerError, 40
 - onServerStart, 41
 - onServerStop, 41
 - onWaitingCommand, 41
 - removeCallback, 42
 - setCallback, 42
 - setClientStatusCheck, 42
 - startServer, 43
 - stopServer, 43
- amelas::common, 9
 - AmelasServerCommand, 10
 - AmelasServerCommandStr, 11
 - AmelasServerResult, 11
 - AmelasServerResultStr, 11
 - ControllerCallback, 10
 - ControllerError, 11
 - EMPTY_CALLBACK, 11
 - END_AMELAS_COMMANDS, 10
 - GetDatetimeCallback, 10
 - GetHomePositionCallback, 10
 - INVALID_CALLBACK, 11
 - INVALID_POSITION, 11
 - kMaxCmdId, 11
 - kMinCmdId, 12
 - REQ_GET_DATETIME, 10
 - REQ_GET_HOME_POSITION, 10
 - REQ_SET_DATETIME, 10
 - REQ_SET_HOME_POSITION, 10
 - SetHomePositionCallback, 10
 - SUCCESS, 11
 - UNSAFE_POSITION, 11
- amelas::common::AltAzPos, 27
 - AltAzPos, 28
 - az, 28
 - el, 28
- amelas::utils, 12
 - makeCallback, 12
- AmelasClient
 - amelas::AmelasClient, 29
- AmelasController
 - amelas::AmelasController, 31

- AmelasServer
 - amelas::AmelasServer, [34](#)
- AmelasServerCommand
 - amelas::common, [10](#)
- AmelasServerCommandStr
 - amelas::common, [11](#)
- AmelasServerResult
 - amelas::common, [11](#)
- AmelasServerResultStr
 - amelas::common, [11](#)
- az
 - amelas::common::AltAzPos, [28](#)
- BAD_NO_PARAMETERS
 - zmqutils::common, [20](#)
- BAD_PARAMETERS
 - zmqutils::common, [20](#)
- binarySerializeDeserialize
 - zmqutils::utils, [23](#)
- bind
 - zmq::detail::socket_base, [80](#)
 - zmq::socket_ref, [84](#)
 - zmq::socket_t, [88](#)
- check_event
 - zmq::monitor_t, [72](#)
- clearCallbacks
 - amelas::AmelasServer, [34](#)
- client
 - zmqutils::common::CommandRequest, [48](#)
- CLIENT_NOT_CONNECTED
 - zmqutils::common, [20](#)
- CLIENT_STOPPED
 - zmqutils::common, [19](#)
- ClientResult
 - zmqutils::common, [19](#)
- close
 - zmq::context_t, [60](#)
 - zmq::socket_t, [88](#)
- command
 - zmqutils::common::CommandRequest, [48](#)
 - zmqutils::common::RequestData, [78](#)
- COMMAND_FAILED
 - zmqutils::common, [20](#)
- COMMAND_OK
 - zmqutils::common, [19](#), [20](#)
- CommandClientBase
 - zmqutils::CommandClientBase, [44](#)
- CommandReply
 - zmqutils::common::CommandReply, [46](#)
- CommandRequest
 - zmqutils::common::CommandRequest, [48](#)
- CommandServerBase
 - zmqutils::CommandServerBase, [51](#)
- CommandType
 - zmqutils::common, [18](#)
- connect
 - zmq::detail::socket_base, [80](#)
 - zmq::socket_ref, [84](#)
 - zmq::socket_t, [88](#)
- connected
 - zmq::detail::socket_base, [80](#)
 - zmq::socket_ref, [84](#)
 - zmq::socket_t, [88](#)
- context_t
 - zmq::context_t, [59](#)
- ControllerCallback
 - amelas::common, [10](#)
- ControllerError
 - amelas::common, [11](#)
- copy
 - zmq::message_t, [67](#)
- CPPZMQ_HAS_INCLUDE_CPP17
 - zmq.hpp, [166](#)
- CPPZMQ_HAS_OPTIONAL
 - zmq.hpp, [166](#)
- CPPZMQ_HAS_STRING_VIEW
 - zmq.hpp, [166](#)
- CPPZMQ_LANG
 - zmq.hpp, [166](#)
- CPPZMQ_VERSION
 - zmq.hpp, [166](#)
- CPPZMQ_VERSION_MAJOR
 - zmq.hpp, [166](#)
- CPPZMQ_VERSION_MINOR
 - zmq.hpp, [166](#)
- CPPZMQ_VERSION_PATCH
 - zmq.hpp, [166](#)
- currentISO8601Date
 - zmqutils::utils, [23](#)
- data
 - zmq::message_t, [68](#)
- descr
 - zmqutils::utils::NetworkAdapterInfo, [76](#)
- disconnect
 - zmq::detail::socket_base, [80](#)
 - zmq::socket_ref, [84](#)
 - zmq::socket_t, [89](#)
- EADDRINUSE
 - zmq.h, [130](#)
- EADDRNOTAVAIL
 - zmq.h, [130](#)
- EAFNOSUPPORT
 - zmq.h, [130](#)
- ECONNABORTED
 - zmq.h, [130](#)
- ECONNREFUSED
 - zmq.h, [130](#)
- ECONRESET
 - zmq.h, [130](#)
- EFSM
 - zmq.h, [130](#)
- EHOSTUNREACH
 - zmq.h, [131](#)
- EINPROGRESS
 - zmq.h, [131](#)

- el
 - amelas::common::AltAzPos, [28](#)
- empty
 - zmq::message_t, [68](#)
- EMPTY_CALLBACK
 - amelas::common, [11](#)
- EMPTY_CLIENT_IP
 - zmquils::common, [20](#)
- EMPTY_CLIENT_NAME
 - zmquils::common, [20](#)
- EMPTY_CLIENT_PID
 - zmquils::common, [20](#)
- EMPTY_MSG
 - zmquils::common, [19](#), [20](#)
- EMPTY_PARAMS
 - zmquils::common, [19](#), [20](#)
- EMSGSIZE
 - zmq.h, [131](#)
- EMTHREAD
 - zmq.h, [131](#)
- END_AMELAS_COMMANDS
 - amelas::common, [10](#)
- END_BASE_COMMANDS
 - zmquils::common, [19](#)
- END_BASE_ERRORS
 - zmquils::common, [19](#), [20](#)
- ENETDOWN
 - zmq.h, [131](#)
- ENETRESET
 - zmq.h, [131](#)
- ENETUNREACH
 - zmq.h, [131](#)
- ENOBUFS
 - zmq.h, [131](#)
- ENOCOMPATPROTO
 - zmq.h, [131](#)
- ENOTCONN
 - zmq.h, [131](#)
- ENOTSOCK
 - zmq.h, [131](#)
- ENOTSUP
 - zmq.h, [131](#)
- EPROTONOSUPPORT
 - zmq.h, [132](#)
- equal
 - zmq::message_t, [68](#)
- error_t
 - zmq::error_t, [62](#)
- ETERM
 - zmq.h, [132](#)
- ETIMEDOUT
 - zmq.h, [132](#)
- event
 - zmq_event_t, [92](#)
- events
 - zmq_pollitem_t, [93](#)
- examples/ExampleZMQCommandClientAmelas/AmelasExampleClient/am
 - [96](#)
- examples/ExampleZMQCommandClientAmelas/ExampleZMQClientAmela
 - [97](#), [98](#)
- examples/ExampleZMQCommanServerAmelas/AmelasExampleController
 - [101](#)
- examples/ExampleZMQCommanServerAmelas/AmelasExampleController
 - [103](#), [104](#)
- examples/ExampleZMQCommanServerAmelas/AmelasExampleController
 - [112](#)
- examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/an
 - [116](#)
- examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/an
 - [120](#)
- examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/co
 - [105](#), [106](#)
- examples/ExampleZMQCommanServerAmelas/ExampleZMQServerAmel
 - [122](#), [123](#)
- ExampleZMQClientAmelas.cpp
 - main, [98](#)
 - parseCommand, [98](#)
- ExampleZMQServerAmelas.cpp
 - gExitCv, [123](#)
 - gMtx, [123](#)
 - gSignInterrupt, [123](#)
 - main, [122](#)
- external/zmq/includes/zmq/zmq.h, [125](#), [155](#)
- external/zmq/includes/zmq/zmq.hpp, [164](#), [168](#)
- external/zmq/includes/zmq/zmq_addon.hpp, [199](#), [200](#)
- fd
 - zmq_pollitem_t, [93](#)
- fd_t
 - zmq, [13](#)
- flags_
 - zmq::detail::socket_base, [82](#)
 - zmq::socket_ref, [86](#)
 - zmq::socket_t, [91](#)
- free_fn
 - zmq, [13](#)
- from_handle
 - zmq, [16](#)
- from_handle_t
 - zmq::from_handle_t, [63](#)
- get
 - zmq::message_t, [68](#)
- getCallbackMap
 - amelas::AmelasServer, [34](#)
- getConnectedClients
 - amelas::AmelasServer, [34](#)
 - zmquils::CommandServerBase, [52](#)
- getctxopt
 - zmq::context_t, [60](#)
- getCurrentPID
 - zmquils::utils, [23](#)
- getDatetime
 - examples/ExampleZMQCommandClientAmelas/AmelasExampleClient/amelas_client.cpp, [31](#)
 - amelas::AmelasController, [31](#)
- GetDatetimeCallback

- amelas::common, 10
- getHomePosition
 - amelas::AmelasController, 31
- GetHomePositionCallback
 - amelas::common, 10
- getHostIPsWithInterfaces
 - zmqutils::utils, 24
- getHostname
 - zmqutils::utils, 24
- gets
 - zmq::message_t, 69
- getServerAddresses
 - amelas::AmelasServer, 34
 - zmqutils::CommandServerBase, 52
- getServerEndpoint
 - amelas::AmelasServer, 35
 - zmqutils::CommandServerBase, 52
- getServerPort
 - amelas::AmelasServer, 35
 - zmqutils::CommandServerBase, 53
- getServerWorkerFuture
 - amelas::AmelasServer, 35
 - zmqutils::CommandServerBase, 53
- getsockopt
 - zmq::detail::socket_base, 80, 81
 - zmq::socket_ref, 84, 85
 - zmq::socket_t, 89
- gExitCv
 - ExampleZMQServerAmelas.cpp, 123
- gMtx
 - ExampleZMQServerAmelas.cpp, 123
- gSignInterrupt
 - ExampleZMQServerAmelas.cpp, 123
- handle
 - zmq::context_t, 60
 - zmq::message_t, 69
- HostClient
 - zmqutils::common::HostClient, 64
- hostname
 - zmqutils::common::HostClient, 65
- HRTIMEPOINTSTD
 - zmqutils::utils, 23
- id
 - zmqutils::common::HostClient, 65
 - zmqutils::utils::NetworkAdapterInfo, 76
- includes/LibZMQUtils/CommandServerClient/command_client.h, 208, 209
- includes/LibZMQUtils/CommandServerClient/command_server.h, 211, 212
- includes/LibZMQUtils/CommandServerClient/common.h, 106, 108
- includes/LibZMQUtils/libzmqutils_global.h, 214
- includes/LibZMQUtils/utils.h, 112, 114
- info
 - zmqutils::common::HostClient, 65
- init
 - zmq::monitor_t, 73
- INTERNAL_ZMQ_ERROR
 - zmqutils::common, 19, 20
- INVALID_CALLBACK
 - amelas::common, 11
- INVALID_COMMAND
 - zmqutils::common, 19
- INVALID_MSG
 - zmqutils::common, 19, 20
- INVALID_PARTS
 - zmqutils::common, 19, 20
- INVALID_POSITION
 - amelas::common, 11
- ip
 - zmqutils::common::HostClient, 65
 - zmqutils::utils::NetworkAdapterInfo, 77
- isCallbackSet
 - amelas::AmelasServer, 36
- isWorking
 - amelas::AmelasServer, 36
 - zmqutils::CommandServerBase, 53
- joinArrays
 - zmqutils::utils::internal, 25
- joinArraysConstexpr
 - zmqutils::utils, 24
- kClientAlivePeriodMsec
 - zmqutils::common, 20
- kDefaultClientAliveTimeoutMsec
 - zmqutils::common, 20
- kDefaultServerAliveTimeoutMsec
 - zmqutils::common, 20
- kMaxBaseCmdId
 - zmqutils::common, 20
- kMaxCmdId
 - amelas::common, 11
- kMinBaseCmdId
 - zmqutils::common, 21
- kMinCmdId
 - amelas::common, 12
- kServerReconnTimes
 - zmqutils::common, 21
- kZmqEFSMError
 - zmqutils::common, 21
- last
 - zmq::detail::socket_base, 82
 - zmq::socket_ref, 86
 - zmq::socket_t, 91
- last_connection
 - zmqutils::common::HostClient, 65
- LIBZMQUTILS_EXPORT
 - libzmqutils_global.h, 214
- libzmqutils_global.h
 - LIBZMQUTILS_EXPORT, 214
- main
 - ExampleZMQClientAmelas.cpp, 98
 - ExampleZMQServerAmelas.cpp, 122

- makeCallback
 - amelas::utils, 12
- message_t
 - zmq::message_t, 66, 67
- MKGMTIME
 - utils.h, 114
- monitor
 - zmq::monitor_t, 73
- monitor_t
 - zmq::monitor_t, 72
 - zmq::socket_t, 91
- more
 - zmq::message_t, 69
- move
 - zmq::message_t, 69
- name
 - zmquils::utils::NetworkAdapterInfo, 77
- NOT_IMPLEMENTED
 - zmquils::common, 20
- num
 - zmq::error_t, 62
- on_event_accept_failed
 - zmq::monitor_t, 73
- on_event_accepted
 - zmq::monitor_t, 73
- on_event_bind_failed
 - zmq::monitor_t, 74
- on_event_close_failed
 - zmq::monitor_t, 74
- on_event_closed
 - zmq::monitor_t, 74
- on_event_connect_delayed
 - zmq::monitor_t, 74
- on_event_connect_retried
 - zmq::monitor_t, 74
- on_event_connected
 - zmq::monitor_t, 74
- on_event_disconnected
 - zmq::monitor_t, 75
- on_event_handshake_failed_auth
 - zmq::monitor_t, 75
- on_event_handshake_failed_no_detail
 - zmq::monitor_t, 75
- on_event_handshake_failed_protocol
 - zmq::monitor_t, 75
- on_event_handshake_succeeded
 - zmq::monitor_t, 75
- on_event_listening
 - zmq::monitor_t, 75
- on_event_unknown
 - zmq::monitor_t, 76
- on_monitor_started
 - zmq::monitor_t, 76
- onCommandReceived
 - amelas::AmelasServer, 36
 - zmquils::CommandServerBase, 53
- onConnected
 - amelas::AmelasServer, 37
 - zmquils::CommandServerBase, 54
- onCustomCommandReceived
 - amelas::AmelasServer, 37, 38
 - zmquils::CommandServerBase, 54
- onDeadClient
 - amelas::AmelasServer, 38
 - zmquils::CommandServerBase, 55
- onDisconnected
 - amelas::AmelasServer, 39
 - zmquils::CommandServerBase, 55
- onInvalidMsgReceived
 - amelas::AmelasServer, 39
 - zmquils::CommandServerBase, 56
- onSendCommand
 - amelas::AmelasClient, 29
 - zmquils::CommandClientBase, 44
- onSendingResponse
 - amelas::AmelasServer, 40
 - zmquils::CommandServerBase, 56
- onServerError
 - amelas::AmelasServer, 40
 - zmquils::CommandServerBase, 56
- onServerStart
 - amelas::AmelasServer, 41
 - zmquils::CommandServerBase, 57
- onServerStop
 - amelas::AmelasServer, 41
 - zmquils::CommandServerBase, 57
- onWaitingCommand
 - amelas::AmelasServer, 41
 - zmquils::CommandServerBase, 57
- operator bool
 - zmq::context_t, 60
- operator socket_ref
 - zmq::socket_t, 89
- operator void *
 - zmq::context_t, 60
 - zmq::socket_t, 89
- operator void const *
 - zmq::context_t, 60
 - zmq::socket_t, 89
- operator!=
 - zmq, 14
 - zmq::message_t, 69
- operator<
 - zmq, 14
- operator<<
 - zmq, 14
- operator<=
 - zmq, 14
- operator>
 - zmq, 14
- operator>=
 - zmq, 14
- operator=
 - zmquils::common::HostClient, 64
- operator==

- zmq, [14](#)
- zmq::message_t, [70](#)
- params
 - zmquils::common::CommandReply, [46](#)
 - zmquils::common::CommandRequest, [48](#)
 - zmquils::common::RequestData, [78](#)
- params_size
 - zmquils::common::CommandReply, [46](#)
 - zmquils::common::CommandRequest, [48](#)
 - zmquils::common::RequestData, [78](#)
- parseCommand
 - ExampleZMQClientAmelas.cpp, [98](#)
- pid
 - zmquils::common::HostClient, [65](#)
- poll
 - zmq, [15](#)
 - zmq::detail, [17](#)
- pollitem_t
 - zmq, [13](#)
- proxy
 - zmq, [15](#)
- proxy_steerable
 - zmq, [15](#)
- raw_msg
 - zmquils::common::CommandReply, [47](#)
 - zmquils::common::CommandRequest, [48](#)
- rebuild
 - zmq::message_t, [70](#)
- removeCallback
 - amelas::AmelasServer, [42](#)
- REQ_ALIVE
 - zmquils::common, [19](#)
- REQ_CONNECT
 - zmquils::common, [19](#)
- REQ_DISCONNECT
 - zmquils::common, [19](#)
- REQ_GET_DATETIME
 - amelas::common, [10](#)
- REQ_GET_HOME_POSITION
 - amelas::common, [10](#)
- REQ_SET_DATETIME
 - amelas::common, [10](#)
- REQ_SET_HOME_POSITION
 - amelas::common, [10](#)
- RequestData
 - zmquils::common::RequestData, [78](#)
- RESERVED_COMMANDS
 - zmquils::common, [19](#)
- resetClient
 - amelas::AmelasClient, [29](#)
 - zmquils::CommandClientBase, [44](#)
- result
 - zmquils::common::CommandReply, [47](#)
- ResultType
 - zmquils::common, [18](#)
- revents
 - zmq_pollitem_t, [93](#)
- SCTimePointStd
 - zmquils::utils, [23](#)
- send
 - zmq::detail::socket_base, [81](#)
 - zmq::socket_ref, [85](#)
 - zmq::socket_t, [90](#)
- sendCommand
 - amelas::AmelasClient, [30](#)
 - zmquils::CommandClientBase, [45](#)
- ServerCommand
 - zmquils::common, [19](#)
- ServerCommandStr
 - zmquils::common, [21](#)
- ServerResult
 - zmquils::common, [19](#)
- ServerResultStr
 - zmquils::common, [21](#)
- setCallback
 - amelas::AmelasServer, [42](#)
- setClientHostIP
 - amelas::AmelasClient, [30](#)
 - zmquils::CommandClientBase, [45](#)
- setClientId
 - amelas::AmelasClient, [30](#)
 - zmquils::CommandClientBase, [45](#)
- setClientStatusCheck
 - amelas::AmelasServer, [42](#)
 - zmquils::CommandServerBase, [58](#)
- setctxopt
 - zmq::context_t, [61](#)
- setHomePosition
 - amelas::AmelasController, [32](#)
- SetHomePositionCallback
 - amelas::common, [10](#)
- setsockopt
 - zmq::detail::socket_base, [81](#)
 - zmq::socket_ref, [85](#)
 - zmq::socket_t, [90](#)
- shutdown
 - zmq::context_t, [61](#)
- size
 - zmq::message_t, [70](#)
- socket
 - zmq_pollitem_t, [93](#)
- socket_base
 - zmq::detail::socket_base, [79](#)
- socket_ref
 - zmq::socket_ref, [83](#)
- socket_t
 - zmq::socket_t, [87](#)
- sources/CommandServerClient/command_client.cpp, [215](#)
- sources/CommandServerClient/command_server.cpp, [220, 221](#)
- sources/CommandServerClient/common.cpp, [229](#)
- sources/utils.cpp, [230](#)
- startAutoAlive
 - amelas::AmelasClient, [30](#)

- zmqutils::CommandClientBase, 45
- startClient
 - amelas::AmelasClient, 30
 - zmqutils::CommandClientBase, 45
- startServer
 - amelas::AmelasServer, 43
 - zmqutils::CommandServerBase, 58
- stopAutoAlive
 - amelas::AmelasClient, 30
 - zmqutils::CommandClientBase, 45
- stopClient
 - amelas::AmelasClient, 30
 - zmqutils::CommandClientBase, 45
- stopServer
 - amelas::AmelasServer, 43
 - zmqutils::CommandServerBase, 58
- str
 - zmq::message_t, 71
- SUCCESS
 - amelas::common, 11
- swap
 - zmq, 16
 - zmq::context_t, 61
 - zmq::message_t, 71
 - zmq::socket_t, 90
- TIMEOUT_REACHED
 - zmqutils::common, 19, 20
- timePointToIso8601
 - zmqutils::utils, 24
- timePointToString
 - zmqutils::utils, 24
- to_string
 - zmq::message_t, 71
- unbind
 - zmq::detail::socket_base, 81, 82
 - zmq::socket_ref, 85, 86
 - zmq::socket_t, 90, 91
- UNKNOWN_COMMAND
 - zmqutils::common, 20
- UNSAFE_POSITION
 - amelas::common, 11
- utils.cpp
 - _WIN32_WINNT, 230
- utils.h
 - MKGMTIME, 114
- value
 - zmq_event_t, 92
- version
 - zmq, 16
- what
 - zmq::error_t, 62
- zmq, 12
 - fd_t, 13
 - free_fn, 13

- from_handle, 16
 - operator!=, 14
 - operator<, 14
 - operator<<, 14
 - operator<=, 14
 - operator>, 14
 - operator>=, 14
 - operator==, 14
 - poll, 15
 - pollitem_t, 13
 - proxy, 15
 - proxy_steerable, 15
 - swap, 16
 - version, 16
- zmq.h
 - EADDRINUSE, 130
 - EADDRNOTAVAIL, 130
 - EAFNOSUPPORT, 130
 - ECONNABORTED, 130
 - ECONNREFUSED, 130
 - ECONNRESET, 130
 - EFSM, 130
 - EHOSTUNREACH, 131
 - EINPROGRESS, 131
 - EMSGSIZE, 131
 - EMTHREAD, 131
 - ENETDOWN, 131
 - ENETRESET, 131
 - ENETUNREACH, 131
 - ENOBUFS, 131
 - ENOCOMPATPROTO, 131
 - ENOTCONN, 131
 - ENOTSOCK, 131
 - ENOTSUP, 131
 - EPROTONOSUPPORT, 132
 - ETERM, 132
 - ETIMEDOUT, 132
 - ZMQ_AFFINITY, 132
 - zmq_atomic_counter_dec, 148
 - zmq_atomic_counter_destroy, 148
 - zmq_atomic_counter_inc, 148
 - zmq_atomic_counter_new, 148
 - zmq_atomic_counter_set, 148
 - zmq_atomic_counter_value, 148
 - ZMQ_BACKLOG, 132
 - zmq_bind, 148
 - ZMQ_BINDTODEVICE, 132
 - ZMQ_BLOCKY, 132
 - zmq_close, 148
 - ZMQ_CONFLATE, 132
 - zmq_connect, 148
 - ZMQ_CONNECT_RID, 132
 - ZMQ_CONNECT_ROUTING_ID, 132
 - ZMQ_CONNECT_TIMEOUT, 132
 - zmq_ctx_destroy, 148
 - zmq_ctx_get, 149
 - zmq_ctx_new, 149
 - zmq_ctx_set, 149

zmq_ctx_shutdown, 149
zmq_ctx_term, 149
ZMQ_CURVE, 132
zmq_curve_keypair, 149
zmq_curve_public, 149
ZMQ_CURVE_PUBLICKEY, 133
ZMQ_CURVE_SECRETKEY, 133
ZMQ_CURVE_SERVER, 133
ZMQ_CURVE_SERVERKEY, 133
ZMQ DEALER, 133
ZMQ_DEFINED_STDINT, 133
ZMQ_DELAY_ATTACH_ON_CONNECT, 133
zmq_device, 149
zmq_disconnect, 149
ZMQ_DONTWAIT, 133
zmq_errno, 149
ZMQ_EVENT_ACCEPT_FAILED, 133
ZMQ_EVENT_ACCEPTED, 133
ZMQ_EVENT_ALL, 133
ZMQ_EVENT_BIND_FAILED, 133
ZMQ_EVENT_CLOSE_FAILED, 134
ZMQ_EVENT_CLOSED, 134
ZMQ_EVENT_CONNECT_DELAYED, 134
ZMQ_EVENT_CONNECT_RETRIED, 134
ZMQ_EVENT_CONNECTED, 134
ZMQ_EVENT_DISCONNECTED, 134
ZMQ_EVENT_HANDSHAKE_FAILED_AUTH, 134
ZMQ_EVENT_HANDSHAKE_FAILED_NO_DETAIL, 134
ZMQ_EVENT_HANDSHAKE_FAILED_PROTOCOL, 134
ZMQ_EVENT_HANDSHAKE_SUCCEEDED, 134
ZMQ_EVENT_LISTENING, 134
ZMQ_EVENT_MONITOR_STOPPED, 134
ZMQ_EVENTS, 135
ZMQ_EXPORT, 135
ZMQ_FAIL_UNROUTABLE, 135
ZMQ_FD, 135
zmq_fd_t, 147
ZMQ_FORWARDER, 135
zmq_free_fn, 147
zmq_getsockopt, 150
ZMQ_GROUP_MAX_LENGTH, 135
ZMQ_GSSAPI, 135
ZMQ_GSSAPI_NT_HOSTBASED, 135
ZMQ_GSSAPI_NT_KRB5_PRINCIPAL, 135
ZMQ_GSSAPI_NT_USER_NAME, 135
ZMQ_GSSAPI_PLAINTEXT, 135
ZMQ_GSSAPI_PRINCIPAL, 135
ZMQ_GSSAPI_PRINCIPAL_NAME_TYPE, 136
ZMQ_GSSAPI_SERVER, 136
ZMQ_GSSAPI_SERVICE_PRINCIPAL, 136
ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAME_TYPE, 136
ZMQ_HANDSHAKE_IVL, 136
zmq_has, 150
ZMQ_HAS_CAPABILITIES, 136
ZMQ_HAUSNUMERO, 136
ZMQ_HAVE_TIMERS, 136
ZMQ_HEARTBEAT_IVL, 136
ZMQ_HEARTBEAT_TIMEOUT, 136
ZMQ_HEARTBEAT_TTL, 136
ZMQ_IDENTITY, 136
ZMQ_IMMEDIATE, 137
zmq_init, 150
ZMQ_INVERT_MATCHING, 137
ZMQ_IO_THREADS, 137
ZMQ_IO_THREADS_DFLT, 137
ZMQ_IPC_FILTER_GID, 137
ZMQ_IPC_FILTER_PID, 137
ZMQ_IPC_FILTER_UID, 137
ZMQ_IPV4ONLY, 137
ZMQ_IPV6, 137
ZMQ_LAST_ENDPOINT, 137
ZMQ_LINGER, 137
ZMQ_MAKE_VERSION, 137
ZMQ_MAX_MSGSZ, 138
ZMQ_MAX_SOCKETS, 138
ZMQ_MAX_SOCKETS_DFLT, 138
ZMQ_MAXMSGSIZE, 138
ZMQ_MECHANISM, 138
ZMQ_MORE, 138
zmq_msg_close, 150
zmq_msg_copy, 150
zmq_msg_data, 150
zmq_msg_get, 150
zmq_msg_gets, 150
zmq_msg_init, 150
zmq_msg_init_data, 150
zmq_msg_init_size, 151
zmq_msg_more, 151
zmq_msg_move, 151
zmq_msg_recv, 151
zmq_msg_send, 151
zmq_msg_set, 151
zmq_msg_size, 151
zmq_msg_t, 147
ZMQ_MSG_T_SIZE, 138
ZMQ_MULTICAST_HOPS, 138
ZMQ_MULTICAST_MAXTPDU, 138
ZMQ_NOBLOCK, 138
ZMQ_NULL, 138
ZMQ_PAIR, 139
ZMQ_PLAIN, 139
ZMQ_PLAIN_PASSWORD, 139
ZMQ_PLAIN_SERVER, 139
ZMQ_PLAIN_USERNAME, 139
zmq_poll, 151
ZMQ_POLLERR, 139
ZMQ_POLLIN, 139
zmq_pollitem_t, 147
ZMQ_POLLITEMS_DFLT, 139
ZMQ_POLLOUT, 139
ZMQ_POLLPRI, 139
ZMQ_PROBE_ROUTER, 139

[ZMQ_PROTOCOL_ERROR_WS_UNSPECIFIED,](#)
[139](#)
[ZMQ_PROTOCOL_ERROR_ZAP_BAD_REQUEST_ID,](#)
[140](#)
[ZMQ_PROTOCOL_ERROR_ZAP_BAD_VERSION,](#)
[140](#)
[ZMQ_PROTOCOL_ERROR_ZAP_INVALID_METADATA,](#)
[140](#)
[ZMQ_PROTOCOL_ERROR_ZAP_INVALID_STATUS_CODE,](#)
[140](#)
[ZMQ_PROTOCOL_ERROR_ZAP_MALFORMED_REPLY,](#)
[140](#)
[ZMQ_PROTOCOL_ERROR_ZAP_UNSPECIFIED,](#)
[140](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_CRYPTOGRAPHIC,](#)
[140](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_METADATA,](#)
[140](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_SEQUENCE,](#)
[140](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_KEY_EXCHANGE,](#)
[140](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND,](#)
[140](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_IDENTIFIER,](#)
[140](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_PAYLOAD,](#)
[141](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_SEQUENCE,](#)
[141](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_SIZE,](#)
[141](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_TYPE,](#)
[141](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_VERSION,](#)
[141](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_MECHANISM_MISMATCH,](#)
[141](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_UNEXPECTED_COMMAND,](#)
[141](#)
[ZMQ_PROTOCOL_ERROR_ZMTP_UNSPECIFIED,](#)
[141](#)
[zmq_proxy,](#) [151](#)
[zmq_proxy_steerable,](#) [152](#)
[ZMQ_PUB,](#) [141](#)
[ZMQ_PULL,](#) [141](#)
[ZMQ_PUSH,](#) [141](#)
[ZMQ_QUEUE,](#) [141](#)
[ZMQ_RATE,](#) [142](#)
[ZMQ_RCVBUF,](#) [142](#)
[ZMQ_RCVHWM,](#) [142](#)
[ZMQ_RCVMORE,](#) [142](#)
[ZMQ_RCVTIMEO,](#) [142](#)
[ZMQ_RECONNECT_IVL,](#) [142](#)
[ZMQ_RECONNECT_IVL_MAX,](#) [142](#)
[ZMQ_RECOVERY_IVL,](#) [142](#)
[zmq_recv,](#) [152](#)
[zmq_rcviov,](#) [152](#)

[zmq_recvmsg,](#) [152](#)
[ZMQ_REP,](#) [142](#)
[ZMQ_REQ,](#) [142](#)
[ZMQ_REQ_CORRELATE,](#) [142](#)
[ZMQ_REQ_RELAXED,](#) [142](#)
[ZMQ_ROUTER,](#) [143](#)
[ZMQ_ROUTER_BEHAVIOR,](#) [143](#)
[ZMQ_ROUTER_HANOVER,](#) [143](#)
[ZMQ_ROUTER_MANDATORY,](#) [143](#)
[ZMQ_ROUTER_RAW,](#) [143](#)
[ZMQ_ROUTING_ID,](#) [143](#)
[zmq_send,](#) [152](#)
[zmq_send_const,](#) [152](#)
[zmq_sendiov,](#) [152](#)
[zmq_sendmsg,](#) [153](#)
[zmq_setsockopt,](#) [153](#)
[ZMQ_SHARED,](#) [143](#)
[zmq_sleep,](#) [153](#)
[ZMQ_SNDBUF,](#) [143](#)
[ZMQ_SNDHWM,](#) [143](#)
[ZMQ_SNDMORE,](#) [143](#)
[ZMQ_SNDTIMEO,](#) [143](#)
[ZMQ_SOCKET,](#) [143](#)
[ZMQ_SOCKET_LIMIT,](#) [143](#)
[ZMQ_SOCKET_MONITOR,](#) [153](#)
[ZMQ SOCKS_PROXY,](#) [144](#)
[ZMQ_STREAM,](#) [144](#)
[ZMQ_STREAM_NOTIFY,](#) [144](#)
[ZMQ_STREAMER,](#) [144](#)
[zmq_stopwatch_intermediate,](#) [153](#)
[ZMQ_STOPWATCH_START,](#) [153](#)
[zmq_stopwatch_stop,](#) [153](#)
[ZMQ_STREAM,](#) [144](#)
[ZMQ_STREAM_NOTIFY,](#) [144](#)
[ZMQ_STREAMER,](#) [144](#)
[zmq_strerror,](#) [153](#)
[ZMQ_SUB,](#) [144](#)
[ZMQ_SUBSCRIBE,](#) [144](#)
[ZMQ_TCP_ACCEPT_FILTER,](#) [144](#)
[ZMQ_TCP_KEEPALIVE,](#) [144](#)
[ZMQ_TCP_KEEPALIVE_CNT,](#) [144](#)
[ZMQ_TCP_KEEPALIVE_IDLE,](#) [144](#)
[ZMQ_TCP_KEEPALIVE_INTVL,](#) [144](#)
[ZMQ_TCP_MAXRT,](#) [145](#)
[zmq_term,](#) [153](#)
[ZMQ_THREAD_AFFINITY_CPU_ADD,](#) [145](#)
[ZMQ_THREAD_AFFINITY_CPU_REMOVE,](#) [145](#)
[zmq_thread_fn,](#) [147](#)
[ZMQ_THREAD_NAME_PREFIX,](#) [145](#)
[ZMQ_THREAD_PRIORITY,](#) [145](#)
[ZMQ_THREAD_PRIORITY_DFLT,](#) [145](#)
[ZMQ_THREAD_SAFE,](#) [145](#)
[ZMQ_THREAD_SCHED_POLICY,](#) [145](#)
[ZMQ_THREAD_SCHED_POLICY_DFLT,](#) [145](#)
[zmq_threadclose,](#) [153](#)
[zmq_threadstart,](#) [154](#)
[zmq_timer_fn,](#) [148](#)
[zmq_timers_add,](#) [154](#)
[zmq_timers_cancel,](#) [154](#)
[zmq_timers_destroy,](#) [154](#)

- zmq_timers_execute, 154
- zmq_timers_new, 154
- zmq_timers_reset, 154
- zmq_timers_set_interval, 154
- zmq_timers_timeout, 154
- ZMQ_TOS, 145
- ZMQ_TYPE, 145
- zmq_unbind, 154
- ZMQ_UNSUBSCRIBE, 145
- ZMQ_USE_FD, 146
- ZMQ_VERSION, 146
- zmq_version, 155
- ZMQ_VERSION_MAJOR, 146
- ZMQ_VERSION_MINOR, 146
- ZMQ_VERSION_PATCH, 146
- ZMQ_VMCI_BUFFER_MAX_SIZE, 146
- ZMQ_VMCI_BUFFER_MIN_SIZE, 146
- ZMQ_VMCI_BUFFER_SIZE, 146
- ZMQ_VMCI_CONNECT_TIMEOUT, 146
- ZMQ_XPUB, 146
- ZMQ_XPUB_MANUAL, 146
- ZMQ_XPUB_NODROP, 146
- ZMQ_XPUB_VERBOSE, 147
- ZMQ_XPUB_VERBOSE, 147
- ZMQ_XPUB_WELCOME_MSG, 147
- ZMQ_XREP, 147
- ZMQ_XREQ, 147
- ZMQ_XSUB, 147
- zmq_z85_decode, 155
- zmq_z85_encode, 155
- ZMQ_ZAP_DOMAIN, 147
- zmq.hpp
 - CPPZMQ_HAS_INCLUDE_CPP17, 166
 - CPPZMQ_HAS_OPTIONAL, 166
 - CPPZMQ_HAS_STRING_VIEW, 166
 - CPPZMQ_LANG, 166
 - CPPZMQ_VERSION, 166
 - CPPZMQ_VERSION_MAJOR, 166
 - CPPZMQ_VERSION_MINOR, 166
 - CPPZMQ_VERSION_PATCH, 166
 - ZMQ_ASSERT, 166
 - ZMQ_CONSTEXPR_FN, 166
 - ZMQ_CONSTEXPR_IF, 166
 - ZMQ_CONSTEXPR_VAR, 167
 - ZMQ_CPP11_DEPRECATED, 167
 - ZMQ_DELETED_FUNCTION, 167
 - ZMQ_DEPRECATED, 167
 - ZMQ_EXPLICIT, 167
 - ZMQ_HAS_PROXY_STEERABLE, 167
 - ZMQ_INLINE_VAR, 167
 - ZMQ_NEW_MONITOR_EVENT_LAYOUT, 167
 - ZMQ_NODISCARD, 167
 - ZMQ_NOTHROW, 167
 - ZMQ_NULLPTR, 167
 - ZMQ_OVERRIDE, 167
- zmq::context_t, 59
 - ~context_t, 60
 - close, 60
 - context_t, 59
 - getctxopt, 60
 - handle, 60
 - operator bool, 60
 - operator void *, 60
 - operator void const *, 60
 - setctxopt, 61
 - shutdown, 61
 - swap, 61
- zmq::detail, 17
 - poll, 17
- zmq::detail::socket_base, 78
 - bind, 80
 - connect, 80
 - connected, 80
 - disconnect, 80
 - flags_, 82
 - getsockopt, 80, 81
 - last, 82
 - send, 81
 - setsockopt, 81
 - socket_base, 79
 - unbind, 81, 82
 - ZMQ_CPP11_DEPRECATED, 82
- zmq::error_t, 61
 - error_t, 62
 - num, 62
 - what, 62
- zmq::from_handle_t, 62
 - from_handle_t, 63
- zmq::from_handle_t::private, 27
- zmq::message_t, 66
 - ~message_t, 67
 - copy, 67
 - data, 68
 - empty, 68
 - equal, 68
 - get, 68
 - gets, 69
 - handle, 69
 - message_t, 66, 67
 - more, 69
 - move, 69
 - operator!=, 69
 - operator==, 70
 - rebuild, 70
 - size, 70
 - str, 71
 - swap, 71
 - to_string, 71
- zmq::monitor_t, 71
 - ~monitor_t, 72
 - abort, 72
 - check_event, 72
 - init, 73
 - monitor, 73
 - monitor_t, 72
 - on_event_accept_failed, 73

- on_event_accepted, [73](#)
- on_event_bind_failed, [74](#)
- on_event_close_failed, [74](#)
- on_event_closed, [74](#)
- on_event_connect_delayed, [74](#)
- on_event_connect_retried, [74](#)
- on_event_connected, [74](#)
- on_event_disconnected, [75](#)
- on_event_handshake_failed_auth, [75](#)
- on_event_handshake_failed_no_detail, [75](#)
- on_event_handshake_failed_protocol, [75](#)
- on_event_handshake_succeeded, [75](#)
- on_event_listening, [75](#)
- on_event_unknown, [76](#)
- on_monitor_started, [76](#)
- zmq::socket_ref, [82](#)
 - bind, [84](#)
 - connect, [84](#)
 - connected, [84](#)
 - disconnect, [84](#)
 - flags_, [86](#)
 - getsockopt, [84](#), [85](#)
 - last, [86](#)
 - send, [85](#)
 - setsockopt, [85](#)
 - socket_ref, [83](#)
 - unbind, [85](#), [86](#)
 - ZMQ_CPP11_DEPRECATED, [86](#)
- zmq::socket_t, [86](#)
 - ~socket_t, [88](#)
 - bind, [88](#)
 - close, [88](#)
 - connect, [88](#)
 - connected, [88](#)
 - disconnect, [89](#)
 - flags_, [91](#)
 - getsockopt, [89](#)
 - last, [91](#)
 - monitor_t, [91](#)
 - operator socket_ref, [89](#)
 - operator void *, [89](#)
 - operator void const *, [89](#)
 - send, [90](#)
 - setsockopt, [90](#)
 - socket_t, [87](#)
 - swap, [90](#)
 - unbind, [90](#), [91](#)
 - ZMQ_CPP11_DEPRECATED, [91](#)
- ZMQ_AFFINITY
 - zmq.h, [132](#)
- ZMQ_ASSERT
 - zmq.hpp, [166](#)
- zmq_atomic_counter_dec
 - zmq.h, [148](#)
- zmq_atomic_counter_destroy
 - zmq.h, [148](#)
- zmq_atomic_counter_inc
 - zmq.h, [148](#)
- zmq_atomic_counter_new
 - zmq.h, [148](#)
- zmq_atomic_counter_set
 - zmq.h, [148](#)
- zmq_atomic_counter_value
 - zmq.h, [148](#)
- ZMQ_BACKLOG
 - zmq.h, [132](#)
- zmq_bind
 - zmq.h, [148](#)
- ZMQ_BINDTODEVICE
 - zmq.h, [132](#)
- ZMQ_BLOCKY
 - zmq.h, [132](#)
- zmq_close
 - zmq.h, [148](#)
- ZMQ_CONFLATE
 - zmq.h, [132](#)
- zmq_connect
 - zmq.h, [148](#)
- ZMQ_CONNECT_RID
 - zmq.h, [132](#)
- ZMQ_CONNECT_ROUTING_ID
 - zmq.h, [132](#)
- ZMQ_CONNECT_TIMEOUT
 - zmq.h, [132](#)
- ZMQ_CONSTEXPR_FN
 - zmq.hpp, [166](#)
- ZMQ_CONSTEXPR_IF
 - zmq.hpp, [166](#)
- ZMQ_CONSTEXPR_VAR
 - zmq.hpp, [167](#)
- ZMQ_CPP11_DEPRECATED
 - zmq.hpp, [167](#)
 - zmq::detail::socket_base, [82](#)
 - zmq::socket_ref, [86](#)
 - zmq::socket_t, [91](#)
- zmq_ctx_destroy
 - zmq.h, [148](#)
- zmq_ctx_get
 - zmq.h, [149](#)
- zmq_ctx_new
 - zmq.h, [149](#)
- zmq_ctx_set
 - zmq.h, [149](#)
- zmq_ctx_shutdown
 - zmq.h, [149](#)
- zmq_ctx_term
 - zmq.h, [149](#)
- ZMQ_CURVE
 - zmq.h, [132](#)
- zmq_curve_keypair
 - zmq.h, [149](#)
- zmq_curve_public
 - zmq.h, [149](#)
- ZMQ_CURVE_PUBLICKEY
 - zmq.h, [133](#)
- ZMQ_CURVE_SECRETKEY

- zmq.h, [133](#)
- ZMQ_CURVE_SERVER
 - zmq.h, [133](#)
- ZMQ_CURVE_SERVERKEY
 - zmq.h, [133](#)
- ZMQ DEALER
 - zmq.h, [133](#)
- ZMQ_DEFINED_STDINT
 - zmq.h, [133](#)
- ZMQ_DELAY_ATTACH_ON_CONNECT
 - zmq.h, [133](#)
- ZMQ_DELETED_FUNCTION
 - zmq.hpp, [167](#)
- ZMQ_DEPRECATED
 - zmq.hpp, [167](#)
- zmq_device
 - zmq.h, [149](#)
- zmq_disconnect
 - zmq.h, [149](#)
- ZMQ_DONTWAIT
 - zmq.h, [133](#)
- zmq_errno
 - zmq.h, [149](#)
- ZMQ_EVENT_ACCEPT_FAILED
 - zmq.h, [133](#)
- ZMQ_EVENT_ACCEPTED
 - zmq.h, [133](#)
- ZMQ_EVENT_ALL
 - zmq.h, [133](#)
- ZMQ_EVENT_BIND_FAILED
 - zmq.h, [133](#)
- ZMQ_EVENT_CLOSE_FAILED
 - zmq.h, [134](#)
- ZMQ_EVENT_CLOSED
 - zmq.h, [134](#)
- ZMQ_EVENT_CONNECT_DELAYED
 - zmq.h, [134](#)
- ZMQ_EVENT_CONNECT_RETRIED
 - zmq.h, [134](#)
- ZMQ_EVENT_CONNECTED
 - zmq.h, [134](#)
- ZMQ_EVENT_DISCONNECTED
 - zmq.h, [134](#)
- ZMQ_EVENT_HANDSHAKE_FAILED_AUTH
 - zmq.h, [134](#)
- ZMQ_EVENT_HANDSHAKE_FAILED_NO_DETAIL
 - zmq.h, [134](#)
- ZMQ_EVENT_HANDSHAKE_FAILED_PROTOCOL
 - zmq.h, [134](#)
- ZMQ_EVENT_HANDSHAKE_SUCCEEDED
 - zmq.h, [134](#)
- ZMQ_EVENT_LISTENING
 - zmq.h, [134](#)
- ZMQ_EVENT_MONITOR_STOPPED
 - zmq.h, [134](#)
- zmq_event_t, [92](#)
 - event, [92](#)
 - value, [92](#)
- ZMQ_EVENTS
 - zmq.h, [135](#)
- ZMQ_EXPLICIT
 - zmq.hpp, [167](#)
- ZMQ_EXPORT
 - zmq.h, [135](#)
- ZMQ_FAIL_UNROUTABLE
 - zmq.h, [135](#)
- ZMQ_FD
 - zmq.h, [135](#)
- zmq_fd_t
 - zmq.h, [147](#)
- ZMQ_FORWARDER
 - zmq.h, [135](#)
- zmq_free_fn
 - zmq.h, [147](#)
- zmq_getsockopt
 - zmq.h, [150](#)
- ZMQ_GROUP_MAX_LENGTH
 - zmq.h, [135](#)
- ZMQ_GSSAPI
 - zmq.h, [135](#)
- ZMQ_GSSAPI_NT_HOSTBASED
 - zmq.h, [135](#)
- ZMQ_GSSAPI_NT_KRB5_PRINCIPAL
 - zmq.h, [135](#)
- ZMQ_GSSAPI_NT_USER_NAME
 - zmq.h, [135](#)
- ZMQ_GSSAPI_PLAINTEXT
 - zmq.h, [135](#)
- ZMQ_GSSAPI_PRINCIPAL
 - zmq.h, [135](#)
- ZMQ_GSSAPI_PRINCIPAL_NAMETYPE
 - zmq.h, [136](#)
- ZMQ_GSSAPI_SERVER
 - zmq.h, [136](#)
- ZMQ_GSSAPI_SERVICE_PRINCIPAL
 - zmq.h, [136](#)
- ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAMETYPE
 - zmq.h, [136](#)
- ZMQ_HANDSHAKE_IVL
 - zmq.h, [136](#)
- zmq_has
 - zmq.h, [150](#)
- ZMQ_HAS_CAPABILITIES
 - zmq.h, [136](#)
- ZMQ_HAS_PROXY_STEERABLE
 - zmq.hpp, [167](#)
- ZMQ_HAUSNUMERO
 - zmq.h, [136](#)
- ZMQ_HAVE_TIMERS
 - zmq.h, [136](#)
- ZMQ_HEARTBEAT_IVL
 - zmq.h, [136](#)
- ZMQ_HEARTBEAT_TIMEOUT
 - zmq.h, [136](#)
- ZMQ_HEARTBEAT_TTL
 - zmq.h, [136](#)

ZMQ_IDENTITY
 zmq.h, 136

ZMQ_IMMEDIATE
 zmq.h, 137

zmq_init
 zmq.h, 150

ZMQ_INLINE_VAR
 zmq.hpp, 167

ZMQ_INVERT_MATCHING
 zmq.h, 137

ZMQ_IO_THREADS
 zmq.h, 137

ZMQ_IO_THREADS_DFLT
 zmq.h, 137

ZMQ_IPC_FILTER_GID
 zmq.h, 137

ZMQ_IPC_FILTER_PID
 zmq.h, 137

ZMQ_IPC_FILTER_UID
 zmq.h, 137

ZMQ_IPV4ONLY
 zmq.h, 137

ZMQ_IPV6
 zmq.h, 137

ZMQ_LAST_ENDPOINT
 zmq.h, 137

ZMQ_LINGER
 zmq.h, 137

ZMQ_MAKE_VERSION
 zmq.h, 137

ZMQ_MAX_MSGSZ
 zmq.h, 138

ZMQ_MAX_SOCKETS
 zmq.h, 138

ZMQ_MAX_SOCKETS_DFLT
 zmq.h, 138

ZMQ_MAXMSGSIZE
 zmq.h, 138

ZMQ_MECHANISM
 zmq.h, 138

ZMQ_MORE
 zmq.h, 138

zmq_msg_close
 zmq.h, 150

zmq_msg_copy
 zmq.h, 150

zmq_msg_data
 zmq.h, 150

zmq_msg_get
 zmq.h, 150

zmq_msg_gets
 zmq.h, 150

zmq_msg_init
 zmq.h, 150

zmq_msg_init_data
 zmq.h, 150

zmq_msg_init_size
 zmq.h, 151

zmq_msg_more
 zmq.h, 151

zmq_msg_move
 zmq.h, 151

zmq_msg_recv
 zmq.h, 151

zmq_msg_send
 zmq.h, 151

zmq_msg_set
 zmq.h, 151

zmq_msg_size
 zmq.h, 151

zmq_msg_t, 93
 _, 93
 zmq.h, 147

ZMQ_MSG_T_SIZE
 zmq.h, 138

ZMQ_MULTICAST_HOPS
 zmq.h, 138

ZMQ_MULTICAST_MAXTPDU
 zmq.h, 138

ZMQ_NEW_MONITOR_EVENT_LAYOUT
 zmq.hpp, 167

ZMQ_NOBLOCK
 zmq.h, 138

ZMQ_NODISCARD
 zmq.hpp, 167

ZMQ_NOTHROW
 zmq.hpp, 167

ZMQ_NULL
 zmq.h, 138

ZMQ_NULLPTR
 zmq.hpp, 167

ZMQ_OVERRIDE
 zmq.hpp, 167

ZMQ_PAIR
 zmq.h, 139

ZMQ_PLAIN
 zmq.h, 139

ZMQ_PLAIN_PASSWORD
 zmq.h, 139

ZMQ_PLAIN_SERVER
 zmq.h, 139

ZMQ_PLAIN_USERNAME
 zmq.h, 139

zmq_poll
 zmq.h, 151

ZMQ_POLLERR
 zmq.h, 139

ZMQ_POLLIN
 zmq.h, 139

zmq_pollitem_t, 93
 events, 93
 fd, 93
 revents, 93
 socket, 93
 zmq.h, 147

ZMQ_POLLITEMS_DFLT

zmq.h, 139
 ZMQ_POLLOUT
 zmq.h, 139
 ZMQ_POLLPRI
 zmq.h, 139
 ZMQ_PROBE_ROUTER
 zmq.h, 139
 ZMQ_PROTOCOL_ERROR_WS_UNSPECIFIED
 zmq.h, 139
 ZMQ_PROTOCOL_ERROR_ZAP_BAD_REQUEST_ID
 zmq.h, 140
 ZMQ_PROTOCOL_ERROR_ZAP_BAD_VERSION
 zmq.h, 140
 ZMQ_PROTOCOL_ERROR_ZAP_INVALID_METADATA
 zmq.h, 140
 ZMQ_PROTOCOL_ERROR_ZAP_INVALID_STATUS_CODE
 zmq.h, 140
 ZMQ_PROTOCOL_ERROR_ZAP_MALFORMED_REPLY
 zmq.h, 140
 ZMQ_PROTOCOL_ERROR_ZAP_UNSPECIFIED
 zmq.h, 140
 ZMQ_PROTOCOL_ERROR_ZMTP_CRYPTOGRAPHIC
 zmq.h, 140
 ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_METADATA
 zmq.h, 140
 ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_SEQUENCE
 zmq.h, 140
 ZMQ_PROTOCOL_ERROR_ZMTP_KEY_EXCHANGE
 zmq.h, 140
 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND
 zmq.h, 140
 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_RELAXED
 zmq.h, 140
 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_ROUTE
 zmq.h, 141
 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_MESSAGE_BEHAVIOR
 zmq.h, 141
 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_REPLY_HANDOVER
 zmq.h, 141
 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_UNSPECIFIED
 zmq.h, 141
 ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_WELCOME
 zmq.h, 141
 ZMQ_PROTOCOL_ERROR_ZMTP_MECHANISM_MISMATCH
 zmq.h, 141
 ZMQ_PROTOCOL_ERROR_ZMTP_UNEXPECTED_COMMAND
 zmq.h, 141
 ZMQ_PROTOCOL_ERROR_ZMTP_UNSPECIFIED
 zmq.h, 141
 zmq_proxy
 zmq.h, 151
 zmq_proxy_steerable
 zmq.h, 152
 ZMQ_PUB
 zmq.h, 141
 ZMQ_PULL
 zmq.h, 141
 ZMQ_PUSH
 zmq.h, 141
 ZMQ_QUEUE
 zmq.h, 141
 ZMQ_RATE
 zmq.h, 142
 ZMQ_RCVBUF
 zmq.h, 142
 ZMQ_RCVHWM
 zmq.h, 142
 ZMQ_RCVMORE
 zmq.h, 142
 ZMQ_RCVTIMEO
 zmq.h, 142
 ZMQ_RECONNECT_IVL
 zmq.h, 142
 ZMQ_RECONNECT_IVL_MAX
 zmq.h, 142
 ZMQ_RECOVERY_IVL
 zmq.h, 142
 zmq_recv
 zmq.h, 152
 zmq_recviov
 zmq.h, 152
 zmq_recvmsg
 zmq.h, 152
 ZMQ_REQ
 zmq.h, 142
 ZMQ_RELAXED
 zmq.h, 142
 ZMQ_ROUTER
 zmq.h, 143
 ZMQ_ROUTER_BEHAVIOR
 zmq.h, 143
 ZMQ_REPLY_HANDOVER
 zmq.h, 143
 ZMQ_REPLY_UNSPECIFIED
 zmq.h, 143
 ZMQ_ROUTING_ID
 zmq.h, 143
 zmq_send
 zmq.h, 152
 zmq_send_const
 zmq.h, 152
 zmq_sendiov
 zmq.h, 152
 zmq_sendmsg
 zmq.h, 153
 zmq_setsockopt
 zmq.h, 153
 ZMQ_SHARED
 zmq.h, 143
 zmq_sleep

zmq.h, [153](#)
ZMQ_SNDBUF
zmq.h, [143](#)
ZMQ_SNDHWM
zmq.h, [143](#)
ZMQ_SNDMORE
zmq.h, [143](#)
ZMQ_SNDTIMEO
zmq.h, [143](#)
zmq_socket
zmq.h, [153](#)
ZMQ_SOCKET_LIMIT
zmq.h, [143](#)
zmq_socket_monitor
zmq.h, [153](#)
ZMQ SOCKS_PROXY
zmq.h, [144](#)
ZMQ_SRCFD
zmq.h, [144](#)
zmq_stopwatch_intermediate
zmq.h, [153](#)
zmq_stopwatch_start
zmq.h, [153](#)
zmq_stopwatch_stop
zmq.h, [153](#)
ZMQ_STREAM
zmq.h, [144](#)
ZMQ_STREAM_NOTIFY
zmq.h, [144](#)
ZMQ_STREAMER
zmq.h, [144](#)
zmq_strerror
zmq.h, [153](#)
ZMQ_SUB
zmq.h, [144](#)
ZMQ_SUBSCRIBE
zmq.h, [144](#)
ZMQ_TCP_ACCEPT_FILTER
zmq.h, [144](#)
ZMQ_TCP_KEEPALIVE
zmq.h, [144](#)
ZMQ_TCP_KEEPALIVE_CNT
zmq.h, [144](#)
ZMQ_TCP_KEEPALIVE_IDLE
zmq.h, [144](#)
ZMQ_TCP_KEEPALIVE_INTVL
zmq.h, [144](#)
ZMQ_TCP_MAXRT
zmq.h, [145](#)
zmq_term
zmq.h, [153](#)
ZMQ_THREAD_AFFINITY_CPU_ADD
zmq.h, [145](#)
ZMQ_THREAD_AFFINITY_CPU_REMOVE
zmq.h, [145](#)
zmq_thread_fn
zmq.h, [147](#)
ZMQ_THREAD_NAME_PREFIX
zmq.h, [145](#)
ZMQ_THREAD_PRIORITY
zmq.h, [145](#)
ZMQ_THREAD_PRIORITY_DFLT
zmq.h, [145](#)
ZMQ_THREAD_SAFE
zmq.h, [145](#)
ZMQ_THREAD_SCHED_POLICY
zmq.h, [145](#)
ZMQ_THREAD_SCHED_POLICY_DFLT
zmq.h, [145](#)
zmq_threadclose
zmq.h, [153](#)
zmq_threadstart
zmq.h, [154](#)
zmq_timer_fn
zmq.h, [148](#)
zmq_timers_add
zmq.h, [154](#)
zmq_timers_cancel
zmq.h, [154](#)
zmq_timers_destroy
zmq.h, [154](#)
zmq_timers_execute
zmq.h, [154](#)
zmq_timers_new
zmq.h, [154](#)
zmq_timers_reset
zmq.h, [154](#)
zmq_timers_set_interval
zmq.h, [154](#)
zmq_timers_timeout
zmq.h, [154](#)
ZMQ_TOS
zmq.h, [145](#)
ZMQ_TYPE
zmq.h, [145](#)
zmq_unbind
zmq.h, [154](#)
ZMQ_UNSUBSCRIBE
zmq.h, [145](#)
ZMQ_USE_FD
zmq.h, [146](#)
ZMQ_VERSION
zmq.h, [146](#)
zmq_version
zmq.h, [155](#)
ZMQ_VERSION_MAJOR
zmq.h, [146](#)
ZMQ_VERSION_MINOR
zmq.h, [146](#)
ZMQ_VERSION_PATCH
zmq.h, [146](#)
ZMQ_VMCI_BUFFER_MAX_SIZE
zmq.h, [146](#)
ZMQ_VMCI_BUFFER_MIN_SIZE
zmq.h, [146](#)
ZMQ_VMCI_BUFFER_SIZE

- zmq.h, 146
- ZMQ_VMCI_CONNECT_TIMEOUT
 - zmq.h, 146
- ZMQ_XPUB
 - zmq.h, 146
- ZMQ_XPUB_MANUAL
 - zmq.h, 146
- ZMQ_XPUB_NODROP
 - zmq.h, 146
- ZMQ_XPUB_VERBOSE
 - zmq.h, 147
- ZMQ_XPUB_VERBOSE
 - zmq.h, 147
- ZMQ_XPUB_WELCOME_MSG
 - zmq.h, 147
- ZMQ_XREP
 - zmq.h, 147
- ZMQ_XREQ
 - zmq.h, 147
- ZMQ_XSUB
 - zmq.h, 147
- zmq_z85_decode
 - zmq.h, 155
- zmq_z85_encode
 - zmq.h, 155
- ZMQ_ZAP_DOMAIN
 - zmq.h, 147
- zmqtutils, 17
- zmqtutils::CommandClientBase, 43
 - ~CommandClientBase, 44
 - CommandClientBase, 44
 - onSendCommand, 44
 - resetClient, 44
 - sendCommand, 45
 - setClientHostIP, 45
 - setClientId, 45
 - startAutoAlive, 45
 - startClient, 45
 - stopAutoAlive, 45
 - stopClient, 45
- zmqtutils::CommandServerBase, 49
 - ~CommandServerBase, 52
 - CommandServerBase, 51
 - getConnectedClients, 52
 - getServerAddresses, 52
 - getServerEndpoint, 52
 - getServerPort, 53
 - getServerWorkerFuture, 53
 - isWorking, 53
 - onCommandReceived, 53
 - onConnected, 54
 - onCustomCommandReceived, 54
 - onDeadClient, 55
 - onDisconnected, 55
 - onInvalidMsgReceived, 56
 - onSendingResponse, 56
 - onServerError, 56
 - onServerStart, 57
 - onServerStop, 57
 - onWaitingCommand, 57
 - setClientStatusCheck, 58
 - startServer, 58
 - stopServer, 58
- zmqtutils::common, 17
 - ALREADY_CONNECTED, 20
 - BAD_NO_PARAMETERS, 20
 - BAD_PARAMETERS, 20
 - CLIENT_NOT_CONNECTED, 20
 - CLIENT_STOPPED, 19
 - ClientResult, 19
 - COMMAND_FAILED, 20
 - COMMAND_OK, 19, 20
 - CommandType, 18
 - EMPTY_CLIENT_IP, 20
 - EMPTY_CLIENT_NAME, 20
 - EMPTY_CLIENT_PID, 20
 - EMPTY_MSG, 19, 20
 - EMPTY_PARAMS, 19, 20
 - END_BASE_COMMANDS, 19
 - END_BASE_ERRORS, 19, 20
 - INTERNAL_ZMQ_ERROR, 19, 20
 - INVALID_COMMAND, 19
 - INVALID_MSG, 19, 20
 - INVALID_PARTS, 19, 20
 - kClientAlivePeriodMsec, 20
 - kDefaultClientAliveTimeoutMsec, 20
 - kDefaultServerAliveTimeoutMsec, 20
 - kMaxBaseCmdId, 20
 - kMinBaseCmdId, 21
 - kServerReconnTimes, 21
 - kZmqEFSMError, 21
 - NOT_IMPLEMENTED, 20
 - REQ_ALIVE, 19
 - REQ_CONNECT, 19
 - REQ_DISCONNECT, 19
 - RESERVED_COMMANDS, 19
 - ResultType, 18
 - ServerCommand, 19
 - ServerCommandStr, 21
 - ServerResult, 19
 - ServerResultStr, 21
 - TIMEOUT_REACHED, 19, 20
 - UNKNOWN_COMMAND, 20
- zmqtutils::common::CommandReply, 46
 - CommandReply, 46
 - params, 46
 - params_size, 46
 - raw_msg, 47
 - result, 47
- zmqtutils::common::CommandRequest, 47
 - client, 48
 - command, 48
 - CommandRequest, 48
 - params, 48
 - params_size, 48
 - raw_msg, 48

- zmqutils::common::HostClient, [63](#)
 - HostClient, [64](#)
 - hostname, [65](#)
 - id, [65](#)
 - info, [65](#)
 - ip, [65](#)
 - last_connection, [65](#)
 - operator=, [64](#)
 - pid, [65](#)
- zmqutils::common::RequestData, [77](#)
 - command, [78](#)
 - params, [78](#)
 - params_size, [78](#)
 - RequestData, [78](#)
- zmqutils::utils, [22](#)
 - binarySerializeDeserialize, [23](#)
 - currentISO8601Date, [23](#)
 - getCurrentPID, [23](#)
 - getHostIPsWithInterfaces, [24](#)
 - getHostname, [24](#)
 - HRTIMEPOINTSTD, [23](#)
 - joinArraysConstexpr, [24](#)
 - SCTIMEPOINTSTD, [23](#)
 - timePointToISO8601, [24](#)
 - timePointToString, [24](#)
- zmqutils::utils::internal, [25](#)
 - joinArrays, [25](#)
- zmqutils::utils::NetworkAdapterInfo, [76](#)
 - descr, [76](#)
 - id, [76](#)
 - ip, [77](#)
 - name, [77](#)