LibZMQUtils

2307.3

Generated by Doxygen 1.9.7

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 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, GetHomePositionCallba
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	*data*	Pointer to the input data that needs to be serialized/deserialized.
in	*data_size_bytes*	Size of the input data in bytes.
out	*dest*	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 HRTimePointStd & 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 HRTimePointStd & 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.

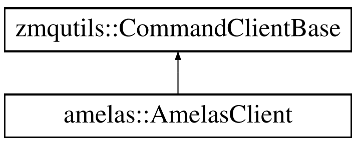
The documentation for this struct was generated from the following file:

- 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.

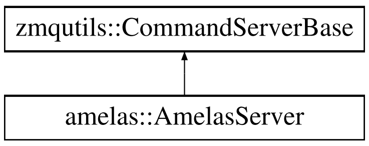
The documentation for this class was generated from the following file:

- 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::getServer←
Addresses ( ) 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

The	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 overrided 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

The	HostClient object representing the connected client.

Warning

The overrided 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	*The*	CommandRequest object representing the command execution request.
out	*The*	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

The	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

The	HostClient object representing the disconnected client.

Warning

> The overrided 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

The	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

The	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

The	`zmq::error_t` object representing the error that occurred.
Optional	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 overrided 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 zmqutils::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 overrided 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 zmqutils::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 overrided 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 zmqutils::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 overrided 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 zmqutils::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 zmqutils::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

The	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.

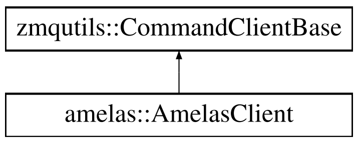
The documentation for this class was generated from the following files:

- 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:

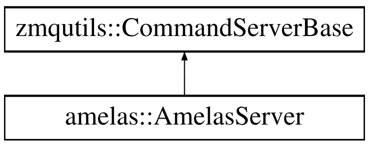
- includes/LibZMQUtils/CommandServerClient/common.h

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 sublcass, 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, getServer↩ Addresses, 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

port	The port number on which the server will listen for incoming requests.
local_addr	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 addresses.

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::getServer←
Addresses ( ) 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

The	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 overrided 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

The	HostClient object representing the connected client.

Warning

The overrided 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	*The*	CommandRequest object representing the command execution request.
out	*The*	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

The	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

The	HostClient object representing the disconnected client.

Warning

The overrided 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

The	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

The	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

The	`zmq::error_t` object representing the error that occurred.
Optional	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 overrided 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 overrided 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 overrided 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 overrided 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

The	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.

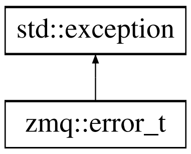
The documentation for this class was generated from the following file:

- 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

  Dinamic 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
```

Dinamic 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.

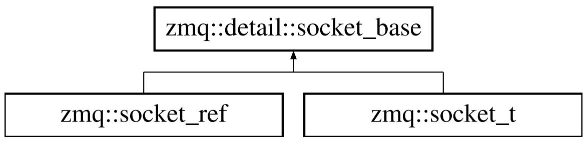
The documentation for this struct was generated from the following file:

- 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  bufferfor 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.

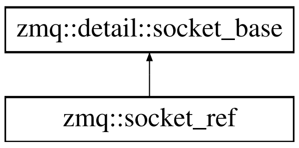
The documentation for this class was generated from the following file:

- 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  bufferfor 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.

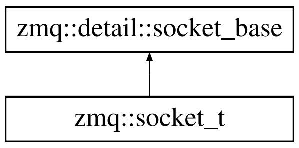
The documentation for this class was generated from the following file:

- 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/AmelasExample←Client/amelas_client.cpp File Reference

```
#include ¨amelas_client.h¨
```

Namespaces

- namespace amelas

7.2 amelas_client.cpp

Go to the documentation of this file.

```
#include "amelas_client.h"

// AMELAS NAMESPACES
// ==================================================================================================================
namespace amelas{
// ==================================================================================================================

using common::AmelasServerCommandStr;
using common::AmelasServerResultStr;
using common::ControllerError;
using common::AmelasServerCommand;
using common::AmelasServerResult;
using zmqutils::common::ServerCommand;
using zmqutils::common::ServerResult;
using zmqutils::common::ResultType;

AmelasClient::AmelasClient(const std::string &server_endpoint) :
    zmqutils::CommandClientBase(server_endpoint)
{}

void AmelasClient::onSendCommand(const RequestData &req, const zmq::multipart_t &msg)
{
    // Get the command string.
    std::string cmd_str;
    cmd_str = (req.command <AmelasServerCommandStr.size()) ? AmelasServerCommandStr[req.command] : "Unknown command";
    // Log.
    std::cout « std::string(80, '-') « std::endl;
    std::cout«"<AMELAS CLIENT>"«std::endl;
    std::cout«"-> ON SEND COMMAND: "«std::endl;
    std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
    std::cout«"Command: "«static_cast<int>(req.command)«std::endl;
    std::cout«"Params size: "«req.params_size«std::endl;
    std::cout«"Msg parts: "«msg.size()«std::endl;
    std::cout « std::string(80, '-') « std::endl;
}



} // END NAMESPACES.
// ==================================================================================================

```

7.3 examples/ExampleZMQCommandClientAmelas/AmelasExample↩ Client/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.

```

// ==================================================================================================
#pragma once
// ==================================================================================================

// C++ INCLUDES
// ==================================================================================================
#include <string>
// ==================================================================================================

// ZMQUTILS INCLUDES
// ==================================================================================================
#include <LibZMQUtils/CommandClient>
#include <LibZMQUtils/Utils>
// ==================================================================================================

// AMELAS INCLUDES
// ==================================================================================================
#include <AmelasExampleServer/amelas_server.h>
#include <AmelasExampleServer/common.h>
// ==================================================================================================

// PROJECT INCLUDES
// ==========================================================================================================
// ==========================================================================================================

// AMELAS NAMESPACES
// ==========================================================================================================
namespace amelas{
// ==========================================================================================================

using common::AmelasServerCommandStr;



class AmelasClient : public zmqutils::CommandClientBase
{
public:

    AmelasClient(const std::string &server_endpoint);

    // TODO
    //virtual void prepareRequest() = 0;

private:

    void onSendCommand(const RequestData& req, const zmq::multipart_t& msg) override;

    /* TODO
    void onConnected(const HostClient& client) override
    {
        // Log.
        std::cout « std::string(80, '-') « std::endl;
        std::cout«"<AMELAS SERVER>"«std::endl;
        std::cout«"-> ON CONNECTED: "«std::endl;
        std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
        std::cout«"Current Clients: "«this->getConnectedClients().size()«std::endl;
        std::cout«"Client Id: "«client.id«std::endl;
        std::cout«"Client Ip: "«client.ip«std::endl;
        std::cout«"Client Host: "«client.hostname«std::endl;
        std::cout«"Client Process: "«client.pid«std::endl;
        std::cout « std::string(80, '-') « std::endl;
    }
    */
};

} // END NAMESPACES.
// ==========================================================================================================
```

7.5 examples/ExampleZMQCommandClientAmelas/ExampleZMQClient↩ Amelas.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.

```


#include <iostream>
#include <cstring>

#include "AmelasExampleController/common.h"
#include "AmelasExampleServer/common.h"
#include "AmelasExampleClient/amelas_client.h"



using namespace zmqutils;
using namespace amelas;

using amelas::common::AmelasServerCommand;
using amelas::common::AmelasServerResult;
using zmqutils::common::CommandType;

void parseCommand(CommandClientBase &client, const std::string &command)
{
    zmqutils::common::ClientResult send_result = ClientResult::COMMAND_OK;

    char *command_str = new char[command.size()];
    std::copy(command.begin(), command.end(), command_str);

    char *token = std::strtok(command_str, " ");

    if (token)
    {
        CommandType command_id;

        try
        {
            command_id = static_cast<CommandType>(std::stoi(token));
        }
        catch (...)
        {
            std::cerr « "Failed at sending command." « std::endl;
            delete[] command_str;
            return;
        }

        RequestData command_msg(command_id);

        bool valid = true;

        if (command_id == static_cast<CommandType>(ServerCommand::REQ_CONNECT))
        {
            std::cout « "Sending connect message" « std::endl;
        }
        else if (command_id == static_cast<CommandType>(ServerCommand::REQ_DISCONNECT))
        {
            std::cout « "Sending disconnect message" « std::endl;
        }
        else if (command_id == static_cast<CommandType>(ServerCommand::REQ_ALIVE))
        {
            std::cout « "Sending keepalive command." « std::endl;
        }
        else if (command_id == static_cast<CommandType>(AmelasServerCommand::REQ_GET_DATETIME))
        {
            std::cout « "Get datetime command not implemented yet." « std::endl;
            valid = false;
        }
        else if (command_id == static_cast<CommandType>(AmelasServerCommand::REQ_SET_DATETIME))
        {
            std::cout « "Set datetime command not implemented yet." « std::endl;
            valid = false;
        }
        else if (command_id == static_cast<CommandType>(AmelasServerCommand::REQ_GET_HOME_POSITION))
        {
            std::cout « "Sending get home position command." « std::endl;
        }
        else if (command_id == static_cast<CommandType>(AmelasServerCommand::REQ_SET_HOME_POSITION))
        {
            std::cout « "Sending set home position command." « std::endl;

            bool valid_params = true;
            double az = 0., el = 0.;
            char *param_token = std::strtok(nullptr, " ");

            try
            {
                az = std::stod(param_token);
            }
            catch (...)
            {
                std::cerr « "Bad parameter azimuth issued.";
                valid_params = false;
            }

            if (valid_params)
            {
                param_token = std::strtok(nullptr, " ");

                try
                {
                    el = std::stod(param_token);
                }
                catch (...)
                {
                    std::cerr « "Bad parameter elevation issued.";
                    valid_params = false;
                }
            }

            if (valid_params)
            {
                std::cout«"Sending: " « az «" "«el«std::endl;

                command_msg.params = std::unique_ptr<std::uint8_t>(new std::uint8_t[16]);
                command_msg.params_size = 16;

                zmqutils::utils::binarySerializeDeserialize(&az, 8, command_msg.params.get());
                zmqutils::utils::binarySerializeDeserialize(&el, 8, command_msg.params.get() + 8);
            }

            valid = valid_params;

        }
        else
        {
            valid = false;
        }

        // TODO MOVE ALL OF THIS TO A SUBCLASS IN A PURE VIRTUAL. THE FUNCTION WILL RETURN
ClientResult
        // TODO THE ERROR CONTROL MUST BE IN THE BASE CLIENT. THE SUBCLASS MUST CONTROL THE OUTPUT
DATA AND CUSTOM ERRORS ONLY.
        // TODO DISABLE SEND WITH THIS WAY THE RESERVED COMMANDS.
        // TODO CREATE doConnect, doDisconnect, checkServerAlive
        // TODO CREATE IN THE CLIENT THE INTERNAL CALLBACKS LIKE THE SERVER.
        // TODO MOVE THE PROCESSING OF EACH COMPLEX RESPONSE TO A FUNCTION.

        if (valid)
        {
            // TODO MOVE ALL
            ClientResult result = ClientResult::COMMAND_OK;
            CommandReply reply;

            send_result = client.sendCommand(command_msg, reply);


            if (send_result != ClientResult::COMMAND_OK)
            {
                //std::cerr « "Command sending failed with code: " « send_result « std::endl;
            }
            else
            {
                constexpr std::size_t res_sz = sizeof(amelas::common::ControllerError);
                constexpr std::size_t double_sz = sizeof(double);

                std::cout«"Server result: "«static_cast<int>(reply.result)«std::endl;

                // Get the controller result.
                // TODO ERROR CONTROL



                if (command_id ==
static_cast<CommandType>(AmelasServerCommand::REQ_GET_HOME_POSITION))
                {
                    if (reply.params_size == (res_sz + 2*double_sz))
                    {
                        amelas::common::ControllerError error;
                        double az;
                        double el;

                        // Deserialize the parameters.
                        zmqutils::utils::binarySerializeDeserialize(reply.params.get(), res_sz,
reply.params.get());
                        zmqutils::utils::binarySerializeDeserialize(reply.params.get() + res_sz,
double_sz, &az);
                        zmqutils::utils::binarySerializeDeserialize(reply.params.get() + res_sz +
double_sz, double_sz, &el);

                        // Generate the struct.
                        std::cout«"Controller error: "«static_cast<int>(error)«std::endl;
                        std::cout«"Az: "«az«std::endl;
                        std::cout«"El: "«el«std::endl;
                    }
                    else
                    {
                        std::cout«"BAD PARAMS"«std::endl;
                        // RETURN BAD PARAMS
                        //result = ClientResult::
                    }
                }
            }
        }
        else
        {
            std::cerr « "Command is not implemented or valid" « std::endl;
        }

    }
    else
    {
        std::cerr « "Not a valid command" « std::endl;
    }


    delete[] command_str;
}


int main(int argc, char**argv)
{

    int port = 9999;
    std::string ip = "127.0.0.1";

    if (argc == 2)
    {
        ip = argv[1];
    }
    if (argc == 3)
    {
        ip = argv[1];
        try
        {
            port = std::stoi(argv[2]);
        }  catch (...)
        {
            std::cerr « "Not recognized port in input: " « argv[2] « std::endl;
            return -1;
        }

    }
    else if (argc > 3)
    {
        std::cout « "Usage: ZMQClient [ip] [port]" « std::endl;
        return 0;
    }

    std::string endpoint = "tcp://" + ip + ":" + std::to_string(port);
    AmelasClient client(endpoint);
    client.startClient("Ethernet");
    //client.setClientHostIP("");
    std::cout « "Connecting to endpoint: " «  endpoint « std::endl;
    //client.startAutoAlive();
    std::string command;

    while(true)
    {
        std::cout«"Write a command: ";
        std::getline(std::cin, command);

        if (command == "exit")
            break;

        parseCommand(client, command);
    }

    std::cout « "Requested client to stop. Bye." « std::endl;

    client.stopClient();


    return 0;
}
```

7.7 examples/ExampleZMQCommanServerAmelas/AmelasExample↩ Controller/amelas_controller.h File Reference

```
#include <map>
#include <string>
#include <LibZMQUtils/CommandServer>
#include <LibZMQUtils/Utils>
#include ¨common.h¨
```

Classes

- class amelas::AmelasController

Namespaces

- namespace amelas

7.8 amelas_controller.h

Go to the documentation of this file.

```
/*****************************************************************************************************
 *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
 *
 *                                                                                                    *
 *   Copyright (C) 2023 Degoras Project Team                                                          *
 *                       < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >           *
 *                       < Jesús Relinque Madroñal >                                                  *
 *                                                                                                    *
 *   This file is part of LibZMQUtils.                                                                *
 *                                                                                                    *
 *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of the EUPL license   *
 *   as soon they will be approved by the European Commission (IDABC).                                *
 *                                                                                                    *
 *   This project is free software: you can redistribute it and/or modify it under the terms of the EUPL license as    *
 *   published by the IDABC, either Version 1.2 or, at your option, any later version.                *
 *                                                                                                    *
 *   This project is distributed in the hope that it will be useful. Unless required by applicable law or agreed to in *
 *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND; without even the  *
 *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to check specific   *
 *   language governing permissions and limitations and more details.                                 *
 *                                                                                                    *
 *   You should use this project in compliance with the EUPL license. You should have received a copy of the license   *
 *   along with this project. If not, see the license at < https://eupl.eu/ >.                        *
 **********************************************************************************************************************/

// C++ INCLUDES
// =====================================================================================================================
#include <map>
#include <string>
// =====================================================================================================================

// ZMQUTILS INCLUDES
// =====================================================================================================================
#include <LibZMQUtils/CommandServer>
#include <LibZMQUtils/Utils>
// =====================================================================================================================

// PROJECT INCLUDES
// =====================================================================================================================
#include "common.h"
// =====================================================================================================================



// AMELAS NAMESPACES
// =====================================================================================================================
namespace amelas{
// =====================================================================================================================

using amelas::common::ControllerError;
using amelas::common::AltAzPos;

class AmelasController
{
public:


    AmelasController() :
        home_pos_({-1,-1})
    {}

    ControllerError setHomePosition(const AltAzPos& pos)
    {
        // Auxiliar result.
        ControllerError error = ControllerError::SUCCESS;

        // Check the provided values.
        if (pos.az >= 360.0 ||  pos.az < 0.0 || pos.el >= 90. || pos.el < 0.)
        {
            error = ControllerError::INVALID_POSITION;
        }
        else
        {
            this->home_pos_ = pos;
        }

        std::cout « std::string(100, '-') « std::endl;
        std::cout«"<AMELAS CONTROLLER>"«std::endl;
        std::cout«"-> SET_HOME_POSITION"«std::endl;
        std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
        std::cout«"Az: "«pos.az«std::endl;
        std::cout«"El: "«pos.el«std::endl;
        std::cout « std::string(100, '-') « std::endl;

        return error;
    }

    ControllerError getHomePosition(AltAzPos& pos)
    {
        pos = this->home_pos_;

        std::cout « std::string(100, '-') « std::endl;
        std::cout«"<AMELAS CONTROLLER>"«std::endl;
        std::cout«"-> GET_HOME_POSITION"«std::endl;
        std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
        std::cout « std::string(100, '-') « std::endl;

        return ControllerError::SUCCESS;
    }

    ControllerError getDatetime(std::string&)
    {
        return ControllerError::SUCCESS;
    }

private:

    AltAzPos home_pos_;

};

} // END NAMESPACES.
// ==================================================================================================================

```

7.9 examples/ExampleZMQCommanServerAmelas/AmelasExample↵ Controller/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.

```

//
===================================================================================================
#pragma once
//
===================================================================================================

// C++ INCLUDES
//
===================================================================================================
#include <string>
#include <map>
#include <vector>
#include <variant>
#include <functional>
//
===================================================================================================

// AMELAS NAMESPACES
//
===================================================================================================
namespace amelas{
namespace common{
//
===================================================================================================

// CONSTANTS
//
===================================================================================================

//
===================================================================================================

// CONVENIENT ALIAS, ENUMERATIONS AND CONSTEXPR
//
===================================================================================================

enum class ControllerError : std::uint32_t
{
    SUCCESS = 0,
    INVALID_POSITION = 1,
    UNSAFE_POSITION = 2
};

struct AltAzPos
{
    AltAzPos(double az, double el):
        az(az), el(el){}

    AltAzPos(): az(-1), el(-1){}

    double az;
    double el;
};

// Callback function type aliases
using SetHomePositionCallback = std::function<ControllerError(const AltAzPos&)>;
using GetHomePositionCallback = std::function<ControllerError(AltAzPos&)>;
using GetDatetimeCallback = std::function<ControllerError(std::string&)>;

// Callback variant.
using ControllerCallback = std::variant<SetHomePositionCallback,
                                        GetHomePositionCallback,
                                        GetDatetimeCallback>;





// 
==================================================================================================================

}} // END NAMESPACES.
// 
==================================================================================================================
```

7.11 examples/ExampleZMQCommanServerAmelas/AmelasExample↩ Server/common.h File Reference

```
#include <functional>
#include <any>
#include <LibZMQUtils/Utils>
```

Namespaces

- namespace amelas
- namespace amelas::common

Enumerations

- enum class amelas::common::AmelasServerCommand : zmqutils::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 : zmqutils::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>(zmqutils::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.

```

#include <functional>
#include <any>

#include <LibZMQUtils/Utils>

// =================================================================================================================
#pragma once
// =================================================================================================================

// AMELAS NAMESPACES
// =================================================================================================================
namespace amelas{
namespace common{
// =================================================================================================================

// Specific subclass commands (0 to 4 are reserved for the base server).
// WARNING: In our approach, the server commands must be always in order.
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
};

// Specific subclass errors (0 to 20 are reserved for the base server).
enum class AmelasServerResult : zmqutils::common::ResultType
{
    EMPTY_CALLBACK = 21,
    INVALID_CALLBACK = 22
};

// Extend the base command strings with those of the subclass.
static constexpr auto AmelasServerCommandStr = 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"
    });

// Extend the base result strings with those of the subclass.
static constexpr auto AmelasServerResultStr = 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."
    });

// Usefull const expressions.
constexpr int kMinCmdId = static_cast<int>(zmqutils::common::ServerCommand::END_BASE_COMMANDS) + 1;
constexpr int kMaxCmdId = static_cast<int>(AmelasServerCommand::END_AMELAS_COMMANDS) - 1;

}} // END NAMESPACES.
// =================================================================================================================
```

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_DISCONNE
  = 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_MSG
  = 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_MSG
  = 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_MSG
  = 2 , zmqutils::common::EMPTY_PARAMS = 6 ,
  zmqutils::common::TIMEOUT_REACHED = 7 , zmqutils::common::INVALID_PARTS = 8 , zmqutils::common::INVALID_MSG
  = 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.

```
/*****************************************************************************************************************
 *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
*
 *
*
 *   Copyright (C) 2023 Degoras Project Team
*
 *                       < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
*
 *                       < Jesús Relinque Madroñal >
*
 *
*
 *   This file is part of LibZMQUtils.
*
 *
*
 *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
the EUPL license   *
 *   as soon they will be approved by the European Commission (IDABC).
*
```

```
 *                                                                                                           *
 *   This project is free software: you can redistribute it and/or modify it under the terms of the EUPL license as    *
 *   published by the IDABC, either Version 1.2 or, at your option, any later version.                       *
 *                                                                                                           *
 *   This project is distributed in the hope that it will be useful. Unless required by applicable law or agreed to in *
 *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND; without even the  *
 *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to check specific   *
 *   language governing permissions and limitations and more details.                                        *
 *                                                                                                           *
 *   You should use this project in compliance with the EUPL license. You should have received a copy of the license   *
 *   along with this project. If not, see the license at < https://eupl.eu/ >.                              *
**********************************************************************************************************************/

// =====================================================================================================================
#pragma once
// =====================================================================================================================

// C++ INCLUDES
// =====================================================================================================================
#include <string>
#include <iostream>
#include <map>
#include <vector>
#include <cstring>
#include <memory>
#include <zmq/zmq.hpp>
#include <zmq/zmq_addon.hpp>
// =====================================================================================================================

// ZMQUTILS INCLUDES
// =====================================================================================================================
#include "LibZMQUtils/libzmqutils_global.h"
#include "LibZMQUtils/utils.h"
// =====================================================================================================================

// ZMQUTILS NAMESPACES
// =====================================================================================================================
namespace zmqutils{
namespace common{
// =====================================================================================================================

// CONSTANTS
// =====================================================================================================================
constexpr int kDefaultClientAliveTimeoutMsec = 8000;
constexpr int kDefaultServerAliveTimeoutMsec = 3000;
constexpr unsigned kServerReconnTimes = 10;
constexpr unsigned kClientAlivePeriodMsec = 1000;
constexpr int kZmqEFSMError = 156384765;
// =====================================================================================================================

// CONVENIENT ALIAS, ENUMERATIONS AND CONSTEXPR
// =====================================================================================================================

using CommandType = std::uint32_t;
using ResultType = std::uint32_t;

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
};


// TODO MORE CASES RELATED TO THE CLIENT
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

};

// Usefull const expressions.
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
{
    "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"
};

static constexpr std::array<const char*, 21>  ServerResultStr
{
    "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"
};

//
============================================================================================================

// COMMON STRUCTS
// ===================================================================================================

struct LIBZMQUTILS_EXPORT HostClient
{
    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 = "");

    // Struct members.
    std::string id;
    std::string ip;
    std::string hostname;
    std::string pid;
    std::string info;
    utils::SCTimePointStd last_connection;
};

struct CommandRequest
{
    CommandRequest():
        command(ServerCommand::INVALID_COMMAND),
        params(nullptr),
        params_size(0)
    {}

    HostClient client;
    ServerCommand command;
    std::unique_ptr<std::uint8_t> params;
    zmq::multipart_t raw_msg;
    size_t params_size;
};

struct CommandReply
{
    CommandReply():
        params(nullptr),
        params_size(0),
        result(ServerResult::COMMAND_OK)
    {}

    std::unique_ptr<std::uint8_t> params;
    zmq::multipart_t raw_msg;
    size_t params_size;
    ServerResult result;
};

struct LIBZMQUTILS_EXPORT RequestData
{
    RequestData(CommandType id) :
        command(id),
        params(nullptr),
        params_size(0){}

    RequestData() :
        command(static_cast<CommandType>(ServerCommand::INVALID_COMMAND)),
        params(nullptr),
        params_size(0){}

    CommandType command;
    std::unique_ptr<std::uint8_t> params;
    size_t params_size;
};

// ===================================================================================================

}} // END NAMESPACES.
// ===================================================================================================
```

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.

```
//
==================================================================================================
#pragma once
//
==================================================================================================

// C++ INCLUDES
//
==================================================================================================
#include <string>
#include <map>
#include <vector>
#include <functional>
//
==================================================================================================

// AMELAS NAMESPACES
//
==================================================================================================
namespace amelas{
namespace utils{
//
==================================================================================================

template<typename ClassType, typename ReturnType, typename... Args>
static std::function<ReturnType(Args...)> makeCallback(ClassType* object,

ReturnType(ClassType::*memberFunction)(Args...))
{
    return [object, memberFunction](Args... args) -> ReturnType
    {
        return (object->*memberFunction)(std::forward<Args>(args)...);
    };
}

}} // END NAMESPACES.
//
==================================================================================================
```

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↩ _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.

```
/***********************************************************************************************************************
 *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.                          *
 *                                                                                                                     *
 *   Copyright (C) 2023 Degoras Project Team                                                                           *
 *                      < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >                             *
 *                      < Jesús Relinque Madroñal >                                                                    *
 *                                                                                                                     *
 *   This file is part of LibZMQUtils.                                                                                 *
 *                                                                                                                     *
 *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of the EUPL license   *
 *   as soon they will be approved by the European Commission (IDABC).                                                 *
 *                                                                                                                     *
 *   This project is free software: you can redistribute it and/or modify it under the terms of the EUPL license as    *
 *   published by the IDABC, either Version 1.2 or, at your option, any later version.                                 *
 *                                                                                                                     *
 *   This project is distributed in the hope that it will be useful. Unless required by applicable law or agreed to in *
 *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND; without even the  *
 *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to check specific   *
 *   language governing permissions and limitations and more details.                                                  *
 *                                                                                                                     *
 *   You should use this project in compliance with the EUPL license. You should have received a copy of the license  *
 *   along with this project. If not, see the license at < https://eupl.eu/ >.                                         *
 **********************************************************************************************************************/

// =====================================================================================================================
#pragma once
// =====================================================================================================================

// C++ INCLUDES
// =====================================================================================================================
#include <algorithm>
#include <string>
#include <iostream>
```

```cpp
#include <map>
#include <vector>
#include <cstring>
#include <chrono>
#include <array>
#include <utility>
// ===================================================================================================

// ZMQUTILS INCLUDES
// ===================================================================================================
#include "LibZMQUtils/libzmqutils_global.h"
// ===================================================================================================

// DEFINITIONS
// ===================================================================================================
#if defined(__MINGW32__) || defined(_MSC_VER)
#define MKGMTIME _mkgmtime
#else
#define MKGMTIME timegm
#endif
// ===================================================================================================

// ZMQUTILS NAMESPACES
// ===================================================================================================
namespace zmqutils{
namespace utils{
// ===================================================================================================

// CONVENIENT ALIAS AND ENUMERATIONS
// ===================================================================================================
using HRTimePointStd = std::chrono::time_point<std::chrono::high_resolution_clock>;
using SCTimePointStd =  std::chrono::steady_clock::time_point;
// ===================================================================================================

struct LIBZMQUTILS_EXPORT NetworkAdapterInfo
{
    std::string id;
    std::string name;
    std::string descr;
    std::string ip;
};

LIBZMQUTILS_EXPORT void binarySerializeDeserialize(const void* data, size_t data_size_bytes, void* dest);

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);

namespace internal
{
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...>)
{
    return { a1[Is1]..., a2[Is2]... };
}
}

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)
{
    return internal::joinArrays(a1, a2, std::make_index_sequence<N1>(), std::make_index_sequence<N2>());
}

}} // END NAMESPACES.
//
================================================================================================================
```

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.

```
#include "amelas_server.h"

// AMELAS NAMESPACES
//
================================================================================================================
namespace amelas{
//
================================================================================================================

using common::AmelasServerCommandStr;
using common::AmelasServerResultStr;
using common::ControllerError;
using common::AmelasServerCommand;
using common::AmelasServerResult;
using zmqutils::common::ServerCommand;
using zmqutils::common::ServerResult;
using zmqutils::common::ResultType;


AmelasServer::AmelasServer(unsigned int port, const std::string &local_addr) :
    CommandServerBase(port, local_addr)
{}

const std::map<AmelasServerCommand, common::ControllerCallback> &AmelasServer::getCallbackMap() const
{
    return this->callback_map_;
}

void AmelasServer::setCallback(common::AmelasServerCommand command, common::ControllerCallback
callback)
{
    callback_map_[command] = callback;
}

void AmelasServer::removeCallback(common::AmelasServerCommand command)
{
    this->callback_map_.erase(command);
}

bool AmelasServer::isCallbackSet(common::AmelasServerCommand command) const
{
    return this->callback_map_.find(command) != this->callback_map_.end();
}

void AmelasServer::clearCallbacks()
{
    this->callback_map_.clear();
}

void AmelasServer::processSetHomePosition(const CommandRequest& request, CommandReply& reply)
{
    // Command and error.
    AmelasServerCommand cmd = AmelasServerCommand::REQ_SET_HOME_POSITION;
    ControllerError controller_err;

    // Auxilar variables.
    double az, el;
    constexpr std::size_t double_sz = sizeof(double);
    bool result;

    // Check the request parameters size.
    if (request.params_size == 0)
    {
        reply.result = ServerResult::EMPTY_PARAMS;
        return;
    }
    else if (request.params_size != double_sz*2)
    {
        reply.result = ServerResult::BAD_PARAMETERS;
        return;
    }

    // Deserialize the parameters.
    zmqutils::utils::binarySerializeDeserialize(request.params.get(), double_sz, &az);
    zmqutils::utils::binarySerializeDeserialize(request.params.get() + double_sz, double_sz, &el);

    // Generate the struct.
    common::AltAzPos pos = {az, el};

    // Process the command.
    // Check the callback.
    if(!this->isCallbackSet(cmd))
    {
        reply.result = static_cast<ServerResult>(AmelasServerResult::EMPTY_CALLBACK);
        return;
    }

    // Process the command.
    try{controller_err = this->invokeCallback<common::SetHomePositionCallback>(cmd, pos);}
    catch(...)
    {
        reply.result = static_cast<ServerResult>(AmelasServerResult::INVALID_CALLBACK);
        return;
    }

    // Store the amelas error.
    reply.params = std::unique_ptr<std::uint8_t>(new std::uint8_t[sizeof(ResultType)]);
    ResultType amelas_res = static_cast<ResultType>(controller_err);
    zmqutils::utils::binarySerializeDeserialize(&amelas_res, sizeof(ResultType), reply.params.get());
    reply.params_size = sizeof(ResultType);
}

void AmelasServer::processGetHomePosition(const CommandRequest &, CommandReply &reply)
{
    // Command and error.
    AmelasServerCommand cmd = AmelasServerCommand::REQ_GET_HOME_POSITION;
    ControllerError controller_err;

    // Auxilar variables.
    constexpr std::size_t res_sz = sizeof(ControllerError);
    constexpr std::size_t double_sz = sizeof(double);
    ControllerError amelas_err = ControllerError::SUCCESS;
    common::AltAzPos pos;

    // Process the command.
    try{controller_err = this->invokeCallback<common::GetHomePositionCallback>(cmd, pos);}
    catch(...)
    {
        reply.result = static_cast<ServerResult>(AmelasServerResult::INVALID_CALLBACK);
        return;
    }

    // Serialize parameters
    reply.params = std::unique_ptr<std::uint8_t>(new std::uint8_t[res_sz + 2*double_sz]);
    reply.params_size = res_sz + 2*double_sz;
    zmqutils::utils::binarySerializeDeserialize(&amelas_err, res_sz, reply.params.get());
    zmqutils::utils::binarySerializeDeserialize(&pos.az, double_sz, reply.params.get() + res_sz);
    zmqutils::utils::binarySerializeDeserialize(&pos.el, double_sz, reply.params.get() + res_sz +
double_sz);

    // Store the server result.
    reply.result = ServerResult::COMMAND_OK;
}

void AmelasServer::processAmelasCommand(const CommandRequest& request, CommandReply& reply)
{
    AmelasServerCommand command = static_cast<AmelasServerCommand>(request.command);

    if(command == AmelasServerCommand::REQ_SET_HOME_POSITION)
    {
        this->processSetHomePosition(request, reply);
    }
    else if (command == AmelasServerCommand::REQ_GET_HOME_POSITION)
    {
        this->processGetHomePosition(request, reply);
    }
    else
    {
        reply.result = ServerResult::NOT_IMPLEMENTED;
    }
}

void AmelasServer::onCustomCommandReceived(const CommandRequest& request, CommandReply& reply)
{
    // Get the command.
    AmelasServerCommand command = static_cast<AmelasServerCommand>(request.command);

    // Get the command string.
    std::string cmd_str;
    std::uint32_t cmd_uint = static_cast<std::uint32_t>(request.command);
    cmd_str = (cmd_uint < AmelasServerCommandStr.size()) ? AmelasServerCommandStr[cmd_uint] : "Unknown
command";

    // Log the command.
    std::cout « std::string(100, '-') « std::endl;
    std::cout«"ON CUSTOM COMMAND RECEIVED: "«std::endl;
    std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
    std::cout«"Client Id: "«request.client.id«std::endl;
    std::cout«"Command: "«cmd_uint«" ("«cmd_str«")"«std::endl;
    std::cout « std::string(100, '-') « std::endl;

    // Process the command if it is implemented.
    if(command == AmelasServerCommand::END_AMELAS_COMMANDS)
    {
        // Update the result.
        reply.result = ServerResult::INVALID_MSG;
    }
    else if(AmelasServer::validateAmelasCommand(command))
    {
        this->processAmelasCommand(request, reply);
    }
    else
    {
        // Call to the base function.
        CommandServerBase::onCustomCommandReceived(request, reply);
    }
}

void AmelasServer::onServerStart()
{
    // Ips.
    std::string ips;

    // Get listen interfaces ips.
    for(const auto& intrfc : this->getServerAddresses())
    {
        ips.append(intrfc.ip);
        ips.append(" - ");
    }
    ips.pop_back();
    ips.pop_back();

    // Log.
    std::cout « std::string(100, '-') « std::endl;
    std::cout«"<AMELAS SERVER>"«std::endl;
    std::cout«"-> ON SERVER START: "«std::endl;
    std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
    std::cout«"Addresses: "«ips«std::endl;
    std::cout«"Port: "«this->getServerPort()«std::endl;
    std::cout « std::string(100, '-') « std::endl;
}

void AmelasServer::onServerStop()
{
    // Log.
    std::cout « std::string(100, '-') « std::endl;
    std::cout«"<AMELAS SERVER>"«std::endl;
    std::cout«"-> ON SERVER CLOSE: "«std::endl;
    std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
    std::cout « std::string(100, '-') « std::endl;
}

void AmelasServer::onWaitingCommand()
{
    // Log.
    std::cout « std::string(100, '-') « std::endl;
    std::cout«"<AMELAS SERVER>"«std::endl;
    std::cout«"-> ON WAITING COMMAND: "«std::endl;
    std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
    std::cout « std::string(100, '-') « std::endl;
}

```

```
void AmelasServer::onDeadClient(const HostClient& client)
{
    // Log.
    std::cout « std::string(100, '-') « std::endl;
    std::cout«"<AMELAS SERVER>"«std::endl;
    std::cout«"-> ON DEAD CLIENT: "«std::endl;
    std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
    std::cout«"Current Clients: "«this->getConnectedClients().size()«std::endl;
    std::cout«"Client Id: "«client.id«std::endl;
    std::cout«"Client Ip: "«client.ip«std::endl;
    std::cout«"Client Host: "«client.hostname«std::endl;
    std::cout«"Client Process: "«client.pid«std::endl;
    std::cout « std::string(100, '-') « std::endl;
}

void AmelasServer::onConnected(const HostClient& client)
{
    // Log.
    std::cout « std::string(100, '-') « std::endl;
    std::cout«"<AMELAS SERVER>"«std::endl;
    std::cout«"-> ON CONNECTED: "«std::endl;
    std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
    std::cout«"Current Clients: "«this->getConnectedClients().size()«std::endl;
    std::cout«"Client Id: "«client.id«std::endl;
    std::cout«"Client Ip: "«client.ip«std::endl;
    std::cout«"Client Host: "«client.hostname«std::endl;
    std::cout«"Client Process: "«client.pid«std::endl;
    std::cout « std::string(100, '-') « std::endl;
}

void AmelasServer::onDisconnected(const HostClient& client)
{
    // Log.
    std::cout « std::string(100, '-') « std::endl;
    std::cout«"<AMELAS SERVER>"«std::endl;
    std::cout«"-> ON DISCONNECTED: "«std::endl;
    std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
    std::cout«"Current Clients: "«this->getConnectedClients().size()«std::endl;
    std::cout«"Client Id: "«client.id«std::endl;
    std::cout«"Client Ip: "«client.ip«std::endl;
    std::cout«"Client Host: "«client.hostname«std::endl;
    std::cout«"Client Process: "«client.pid«std::endl;
    std::cout « std::string(100, '-') « std::endl;
}

void AmelasServer::onServerError(const zmq::error_t &error, const std::string &ext_info)
{
    // Log.
    std::cout « std::string(100, '-') « std::endl;
    std::cout«"<AMELAS SERVER>"«std::endl;
    std::cout«"-> ON SERVER ERROR: "«std::endl;
    std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
    std::cout«"Code: "«error.num()«std::endl;
    std::cout«"Error: "«error.what()«std::endl;
    std::cout«"Info: "«ext_info«std::endl;
    std::cout « std::string(100, '-') « std::endl;
}

void AmelasServer::onCommandReceived(const CommandRequest &cmd_req)
{
    // Get the command string.
    std::string cmd_str;
    std::uint32_t command = static_cast<std::uint32_t>(cmd_req.command);
    cmd_str = (command < AmelasServerCommandStr.size()) ? AmelasServerCommandStr[command] : "Unknown
command";
    // Log.
    std::cout « std::string(100, '-') « std::endl;
    std::cout«"<AMELAS SERVER>"«std::endl;
    std::cout«"-> ON COMMAND RECEIVED: "«std::endl;
    std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
    std::cout«"Client Id: "«cmd_req.client.id«std::endl;
    std::cout«"Command: "«command«" ("«cmd_str«")"«std::endl;
    std::cout « std::string(100, '-') « std::endl;
}

void AmelasServer::onInvalidMsgReceived(const CommandRequest &cmd_req)
{
    // Log.
    std::cout « std::string(100, '-') « std::endl;
    std::cout«"<AMELAS SERVER>"«std::endl;
    std::cout«"-> ON BAD COMMAND RECEIVED: "«std::endl;
    std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
    std::cout«"Raw Str: "«cmd_req.raw_msg.str()«std::endl;
    std::cout«"Client Id: "«cmd_req.client.id«std::endl;
    std::cout«"Client Ip: "«cmd_req.client.ip«std::endl;
    std::cout«"Client Host: "«cmd_req.client.hostname«std::endl;
    std::cout«"Client Process: "«cmd_req.client.pid«std::endl;
    std::cout«"Command: "«static_cast<int>(cmd_req.command)«std::endl;
    std::cout«"Params Size: "«cmd_req.params_size«std::endl;
    std::cout « std::string(100, ’-’) « std::endl;
}

void AmelasServer::onSendingResponse(const CommandReply &cmd_rep)
{
    // Log.
    int result = static_cast<int>(cmd_rep.result);
    std::cout « std::string(100, ’-’) « std::endl;
    std::cout«"<AMELAS SERVER>"«std::endl;
    std::cout«"-> ON SENDING RESPONSE: "«std::endl;
    std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
    std::cout«"Result: "«result«" ("«AmelasServerResultStr[result]«")"«std::endl;
    std::cout«"Params Size: "«cmd_rep.params_size«std::endl;
    std::cout « std::string(100, ’-’) « std::endl;
}

bool AmelasServer::validateAmelasCommand(AmelasServerCommand command)
{
    // Auxiliar variables.
    bool result = false;
    zmqutils::common::CommandType cmd = static_cast<zmqutils::common::CommandType>(command);
    // Check if the command is within the range of implemented custom commands.
    if (cmd >= common::kMinCmdId && cmd <= common::kMaxCmdId)
        result = true;
    return result;
}

} // END NAMESPACES.
//
=====================================================================================================

```

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.

```
//
=====================================================================================================
#pragma once
//
=====================================================================================================

// C++ INCLUDES
//
=====================================================================================================
#include <unordered_map>
#include <string>
#include <any>
#include <variant>
```

```
//
================================================================================================================

// ZMQUTILS INCLUDES
//
================================================================================================================
#include <LibZMQUtils/CommandServer>
#include <LibZMQUtils/Utils>
//
================================================================================================================

// PROJECT INCLUDES
//
================================================================================================================
#include "AmelasExampleController/common.h"
#include "AmelasExampleController/utils.h"
#include "common.h"
//
================================================================================================================

// AMELAS NAMESPACES
//
================================================================================================================
namespace amelas{
//
================================================================================================================

using namespace zmqutils;

// Example of creating a command server from the base.
class AmelasServer : public CommandServerBase
{
public:

    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...))
    {
        callback_map_[command] = utils::makeCallback(object, callback);
    }

    // Removes a callback for a command
    void removeCallback(common::AmelasServerCommand);

    // Clears all the callbacks
    void clearCallbacks();

    // Checks if a callback is set for a command
    bool isCallbackSet(common::AmelasServerCommand) const;

private:

    template <typename CallbackType, typename... Args>
    common::ControllerError invokeCallback(common::AmelasServerCommand command, Args&&... args)
    {
        if (auto callback = std::get_if<CallbackType>(&callback_map_[command]))
        {
            return (*callback)(std::forward<Args>(args)...);
        }
        throw std::runtime_error("Invalid command or incorrect callback type");
    }

    // Helper to check if the custom command is valid.
    static bool validateAmelasCommand(common::AmelasServerCommand command);

    // Process the specific commands.
    void processAmelasCommand(const CommandRequest&, CommandReply&);
    void processSetHomePosition(const CommandRequest&, CommandReply&);
    void processGetHomePosition(const CommandRequest&, CommandReply&);

    // Internal overrided custom command received callback.
    // WARNING The most important part.
    virtual void onCustomCommandReceived(const CommandRequest&, CommandReply&) final;

    // Internal overrided start callback.
    virtual void onServerStart() final;

    // Internal overrided close callback.
    virtual void onServerStop() final;

    // Internal waiting command callback.
    virtual void onWaitingCommand() final;

    // Internal dead client callback.
    virtual void onDeadClient(const HostClient&) final;

    // Internal overrided connect callback.
    virtual void onConnected(const HostClient&) final;

    // Internal overrided disconnect callback.
    virtual void onDisconnected(const HostClient&) final;

    // Internal overrided command received callback.
    virtual void onCommandReceived(const CommandRequest&) final;

    // Internal overrided bad command received callback.
    virtual void onInvalidMsgReceived(const CommandRequest&) final;

    // Internal overrided sending response callback.
    virtual void onSendingResponse(const CommandReply&) final;

    // Internal overrided server error callback.
    virtual void onServerError(const zmq::error_t&, const std::string& ext_info) final;

    // External callbacks map.
    std::map<common::AmelasServerCommand, common::ControllerCallback> callback_map_;
};

} // END NAMESPACES.
//
=============================================================================================================
```

7.23 examples/ExampleZMQCommanServerAmelas/Example←ZMQServerAmelas.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
```

```
// TODO
#endif

// -----------------------------------------------------------------------------------------------------------------------

// Main function.
//
// In the main we will create an AmelasController and an AmelasServer that will
// work together thanks to the callbacks. For safe finish, press ctrl-c.
//
int main(int argc, char**argv)
{
    // Using.
    using amelas::common::AmelasServerCommand;

    // Set up the Windows Console Control Handler
    SetConsoleCtrlHandler(ConsoleCtrlHandler, TRUE);

    // Configuration variables.
    unsigned port = 9999;
    bool client_status_check = false;

    // Get the port.
    if (argc == 2)
    {
        try
        {
           port = std::stoul(argv[1]);
        }  catch (...)
        {
            std::cerr « "Not recognized port in input: " « argv[1] « std::endl;
            return -1;
        }

    }
    else if (argc > 2)
    {
        std::cout « "Usage: ZMQServer [port]" « std::endl;
        return 0;
    }

    // Instantiate the Amelas controller.
    amelas::AmelasController amelas_controller;

    // Instantiate the server.
    amelas::AmelasServer amelas_server(port);

    // Disable or enables the client status checking.
    amelas_server.setClientStatusCheck(client_status_check);

    // ----------------------------------------
    // Set the controller callbacks in the server.

    amelas_server.setCallback(AmelasServerCommand::REQ_SET_HOME_POSITION,
                              &amelas_controller,
                              &amelas::AmelasController::setHomePosition);

    amelas_server.setCallback(AmelasServerCommand::REQ_GET_HOME_POSITION,
                              &amelas_controller,
                              &amelas::AmelasController::getHomePosition);

    // ----------------------------------------

    // Start the server.
    amelas_server.startServer();

    // Use the condition variable as an infinite loop until ctrl-c.
    std::unique_lock<std::mutex> lock(gMtx);
    gExitCv.wait(lock, [] { return gSignInterrupt == 1; });

    // Stop the server and wait the future.
    amelas_server.stopServer();

    // Final log.
    std::cout « "Server stoped. Press Enter to exit!" « std::endl;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(),'\n');

    // Return.
    return 0;
}

// -----------------------------------------------------------------------------------------------------------------------
```

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_VERBOSER 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_NAMETYPE 90
- #define ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAMETYPE 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_HANDOVER

```
#define ZMQ_ROUTER_HANDOVER 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_KEEPALIVE

```
#define ZMQ_TCP_KEEPALIVE 34
```

Definition at line 333 of file zmq.h.

7.25.1.172 ZMQ_TCP_KEEPALIVE_CNT

```
#define ZMQ_TCP_KEEPALIVE_CNT 35
```

Definition at line 334 of file zmq.h.

7.25.1.173 ZMQ_TCP_KEEPALIVE_IDLE

```
#define ZMQ_TCP_KEEPALIVE_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_VERBOSER

```
#define ZMQ_XPUB_VERBOSER 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.

```
/*
    Copyright (c) 2007-2016 Contributors as noted in the AUTHORS file

    This file is part of libzmq, the ZeroMQ core engine in C++.

    libzmq is free software; you can redistribute it and/or modify it under
    the terms of the GNU Lesser General Public License (LGPL) as published
    by the Free Software Foundation; either version 3 of the License, or
    (at your option) any later version.

    As a special exception, the Contributors give you permission to link
    this library with independent modules to produce an executable,
    regardless of the license terms of these independent modules, and to
    copy and distribute the resulting executable under terms of your choice,
    provided that you also meet, for each linked independent module, the
    terms and conditions of the license of that module. An independent
    module is a module which is not derived from or based on this library.
    If you modify this library, you must extend this exception to your
    version of the library.

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

    You should have received a copy of the GNU Lesser General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

    *************************************************************************
    NOTE to contributors. This file comprises the principal public contract
    for ZeroMQ API users. Any change to this file supplied in a stable
    release SHOULD not break existing applications.
    In practice this means that the value of constants must not change, and
    that old values may not be reused for new constants.
    *************************************************************************
*/

#ifndef __ZMQ_H_INCLUDED__
#define __ZMQ_H_INCLUDED__

/*  Version macros for compile-time API version detection                     */
#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)

#ifdef __cplusplus
extern "C" {
#endif

#if !defined _WIN32_WCE
#include <errno.h>
#endif
#include <stddef.h>
#include <stdio.h>
#if defined _WIN32
//  Set target version to Windows Server 2008, Windows Vista or higher.
//  Windows XP (0x0501) is supported but without client & server socket types.
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0600
#endif

#ifdef __MINGW32__
//  Require Windows XP or higher with MinGW for getaddrinfo().
#if (_WIN32_WINNT >= 0x0501)
#else
#error You need at least Windows XP target
#endif
#endif
#endif

/*  Handle DSO symbol visibility                                             */
#if defined _WIN32
#if defined ZMQ_STATIC
#define ZMQ_EXPORT
#elif defined DLL_EXPORT
#define ZMQ_EXPORT __declspec(dllexport)
#else
#define ZMQ_EXPORT __declspec(dllimport)
#endif
#else
#if defined __SUNPRO_C || defined __SUNPRO_CC
#define ZMQ_EXPORT __global
#elif (defined __GNUC__ && __GNUC__ >= 4) || defined __INTEL_COMPILER
#define ZMQ_EXPORT __attribute__ ((visibility ("default")))
#else
#define ZMQ_EXPORT
#endif
#endif

/*  Define integer types needed for event interface                          */
#define ZMQ_DEFINED_STDINT 1
#if defined ZMQ_HAVE_SOLARIS || defined ZMQ_HAVE_OPENVMS
#include <inttypes.h>
#elif defined _MSC_VER && _MSC_VER < 1600
#ifndef uint64_t
typedef unsigned __int64 uint64_t;
#endif
#ifndef int32_t
typedef __int32 int32_t;
#endif
#ifndef uint32_t
typedef unsigned __int32 uint32_t;
#endif
#ifndef uint16_t
typedef unsigned __int16 uint16_t;
#endif
#ifndef uint8_t
typedef unsigned __int8 uint8_t;
#endif
#else
#include <stdint.h>
#endif

//  32-bit AIX's pollfd struct members are called reqevents and rtnevents so it
//  defines compatibility macros for them. Need to include that header first to
//  stop build failures since zmq_pollset_t defines them as events and revents.
#ifdef ZMQ_HAVE_AIX
#include <poll.h>
#endif


/******************************************************************************/
/*  0MQ errors.                                                               */
/******************************************************************************/

/*  A number random enough not to collide with different errno ranges on      */
/*  different OSes. The assumption is that error_t is at least 32-bit type.   */
#define ZMQ_HAUSNUMERO 156384712

/*  On Windows platform some of the standard POSIX errnos are not defined.    */
#ifndef ENOTSUP
#define ENOTSUP (ZMQ_HAUSNUMERO + 1)
#endif
#ifndef EPROTONOSUPPORT
#define EPROTONOSUPPORT (ZMQ_HAUSNUMERO + 2)
#endif
#ifndef ENOBUFS
#define ENOBUFS (ZMQ_HAUSNUMERO + 3)
#endif
#ifndef ENETDOWN
#define ENETDOWN (ZMQ_HAUSNUMERO + 4)
#endif
#ifndef EADDRINUSE
#define EADDRINUSE (ZMQ_HAUSNUMERO + 5)
#endif
#ifndef EADDRNOTAVAIL
#define EADDRNOTAVAIL (ZMQ_HAUSNUMERO + 6)
#endif
#ifndef ECONNREFUSED
#define ECONNREFUSED (ZMQ_HAUSNUMERO + 7)
#endif
#ifndef EINPROGRESS
#define EINPROGRESS (ZMQ_HAUSNUMERO + 8)
#endif
#ifndef ENOTSOCK
#define ENOTSOCK (ZMQ_HAUSNUMERO + 9)
#endif
#ifndef EMSGSIZE
#define EMSGSIZE (ZMQ_HAUSNUMERO + 10)
#endif
#ifndef EAFNOSUPPORT
#define EAFNOSUPPORT (ZMQ_HAUSNUMERO + 11)
#endif
#ifndef ENETUNREACH
#define ENETUNREACH (ZMQ_HAUSNUMERO + 12)
#endif
#ifndef ECONNABORTED
#define ECONNABORTED (ZMQ_HAUSNUMERO + 13)
#endif
#ifndef ECONNRESET
#define ECONNRESET (ZMQ_HAUSNUMERO + 14)
#endif
#ifndef ENOTCONN
#define ENOTCONN (ZMQ_HAUSNUMERO + 15)
#endif
#ifndef ETIMEDOUT
#define ETIMEDOUT (ZMQ_HAUSNUMERO + 16)
#endif
#ifndef EHOSTUNREACH
#define EHOSTUNREACH (ZMQ_HAUSNUMERO + 17)
#endif
#ifndef ENETRESET
#define ENETRESET (ZMQ_HAUSNUMERO + 18)
#endif

/*  Native 0MQ error codes.                                                   */
#define EFSM (ZMQ_HAUSNUMERO + 51)
#define ENOCOMPATPROTO (ZMQ_HAUSNUMERO + 52)
#define ETERM (ZMQ_HAUSNUMERO + 53)
#define EMTHREAD (ZMQ_HAUSNUMERO + 54)

/*  This function retrieves the errno as it is known to 0MQ library. The goal */
/*  of this function is to make the code 100% portable, including where 0MQ   */
/*  compiled with certain CRT library (on Windows) is linked to an            */
/*  application that uses different CRT library.                              */
ZMQ_EXPORT int zmq_errno (void);

/*  Resolves system errors and 0MQ errors to human-readable string.           */
ZMQ_EXPORT const char *zmq_strerror (int errnum_);

/*  Run-time API version detection                                            */
ZMQ_EXPORT void zmq_version (int *major_, int *minor_, int *patch_);

/******************************************************************************/
/*  0MQ infrastructure (a.k.a. context) initialisation & termination.         */
/******************************************************************************/

/*  Context options                                                           */
#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

/*  Default for new contexts                                                  */
#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

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_);

/*  Old (legacy) API                                                          */
ZMQ_EXPORT void *zmq_init (int io_threads_);
ZMQ_EXPORT int zmq_term (void *context_);
ZMQ_EXPORT int zmq_ctx_destroy (void *context_);


/******************************************************************************/
/*  0MQ message definition.                                                   */
/******************************************************************************/

/* Some architectures, like sparc64 and some variants of aarch64, enforce pointer
 * alignment and raise sigbus on violations. Make sure applications allocate
 * zmq_msg_t on addresses aligned on a pointer-size boundary to avoid this issue.
 */
typedef struct zmq_msg_t
{
#if defined(_MSC_VER) && (defined(_M_X64) || defined(_M_ARM64))
    __declspec(align (8)) unsigned char _[64];
#elif defined(_MSC_VER)                                                        \
  && (defined(_M_IX86) || defined(_M_ARM_ARMV7VE) || defined(_M_ARM))
    __declspec(align (4)) unsigned char _[64];
#elif defined(__GNUC__) || defined(__INTEL_COMPILER)                           \
  || (defined(__SUNPRO_C) && __SUNPRO_C >= 0x590)                              \
  || (defined(__SUNPRO_CC) && __SUNPRO_CC >= 0x590)
    unsigned char _[64] __attribute__ ((aligned (sizeof (void *))));
#else
    unsigned char _[64];
#endif
} zmq_msg_t;

typedef void(zmq_free_fn) (void *data_, void *hint_);

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_);

/******************************************************************************/
/*  0MQ socket definition.                                                    */
/******************************************************************************/

/*  Socket types.                                                             */
#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

/*  Deprecated aliases                                                        */
#define ZMQ_XREQ ZMQ_DEALER
```

```
#define ZMQ_XREP ZMQ_ROUTER

/*  Socket options.                                                           */
#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_VERBOSER 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_NAMETYPE 90
#define ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAMETYPE 91
#define ZMQ_BINDTODEVICE 92

/*  Message options                                                           */
#define ZMQ_MORE 1
#define ZMQ_SHARED 3

/*  Send/recv options.                                                        */
#define ZMQ_DONTWAIT 1
#define ZMQ_SNDMORE 2
```

```

/*  Security mechanisms                                                      */
#define ZMQ_NULL 0
#define ZMQ_PLAIN 1
#define ZMQ_CURVE 2
#define ZMQ_GSSAPI 3

/*  RADIO-DISH protocol                                                      */
#define ZMQ_GROUP_MAX_LENGTH 255

/*  Deprecated options and aliases                                           */
#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

/*  Deprecated Message options                                               */
#define ZMQ_SRCFD 2

/******************************************************************************/
/*  GSSAPI definitions                                                       */
/******************************************************************************/

/*  GSSAPI principal name types                                              */
#define ZMQ_GSSAPI_NT_HOSTBASED 0
#define ZMQ_GSSAPI_NT_USER_NAME 1
#define ZMQ_GSSAPI_NT_KRB5_PRINCIPAL 2

/******************************************************************************/
/*  0MQ socket events and monitoring                                         */
/******************************************************************************/

/*  Socket transport events (TCP, IPC and TIPC only)                         */

#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
/*  Unspecified system errors during handshake. Event value is an errno.     */
#define ZMQ_EVENT_HANDSHAKE_FAILED_NO_DETAIL 0x0800
/*  Handshake complete successfully with successful authentication (if       *
 *  enabled). Event value is unused.                                         */
#define ZMQ_EVENT_HANDSHAKE_SUCCEEDED 0x1000
/*  Protocol errors between ZMTP peers or between server and ZAP handler.    *
 *  Event value is one of ZMQ_PROTOCOL_ERROR_*                               */
#define ZMQ_EVENT_HANDSHAKE_FAILED_PROTOCOL 0x2000
/*  Failed authentication requests. Event value is the numeric ZAP status    *
 *  code, i.e. 300, 400 or 500.                                              */
#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
// the following two may be due to erroneous configuration of a peer
#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

```

```
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_);

/******************************************************************************/
/*  Hide socket fd type; this was before zmq_poller_event_t typedef below     */
/******************************************************************************/

#if defined _WIN32
// Windows uses a pointer-sized unsigned integer to store the socket fd.
#if defined _WIN64
typedef unsigned __int64 zmq_fd_t;
#else
typedef unsigned int zmq_fd_t;
#endif
#else
typedef int zmq_fd_t;
#endif

/******************************************************************************/
/*  Deprecated I/O multiplexing. Prefer using zmq_poller API                  */
/******************************************************************************/

#define ZMQ_POLLIN 1
#define ZMQ_POLLOUT 2
#define ZMQ_POLLERR 4
#define ZMQ_POLLPRI 8

typedef struct zmq_pollitem_t
{
    void *socket;
    zmq_fd_t fd;
    short events;
    short revents;
} zmq_pollitem_t;

#define ZMQ_POLLITEMS_DFLT 16

ZMQ_EXPORT int zmq_poll (zmq_pollitem_t *items_, int nitems_, long timeout_);

/******************************************************************************/
/*  Message proxying                                                          */
/******************************************************************************/

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_);

/******************************************************************************/
/*  Probe library capabilities                                                */
/******************************************************************************/

#define ZMQ_HAS_CAPABILITIES 1
ZMQ_EXPORT int zmq_has (const char *capability_);

/*  Deprecated aliases */
#define ZMQ_STREAMER 1
#define ZMQ_FORWARDER 2
#define ZMQ_QUEUE 3

/*  Deprecated methods */
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_);
struct iovec;
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_);

/******************************************************************************/
/*  Encryption functions                                                      */
/******************************************************************************/
```

```

/*  Encode data with Z85 encoding. Returns encoded data                     */
ZMQ_EXPORT char *
zmq_z85_encode (char *dest_, const uint8_t *data_, size_t size_);

/*  Decode data with Z85 encoding. Returns decoded data                     */
ZMQ_EXPORT uint8_t *zmq_z85_decode (uint8_t *dest_, const char *string_);

/*  Generate z85-encoded public and private keypair with tweetnacl/libsodium. */
/*  Returns 0 on success.                                                    */
ZMQ_EXPORT int zmq_curve_keypair (char *z85_public_key_, char *z85_secret_key_);

/*  Derive the z85-encoded public key from the z85-encoded secret key.       */
/*  Returns 0 on success.                                                    */
ZMQ_EXPORT int zmq_curve_public (char *z85_public_key_,
                                 const char *z85_secret_key_);

/******************************************************************************/
/*  Atomic utility methods                                                   */
/******************************************************************************/

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_);

/******************************************************************************/
/*  Scheduling timers                                                        */
/******************************************************************************/

#define ZMQ_HAVE_TIMERS

typedef void(zmq_timer_fn) (int timer_id, void *arg);

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);


/******************************************************************************/
/*  These functions are not documented by man pages -- use at your own risk. */
/*  If you need these to be part of the formal ZMQ API, then (a) write a man */
/*  page, and (b) write a test case in tests.                                */
/******************************************************************************/

/*  Helper functions are used by perf tests so that they don't have to care   */
/*  about minutiae of time-related functions on different OS platforms.      */

/*  Starts the stopwatch. Returns the handle to the watch.                   */
ZMQ_EXPORT void *zmq_stopwatch_start (void);

/*  Returns the number of microseconds elapsed since the stopwatch was       */
/*  started, but does not stop or deallocate the stopwatch.                  */
ZMQ_EXPORT unsigned long zmq_stopwatch_intermediate (void *watch_);

/*  Stops the stopwatch. Returns the number of microseconds elapsed since    */
/*  the stopwatch was started, and deallocates that watch.                   */
ZMQ_EXPORT unsigned long zmq_stopwatch_stop (void *watch_);

/*  Sleeps for specified number of seconds.                                  */
ZMQ_EXPORT void zmq_sleep (int seconds_);

typedef void(zmq_thread_fn) (void *);

/* Start a thread. Returns a handle to the thread.                           */
ZMQ_EXPORT void *zmq_threadstart (zmq_thread_fn *func_, void *arg_);

/* Wait for thread to complete then free up resources.                       */
ZMQ_EXPORT void zmq_threadclose (void *thread_);


/******************************************************************************/
/*  These functions are DRAFT and disabled in stable releases, and subject to */
/*  change at ANY time until declared stable.                                */
/******************************************************************************/

#ifdef ZMQ_BUILD_DRAFT_API

```

```
/*  DRAFT Socket types.                                                       */
#define ZMQ_SERVER 12
#define ZMQ_CLIENT 13
#define ZMQ_RADIO 14
#define ZMQ_DISH 15
#define ZMQ_GATHER 16
#define ZMQ_SCATTER 17
#define ZMQ_DGRAM 18
#define ZMQ_PEER 19
#define ZMQ_CHANNEL 20

/*  DRAFT Socket options.                                                     */
#define ZMQ_ZAP_ENFORCE_DOMAIN 93
#define ZMQ_LOOPBACK_FASTPATH 94
#define ZMQ_METADATA 95
#define ZMQ_MULTICAST_LOOP 96
#define ZMQ_ROUTER_NOTIFY 97
#define ZMQ_XPUB_MANUAL_LAST_VALUE 98
#define ZMQ_SOCKS_USERNAME 99
#define ZMQ_SOCKS_PASSWORD 100
#define ZMQ_IN_BATCH_SIZE 101
#define ZMQ_OUT_BATCH_SIZE 102
#define ZMQ_WSS_KEY_PEM 103
#define ZMQ_WSS_CERT_PEM 104
#define ZMQ_WSS_TRUST_PEM 105
#define ZMQ_WSS_HOSTNAME 106
#define ZMQ_WSS_TRUST_SYSTEM 107
#define ZMQ_ONLY_FIRST_SUBSCRIBE 108
#define ZMQ_RECONNECT_STOP 109
#define ZMQ_HELLO_MSG 110
#define ZMQ_DISCONNECT_MSG 111
#define ZMQ_PRIORITY 112

/*  DRAFT ZMQ_RECONNECT_STOP options                                          */
#define ZMQ_RECONNECT_STOP_CONN_REFUSED 0x1
#define ZMQ_RECONNECT_STOP_HANDSHAKE_FAILED 0x2
#define ZMQ_RECONNECT_STOP_AFTER_DISCONNECT 0x3

/*  DRAFT Context options                                                     */
#define ZMQ_ZERO_COPY_RECV 10

/*  DRAFT Context methods.                                                    */
ZMQ_EXPORT int zmq_ctx_set_ext (void *context_,
                                int option_,
                                const void *optval_,
                                size_t optvallen_);
ZMQ_EXPORT int zmq_ctx_get_ext (void *context_,
                                int option_,
                                void *optval_,
                                size_t *optvallen_);

/*  DRAFT Socket methods.                                                     */
ZMQ_EXPORT int zmq_join (void *s, const char *group);
ZMQ_EXPORT int zmq_leave (void *s, const char *group);
ZMQ_EXPORT uint32_t zmq_connect_peer (void *s_, const char *addr_);

/*  DRAFT Msg methods.                                                        */
ZMQ_EXPORT int zmq_msg_set_routing_id (zmq_msg_t *msg, uint32_t routing_id);
ZMQ_EXPORT uint32_t zmq_msg_routing_id (zmq_msg_t *msg);
ZMQ_EXPORT int zmq_msg_set_group (zmq_msg_t *msg, const char *group);
ZMQ_EXPORT const char *zmq_msg_group (zmq_msg_t *msg);
ZMQ_EXPORT int
zmq_msg_init_buffer (zmq_msg_t *msg_, const void *buf_, size_t size_);

/*  DRAFT Msg property names.                                                 */
#define ZMQ_MSG_PROPERTY_ROUTING_ID "Routing-Id"
#define ZMQ_MSG_PROPERTY_SOCKET_TYPE "Socket-Type"
#define ZMQ_MSG_PROPERTY_USER_ID "User-Id"
#define ZMQ_MSG_PROPERTY_PEER_ADDRESS "Peer-Address"

/*  Router notify options                                                     */
#define ZMQ_NOTIFY_CONNECT 1
#define ZMQ_NOTIFY_DISCONNECT 2

/******************************************************************************/
/*  Poller polling on sockets,fd and thread-safe sockets                      */
/******************************************************************************/

#define ZMQ_HAVE_POLLER

typedef struct zmq_poller_event_t
{
    void *socket;
    zmq_fd_t fd;
    void *user_data;
    short events;
} zmq_poller_event_t;
```

```

ZMQ_EXPORT void *zmq_poller_new (void);
ZMQ_EXPORT int zmq_poller_destroy (void **poller_p);
ZMQ_EXPORT int zmq_poller_size (void *poller);
ZMQ_EXPORT int
zmq_poller_add (void *poller, void *socket, void *user_data, short events);
ZMQ_EXPORT int zmq_poller_modify (void *poller, void *socket, short events);
ZMQ_EXPORT int zmq_poller_remove (void *poller, void *socket);
ZMQ_EXPORT int
zmq_poller_wait (void *poller, zmq_poller_event_t *event, long timeout);
ZMQ_EXPORT int zmq_poller_wait_all (void *poller,
                                    zmq_poller_event_t *events,
                                    int n_events,
                                    long timeout);
ZMQ_EXPORT int zmq_poller_fd (void *poller, zmq_fd_t *fd);

ZMQ_EXPORT int
zmq_poller_add_fd (void *poller, zmq_fd_t fd, void *user_data, short events);
ZMQ_EXPORT int zmq_poller_modify_fd (void *poller, zmq_fd_t fd, short events);
ZMQ_EXPORT int zmq_poller_remove_fd (void *poller, zmq_fd_t fd);

ZMQ_EXPORT int zmq_socket_get_peer_state (void *socket,
                                          const void *routing_id,
                                          size_t routing_id_size);

/*  DRAFT Socket monitoring events                                            */
#define ZMQ_EVENT_PIPES_STATS 0x10000

#define ZMQ_CURRENT_EVENT_VERSION 1
#define ZMQ_CURRENT_EVENT_VERSION_DRAFT 2

#define ZMQ_EVENT_ALL_V1 ZMQ_EVENT_ALL
#define ZMQ_EVENT_ALL_V2 ZMQ_EVENT_ALL_V1 | ZMQ_EVENT_PIPES_STATS

ZMQ_EXPORT int zmq_socket_monitor_versioned (
  void *s_, const char *addr_, uint64_t events_, int event_version_, int type_);
ZMQ_EXPORT int zmq_socket_monitor_pipes_stats (void *s);

#endif // ZMQ_BUILD_DRAFT_API


#undef ZMQ_EXPORT

#ifdef __cplusplus
}
#endif

#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.

```
/*
    Copyright (c) 2016-2017 ZeroMQ community
    Copyright (c) 2009-2011 250bpm s.r.o.
    Copyright (c) 2011 Botond Ballo
    Copyright (c) 2007-2009 iMatix Corporation

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to
    deal in the Software without restriction, including without limitation the
    rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
    sell copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
    FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
    IN THE SOFTWARE.
*/

#ifndef __ZMQ_HPP_INCLUDED__
#define __ZMQ_HPP_INCLUDED__

#ifdef _WIN32
#ifndef NOMINMAX
#define NOMINMAX
#endif
#endif

// included here for _HAS_CXX* macros
#include "zmq.h"

#if defined(_MSVC_LANG)
#define CPPZMQ_LANG _MSVC_LANG
#else
#define CPPZMQ_LANG __cplusplus
#endif
// overwrite if specific language macros indicate higher version
#if defined(_HAS_CXX14) && _HAS_CXX14 && CPPZMQ_LANG < 201402L
#undef CPPZMQ_LANG
#define CPPZMQ_LANG 201402L
#endif
#if defined(_HAS_CXX17) && _HAS_CXX17 && CPPZMQ_LANG < 201703L
#undef CPPZMQ_LANG
#define CPPZMQ_LANG 201703L
#endif

// macros defined if has a specific standard or greater
#if CPPZMQ_LANG >= 201103L || (defined(_MSC_VER) && _MSC_VER >= 1900)
#define ZMQ_CPP11
#endif
#if CPPZMQ_LANG >= 201402L
#define ZMQ_CPP14
#endif
#if CPPZMQ_LANG >= 201703L
#define ZMQ_CPP17
#endif

#if defined(ZMQ_CPP14) && !defined(_MSC_VER)
#define ZMQ_DEPRECATED(msg) [[deprecated(msg)]]
#elif defined(_MSC_VER)
#define ZMQ_DEPRECATED(msg) __declspec(deprecated(msg))
#elif defined(__GNUC__)
#define ZMQ_DEPRECATED(msg) __attribute__((deprecated(msg)))
#else
#define ZMQ_DEPRECATED(msg)
#endif

#if defined(ZMQ_CPP17)
#define ZMQ_NODISCARD [[nodiscard]]
```

```
#else
#define ZMQ_NODISCARD
#endif

#if defined(ZMQ_CPP11)
#define ZMQ_NOTHROW noexcept
#define ZMQ_EXPLICIT explicit
#define ZMQ_OVERRIDE override
#define ZMQ_NULLPTR nullptr
#define ZMQ_CONSTEXPR_FN constexpr
#define ZMQ_CONSTEXPR_VAR constexpr
#define ZMQ_CPP11_DEPRECATED(msg) ZMQ_DEPRECATED(msg)
#else
#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)
#endif
#if defined(ZMQ_CPP14) && (!defined(_MSC_VER) || _MSC_VER > 1900) && (!defined(__GNUC__) || __GNUC__ > 5 || (__GNUC__ == 5 && __GNUC_MINOR__ > 3))
#define ZMQ_EXTENDED_CONSTEXPR
#endif
#if defined(ZMQ_CPP17)
#define ZMQ_INLINE_VAR inline
#define ZMQ_CONSTEXPR_IF constexpr
#else
#define ZMQ_INLINE_VAR
#define ZMQ_CONSTEXPR_IF
#endif

#include <cassert>
#include <cstring>

#include <algorithm>
#include <exception>
#include <iomanip>
#include <sstream>
#include <string>
#include <vector>
#ifdef ZMQ_CPP11
#include <array>
#include <chrono>
#include <tuple>
#include <memory>
#endif

#if defined(__has_include) && defined(ZMQ_CPP17)
#define CPPZMQ_HAS_INCLUDE_CPP17(X) __has_include(X)
#else
#define CPPZMQ_HAS_INCLUDE_CPP17(X) 0
#endif

#if CPPZMQ_HAS_INCLUDE_CPP17(<optional>) && !defined(CPPZMQ_HAS_OPTIONAL)
#define CPPZMQ_HAS_OPTIONAL 1
#endif
#ifndef CPPZMQ_HAS_OPTIONAL
#define CPPZMQ_HAS_OPTIONAL 0
#elif CPPZMQ_HAS_OPTIONAL
#include <optional>
#endif

#if CPPZMQ_HAS_INCLUDE_CPP17(<string_view>) && !defined(CPPZMQ_HAS_STRING_VIEW)
#define CPPZMQ_HAS_STRING_VIEW 1
#endif
#ifndef CPPZMQ_HAS_STRING_VIEW
#define CPPZMQ_HAS_STRING_VIEW 0
#elif CPPZMQ_HAS_STRING_VIEW
#include <string_view>
#endif

/*  Version macros for compile-time API version detection                      */
#define CPPZMQ_VERSION_MAJOR 4
#define CPPZMQ_VERSION_MINOR 10
#define CPPZMQ_VERSION_PATCH 0

#define CPPZMQ_VERSION                                                               \
    ZMQ_MAKE_VERSION(CPPZMQ_VERSION_MAJOR, CPPZMQ_VERSION_MINOR,                     \
                     CPPZMQ_VERSION_PATCH)

//  Detect whether the compiler supports C++11 rvalue references.
#if (defined(__GNUC__) && (__GNUC__ > 4 || (__GNUC__ == 4 && __GNUC_MINOR__ > 2))    \
     && defined(__GXX_EXPERIMENTAL_CXX0X__))
#define ZMQ_HAS_RVALUE_REFS
#define ZMQ_DELETED_FUNCTION = delete
```

```
#elif defined(__clang__)
#if __has_feature(cxx_rvalue_references)
#define ZMQ_HAS_RVALUE_REFS
#endif

#if __has_feature(cxx_deleted_functions)
#define ZMQ_DELETED_FUNCTION = delete
#else
#define ZMQ_DELETED_FUNCTION
#endif
#elif defined(_MSC_VER) && (_MSC_VER >= 1900)
#define ZMQ_HAS_RVALUE_REFS
#define ZMQ_DELETED_FUNCTION = delete
#elif defined(_MSC_VER) && (_MSC_VER >= 1600)
#define ZMQ_HAS_RVALUE_REFS
#define ZMQ_DELETED_FUNCTION
#else
#define ZMQ_DELETED_FUNCTION
#endif

#if defined(ZMQ_CPP11) && !defined(__llvm__) && !defined(__INTEL_COMPILER)          \
  && defined(__GNUC__) && __GNUC__ < 5
#define ZMQ_CPP11_PARTIAL
#elif defined(__GLIBCXX__) && __GLIBCXX__ < 20160805
//the date here is the last date of gcc 4.9.4, which
// effectively means libstdc++ from gcc 5.5 and higher won't trigger this branch
#define ZMQ_CPP11_PARTIAL
#endif

#ifdef ZMQ_CPP11
#ifdef ZMQ_CPP11_PARTIAL
#define ZMQ_IS_TRIVIALLY_COPYABLE(T) __has_trivial_copy(T)
#else
#include <type_traits>
#define ZMQ_IS_TRIVIALLY_COPYABLE(T) std::is_trivially_copyable<T>::value
#endif
#endif

#if ZMQ_VERSION >= ZMQ_MAKE_VERSION(3, 3, 0)
#define ZMQ_NEW_MONITOR_EVENT_LAYOUT
#endif

#if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 1, 0)
#define ZMQ_HAS_PROXY_STEERABLE
/*  Socket event data  */
typedef struct
{
    uint16_t event; // id of the event as bitfield
    int32_t value;  // value is either error code, fd or reconnect interval
} zmq_event_t;
#endif

// Avoid using deprecated message receive function when possible
#if ZMQ_VERSION < ZMQ_MAKE_VERSION(3, 2, 0)
#define zmq_msg_recv(msg, socket, flags) zmq_recvmsg(socket, msg, flags)
#endif


// In order to prevent unused variable warnings when building in non-debug
// mode use this macro to make assertions.
#ifndef NDEBUG
#define ZMQ_ASSERT(expression) assert(expression)
#else
#define ZMQ_ASSERT(expression) (void) (expression)
#endif

namespace zmq
{
#ifdef ZMQ_CPP11
namespace detail
{
namespace ranges
{
using std::begin;
using std::end;
template<class T> auto begin(T &&r) -> decltype(begin(std::forward<T>(r)))
{
    return begin(std::forward<T>(r));
}
template<class T> auto end(T &&r) -> decltype(end(std::forward<T>(r)))
{
    return end(std::forward<T>(r));
}
} // namespace ranges

template<class T> using void_t = void;

```

```
template<class Iter>
using iter_value_t = typename std::iterator_traits<Iter>::value_type;

template<class Range>
using range_iter_t = decltype(
  ranges::begin(std::declval<typename std::remove_reference<Range>::type &>()));

template<class Range> using range_value_t = iter_value_t<range_iter_t<Range»;

template<class T, class = void> struct is_range : std::false_type
{
};

template<class T>
struct is_range<
  T,
  void_t<decltype(
    ranges::begin(std::declval<typename std::remove_reference<T>::type &>())
    == ranges::end(std::declval<typename std::remove_reference<T>::type &>()))»
    : std::true_type
{
};

} // namespace detail
#endif

typedef zmq_free_fn free_fn;
typedef zmq_pollitem_t pollitem_t;

// duplicate definition from libzmq 4.3.3
#if defined _WIN32
#if defined _WIN64
typedef unsigned __int64 fd_t;
#else
typedef unsigned int fd_t;
#endif
#else
typedef int fd_t;
#endif

class error_t : public std::exception
{
  public:
    error_t() ZMQ_NOTHROW : errnum(zmq_errno()) {}
    explicit error_t(int err) ZMQ_NOTHROW : errnum(err) {}
    virtual const char *what() const ZMQ_NOTHROW ZMQ_OVERRIDE
    {
        return zmq_strerror(errnum);
    }
    int num() const ZMQ_NOTHROW { return errnum; }

  private:
    int errnum;
};

namespace detail {
inline int poll(zmq_pollitem_t *items_, size_t nitems_, long timeout_)
{
    int rc = zmq_poll(items_, static_cast<int>(nitems_), timeout_);
    if (rc < 0)
        throw error_t();
    return rc;
}
}

#ifdef ZMQ_CPP11
ZMQ_DEPRECATED("from 4.8.0, use poll taking std::chrono::duration instead of long")
inline int poll(zmq_pollitem_t *items_, size_t nitems_, long timeout_)
#else
inline int poll(zmq_pollitem_t *items_, size_t nitems_, long timeout_ = -1)
#endif
{
    return detail::poll(items_, nitems_, timeout_);
}

ZMQ_DEPRECATED("from 4.3.1, use poll taking non-const items")
inline int poll(zmq_pollitem_t const *items_, size_t nitems_, long timeout_ = -1)
{
    return detail::poll(const_cast<zmq_pollitem_t *>(items_), nitems_, timeout_);
}

#ifdef ZMQ_CPP11
ZMQ_DEPRECATED("from 4.3.1, use poll taking non-const items")
inline int
poll(zmq_pollitem_t const *items, size_t nitems, std::chrono::milliseconds timeout)
{
    return detail::poll(const_cast<zmq_pollitem_t *>(items), nitems,
                static_cast<long>(timeout.count()));
}

ZMQ_DEPRECATED("from 4.3.1, use poll taking non-const items")
inline int poll(std::vector<zmq_pollitem_t> const &items,
                std::chrono::milliseconds timeout)
{
    return detail::poll(const_cast<zmq_pollitem_t *>(items.data()), items.size(),
                static_cast<long>(timeout.count()));
}

ZMQ_DEPRECATED("from 4.3.1, use poll taking non-const items")
inline int poll(std::vector<zmq_pollitem_t> const &items, long timeout_ = -1)
{
    return detail::poll(const_cast<zmq_pollitem_t *>(items.data()), items.size(), timeout_);
}

inline int
poll(zmq_pollitem_t *items, size_t nitems, std::chrono::milliseconds timeout =
std::chrono::milliseconds{-1})
{
    return detail::poll(items, nitems, static_cast<long>(timeout.count()));
}

inline int poll(std::vector<zmq_pollitem_t> &items,
                std::chrono::milliseconds timeout = std::chrono::milliseconds{-1})
{
    return detail::poll(items.data(), items.size(), static_cast<long>(timeout.count()));
}

ZMQ_DEPRECATED("from 4.3.1, use poll taking std::chrono::duration instead of long")
inline int poll(std::vector<zmq_pollitem_t> &items, long timeout_)
{
    return detail::poll(items.data(), items.size(), timeout_);
}

template<std::size_t SIZE>
inline int poll(std::array<zmq_pollitem_t, SIZE> &items,
                std::chrono::milliseconds timeout = std::chrono::milliseconds{-1})
{
    return detail::poll(items.data(), items.size(), static_cast<long>(timeout.count()));
}
#endif


inline void version(int *major_, int *minor_, int *patch_)
{
    zmq_version(major_, minor_, patch_);
}

#ifdef ZMQ_CPP11
inline std::tuple<int, int, int> version()
{
    std::tuple<int, int, int> v;
    zmq_version(&std::get<0>(v), &std::get<1>(v), &std::get<2>(v));
    return v;
}

#if !defined(ZMQ_CPP11_PARTIAL)
namespace detail
{
template<class T> struct is_char_type
{
    // true if character type for string literals in C++11
    static constexpr bool value =
      std::is_same<T, char>::value || std::is_same<T, wchar_t>::value
      || std::is_same<T, char16_t>::value || std::is_same<T, char32_t>::value;
};
}
#endif

#endif

class message_t
{
  public:
    message_t() ZMQ_NOTHROW
    {
        int rc = zmq_msg_init(&msg);
        ZMQ_ASSERT(rc == 0);
    }

    explicit message_t(size_t size_)
    {
        int rc = zmq_msg_init_size(&msg, size_);
        if (rc != 0)
            throw error_t();
    }

    template<class ForwardIter> message_t(ForwardIter first, ForwardIter last)
    {
        typedef typename std::iterator_traits<ForwardIter>::value_type value_t;

        assert(std::distance(first, last) >= 0);
        size_t const size_ =
          static_cast<size_t>(std::distance(first, last)) * sizeof(value_t);
        int const rc = zmq_msg_init_size(&msg, size_);
        if (rc != 0)
            throw error_t();
        std::copy(first, last, data<value_t>());
    }

    message_t(const void *data_, size_t size_)
    {
        int rc = zmq_msg_init_size(&msg, size_);
        if (rc != 0)
            throw error_t();
        if (size_) {
            // this constructor allows (nullptr, 0),
            // memcpy with a null pointer is UB
            memcpy(data(), data_, size_);
        }
    }

    message_t(void *data_, size_t size_, free_fn *ffn_, void *hint_ = ZMQ_NULLPTR)
    {
        int rc = zmq_msg_init_data(&msg, data_, size_, ffn_, hint_);
        if (rc != 0)
            throw error_t();
    }

    // overload set of string-like types and generic containers
#if defined(ZMQ_CPP11) && !defined(ZMQ_CPP11_PARTIAL)
    // NOTE this constructor will include the null terminator
    // when called with a string literal.
    // An overload taking const char* can not be added because
    // it would be preferred over this function and break compatiblity.
    template<
      class Char,
      size_t N,
      typename = typename std::enable_if<detail::is_char_type<Char>::value>::type>
    ZMQ_DEPRECATED("from 4.7.0, use constructors taking iterators, (pointer, size) "
                   "or strings instead")
    explicit message_t(const Char (&data)[N]) :
        message_t(detail::ranges::begin(data), detail::ranges::end(data))
    {
    }

    template<class Range,
             typename = typename std::enable_if<
               detail::is_range<Range>::value
               && ZMQ_IS_TRIVIALLY_COPYABLE(detail::range_value_t<Range>)
               && !detail::is_char_type<detail::range_value_t<Range>>::value
               && !std::is_same<Range, message_t>::value>::type>
    explicit message_t(const Range &rng) :
        message_t(detail::ranges::begin(rng), detail::ranges::end(rng))
    {
    }

    explicit message_t(const std::string &str) : message_t(str.data(), str.size()) {}

#if CPPZMQ_HAS_STRING_VIEW
    explicit message_t(std::string_view str) : message_t(str.data(), str.size()) {}
#endif

#endif

#ifdef ZMQ_HAS_RVALUE_REFS
    message_t(message_t &&rhs) ZMQ_NOTHROW : msg(rhs.msg)
    {
        int rc = zmq_msg_init(&rhs.msg);
        ZMQ_ASSERT(rc == 0);
    }

    message_t &operator=(message_t &&rhs) ZMQ_NOTHROW
    {
        std::swap(msg, rhs.msg);
        return *this;
    }
#endif

    ~message_t() ZMQ_NOTHROW
    {
        int rc = zmq_msg_close(&msg);
        ZMQ_ASSERT(rc == 0);
    }

    void rebuild()
    {
        int rc = zmq_msg_close(&msg);
        if (rc != 0)
            throw error_t();
        rc = zmq_msg_init(&msg);
        ZMQ_ASSERT(rc == 0);
    }

    void rebuild(size_t size_)
    {
        int rc = zmq_msg_close(&msg);
        if (rc != 0)
            throw error_t();
        rc = zmq_msg_init_size(&msg, size_);
        if (rc != 0)
            throw error_t();
    }

    void rebuild(const void *data_, size_t size_)
    {
        int rc = zmq_msg_close(&msg);
        if (rc != 0)
            throw error_t();
        rc = zmq_msg_init_size(&msg, size_);
        if (rc != 0)
            throw error_t();
        memcpy(data(), data_, size_);
    }

    void rebuild(const std::string &str)
    {
        rebuild(str.data(), str.size());
    }

    void rebuild(void *data_, size_t size_, free_fn *ffn_, void *hint_ = ZMQ_NULLPTR)
    {
        int rc = zmq_msg_close(&msg);
        if (rc != 0)
            throw error_t();
        rc = zmq_msg_init_data(&msg, data_, size_, ffn_, hint_);
        if (rc != 0)
            throw error_t();
    }

    ZMQ_DEPRECATED("from 4.3.1, use move taking non-const reference instead")
    void move(message_t const *msg_)
    {
        int rc = zmq_msg_move(&msg, const_cast<zmq_msg_t *>(msg_->handle()));
        if (rc != 0)
            throw error_t();
    }

    void move(message_t &msg_)
    {
        int rc = zmq_msg_move(&msg, msg_.handle());
        if (rc != 0)
            throw error_t();
    }

    ZMQ_DEPRECATED("from 4.3.1, use copy taking non-const reference instead")
    void copy(message_t const *msg_)
    {
        int rc = zmq_msg_copy(&msg, const_cast<zmq_msg_t *>(msg_->handle()));
        if (rc != 0)
            throw error_t();
    }

    void copy(message_t &msg_)
    {
        int rc = zmq_msg_copy(&msg, msg_.handle());
        if (rc != 0)
            throw error_t();
    }

    bool more() const ZMQ_NOTHROW
    {
        int rc = zmq_msg_more(const_cast<zmq_msg_t *>(&msg));
        return rc != 0;
    }

    void *data() ZMQ_NOTHROW { return zmq_msg_data(&msg); }

    const void *data() const ZMQ_NOTHROW
    {
        return zmq_msg_data(const_cast<zmq_msg_t *>(&msg));
    }

    size_t size() const ZMQ_NOTHROW
    {
        return zmq_msg_size(const_cast<zmq_msg_t *>(&msg));
    }

    ZMQ_NODISCARD bool empty() const ZMQ_NOTHROW { return size() == 0u; }

    template<typename T> T *data() ZMQ_NOTHROW { return static_cast<T *>(data()); }

    template<typename T> T const *data() const ZMQ_NOTHROW
    {
        return static_cast<T const *>(data());
    }

    ZMQ_DEPRECATED("from 4.3.0, use operator== instead")
    bool equal(const message_t *other) const ZMQ_NOTHROW { return *this == *other; }

    bool operator==(const message_t &other) const ZMQ_NOTHROW
    {
        const size_t my_size = size();
        return my_size == other.size() && 0 == memcmp(data(), other.data(), my_size);
    }

    bool operator!=(const message_t &other) const ZMQ_NOTHROW
    {
        return !(*this == other);
    }

#if ZMQ_VERSION >= ZMQ_MAKE_VERSION(3, 2, 0)
    int get(int property_)
    {
        int value = zmq_msg_get(&msg, property_);
        if (value == -1)
            throw error_t();
        return value;
    }
#endif

#if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 1, 0)
    const char *gets(const char *property_)
    {
        const char *value = zmq_msg_gets(&msg, property_);
        if (value == ZMQ_NULLPTR)
            throw error_t();
        return value;
    }
#endif

#if defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 0)
    uint32_t routing_id() const
    {
        return zmq_msg_routing_id(const_cast<zmq_msg_t *>(&msg));
    }

    void set_routing_id(uint32_t routing_id)
    {
        int rc = zmq_msg_set_routing_id(&msg, routing_id);
        if (rc != 0)
            throw error_t();
    }

    const char *group() const
    {
        return zmq_msg_group(const_cast<zmq_msg_t *>(&msg));
    }

    void set_group(const char *group)
    {
        int rc = zmq_msg_set_group(&msg, group);
        if (rc != 0)
            throw error_t();
    }
#endif

    // interpret message content as a string
    std::string to_string() const
    {
        return std::string(static_cast<const char *>(data()), size());
    }
#if CPPZMQ_HAS_STRING_VIEW
    // interpret message content as a string
    std::string_view to_string_view() const noexcept
    {
        return std::string_view(static_cast<const char *>(data()), size());
    }
#endif

    std::string str() const
    {
        // Partly mutuated from the same method in zmq::multipart_t
        std::stringstream os;

        const unsigned char *msg_data = this->data<unsigned char>();
        unsigned char byte;
        size_t size = this->size();
        int is_ascii[2] = {0, 0};

        os « "zmq::message_t [size " « std::dec « std::setw(3)
           « std::setfill('0') « size « "] (";
        // Totally arbitrary
        if (size >= 1000) {
            os « "... too big to print)";
        } else {
            while (size--) {
                byte = *msg_data++;

                is_ascii[1] = (byte >= 32 && byte < 127);
                if (is_ascii[1] != is_ascii[0])
                    os « " "; // Separate text/non text

                if (is_ascii[1]) {
                    os « byte;
                } else {
                    os « std::hex « std::uppercase « std::setw(2)
                       « std::setfill('0') « static_cast<short>(byte);
                }
                is_ascii[0] = is_ascii[1];
            }
            os « ")";
        }
        return os.str();
    }

    void swap(message_t &other) ZMQ_NOTHROW
    {
        // this assumes zmq::msg_t from libzmq is trivially relocatable
        std::swap(msg, other.msg);
    }

    ZMQ_NODISCARD zmq_msg_t *handle() ZMQ_NOTHROW { return &msg; }
    ZMQ_NODISCARD const zmq_msg_t *handle() const ZMQ_NOTHROW { return &msg; }

  private:
    //  The underlying message
    zmq_msg_t msg;

    //  Disable implicit message copying, so that users won't use shared
    //  messages (less efficient) without being aware of the fact.
    message_t(const message_t &) ZMQ_DELETED_FUNCTION;
    void operator=(const message_t &) ZMQ_DELETED_FUNCTION;
};

inline void swap(message_t &a, message_t &b) ZMQ_NOTHROW
{
    a.swap(b);
}

#ifdef ZMQ_CPP11
enum class ctxopt
{
#ifdef ZMQ_BLOCKY
    blocky = ZMQ_BLOCKY,
#endif
#ifdef ZMQ_IO_THREADS
    io_threads = ZMQ_IO_THREADS,
#endif
#ifdef ZMQ_THREAD_SCHED_POLICY
    thread_sched_policy = ZMQ_THREAD_SCHED_POLICY,
#endif
#ifdef ZMQ_THREAD_PRIORITY
    thread_priority = ZMQ_THREAD_PRIORITY,
#endif
#ifdef ZMQ_THREAD_AFFINITY_CPU_ADD
    thread_affinity_cpu_add = ZMQ_THREAD_AFFINITY_CPU_ADD,
#endif
#ifdef ZMQ_THREAD_AFFINITY_CPU_REMOVE
    thread_affinity_cpu_remove = ZMQ_THREAD_AFFINITY_CPU_REMOVE,
#endif
#ifdef ZMQ_THREAD_NAME_PREFIX
    thread_name_prefix = ZMQ_THREAD_NAME_PREFIX,
#endif
#ifdef ZMQ_MAX_MSGSZ
    max_msgsz = ZMQ_MAX_MSGSZ,
#endif
#ifdef ZMQ_ZERO_COPY_RECV
    zero_copy_recv = ZMQ_ZERO_COPY_RECV,
#endif
#ifdef ZMQ_MAX_SOCKETS
    max_sockets = ZMQ_MAX_SOCKETS,
#endif
#ifdef ZMQ_SOCKET_LIMIT
    socket_limit = ZMQ_SOCKET_LIMIT,
#endif
#ifdef ZMQ_IPV6
    ipv6 = ZMQ_IPV6,
#endif
#ifdef ZMQ_MSG_T_SIZE
    msg_t_size = ZMQ_MSG_T_SIZE
#endif
};
#endif

class context_t
{
  public:
    context_t()
    {
        ptr = zmq_ctx_new();
        if (ptr == ZMQ_NULLPTR)
            throw error_t();
    }


    explicit context_t(int io_threads_, int max_sockets_ = ZMQ_MAX_SOCKETS_DFLT)
    {
        ptr = zmq_ctx_new();
        if (ptr == ZMQ_NULLPTR)
            throw error_t();

        int rc = zmq_ctx_set(ptr, ZMQ_IO_THREADS, io_threads_);
        ZMQ_ASSERT(rc == 0);

        rc = zmq_ctx_set(ptr, ZMQ_MAX_SOCKETS, max_sockets_);
        ZMQ_ASSERT(rc == 0);
    }

#ifdef ZMQ_HAS_RVALUE_REFS
    context_t(context_t &&rhs) ZMQ_NOTHROW : ptr(rhs.ptr) { rhs.ptr = ZMQ_NULLPTR; }
    context_t &operator=(context_t &&rhs) ZMQ_NOTHROW
    {
        close();
        std::swap(ptr, rhs.ptr);
        return *this;
    }
#endif

    ~context_t() ZMQ_NOTHROW { close(); }

    ZMQ_CPP11_DEPRECATED("from 4.7.0, use set taking zmq::ctxopt instead")
    int setctxopt(int option_, int optval_)
    {
        int rc = zmq_ctx_set(ptr, option_, optval_);
        ZMQ_ASSERT(rc == 0);
        return rc;
    }

    ZMQ_CPP11_DEPRECATED("from 4.7.0, use get taking zmq::ctxopt instead")
    int getctxopt(int option_) { return zmq_ctx_get(ptr, option_); }

#ifdef ZMQ_CPP11
    void set(ctxopt option, int optval)
    {
        int rc = zmq_ctx_set(ptr, static_cast<int>(option), optval);
        if (rc == -1)
            throw error_t();
    }

    ZMQ_NODISCARD int get(ctxopt option)
    {
        int rc = zmq_ctx_get(ptr, static_cast<int>(option));
        // some options have a default value of -1
        // which is unfortunate, and may result in errors
        // that don't make sense
        if (rc == -1)
            throw error_t();
        return rc;
    }
```

```
#endif

    // Terminates context (see also shutdown()).
    void close() ZMQ_NOTHROW
    {
        if (ptr == ZMQ_NULLPTR)
            return;

        int rc;
        do {
            rc = zmq_ctx_term(ptr);
        } while (rc == -1 && errno == EINTR);

        ZMQ_ASSERT(rc == 0);
        ptr = ZMQ_NULLPTR;
    }

    // Shutdown context in preparation for termination (close()).
    // Causes all blocking socket operations and any further
    // socket operations to return with ETERM.
    void shutdown() ZMQ_NOTHROW
    {
        if (ptr == ZMQ_NULLPTR)
            return;
        int rc = zmq_ctx_shutdown(ptr);
        ZMQ_ASSERT(rc == 0);
    }

    //  Be careful with this, it's probably only useful for
    //  using the C api together with an existing C++ api.
    //  Normally you should never need to use this.
    ZMQ_EXPLICIT operator void *() ZMQ_NOTHROW { return ptr; }

    ZMQ_EXPLICIT operator void const *() const ZMQ_NOTHROW { return ptr; }

    ZMQ_NODISCARD void *handle() ZMQ_NOTHROW { return ptr; }

    ZMQ_DEPRECATED("from 4.7.0, use handle() != nullptr instead")
    operator bool() const ZMQ_NOTHROW { return ptr != ZMQ_NULLPTR; }

    void swap(context_t &other) ZMQ_NOTHROW { std::swap(ptr, other.ptr); }

  private:
    void *ptr;

    context_t(const context_t &) ZMQ_DELETED_FUNCTION;
    void operator=(const context_t &) ZMQ_DELETED_FUNCTION;
};

inline void swap(context_t &a, context_t &b) ZMQ_NOTHROW
{
    a.swap(b);
}

#ifdef ZMQ_CPP11

struct recv_buffer_size
{
    size_t size;             // number of bytes written to buffer
    size_t untruncated_size; // untruncated message size in bytes

    ZMQ_NODISCARD bool truncated() const noexcept
    {
        return size != untruncated_size;
    }
};

#if CPPZMQ_HAS_OPTIONAL

using send_result_t = std::optional<size_t>;
using recv_result_t = std::optional<size_t>;
using recv_buffer_result_t = std::optional<recv_buffer_size>;

#else

namespace detail
{
// A C++11 type emulating the most basic
// operations of std::optional for trivial types
template<class T> class trivial_optional
{
  public:
    static_assert(std::is_trivial<T>::value, "T must be trivial");
    using value_type = T;

    trivial_optional() = default;
    trivial_optional(T value) noexcept : _value(value), _has_value(true) {}

    const T *operator->() const noexcept
    {
        assert(_has_value);
        return &_value;
    }
    T *operator->() noexcept
    {
        assert(_has_value);
        return &_value;
    }

    const T &operator*() const noexcept
    {
        assert(_has_value);
        return _value;
    }
    T &operator*() noexcept
    {
        assert(_has_value);
        return _value;
    }

    T &value()
    {
        if (!_has_value)
            throw std::exception();
        return _value;
    }
    const T &value() const
    {
        if (!_has_value)
            throw std::exception();
        return _value;
    }

    explicit operator bool() const noexcept { return _has_value; }
    bool has_value() const noexcept { return _has_value; }

  private:
    T _value{};
    bool _has_value{false};
};
} // namespace detail

using send_result_t = detail::trivial_optional<size_t>;
using recv_result_t = detail::trivial_optional<size_t>;
using recv_buffer_result_t = detail::trivial_optional<recv_buffer_size>;

#endif

namespace detail
{
template<class T> constexpr T enum_bit_or(T a, T b) noexcept
{
    static_assert(std::is_enum<T>::value, "must be enum");
    using U = typename std::underlying_type<T>::type;
    return static_cast<T>(static_cast<U>(a) | static_cast<U>(b));
}
template<class T> constexpr T enum_bit_and(T a, T b) noexcept
{
    static_assert(std::is_enum<T>::value, "must be enum");
    using U = typename std::underlying_type<T>::type;
    return static_cast<T>(static_cast<U>(a) & static_cast<U>(b));
}
template<class T> constexpr T enum_bit_xor(T a, T b) noexcept
{
    static_assert(std::is_enum<T>::value, "must be enum");
    using U = typename std::underlying_type<T>::type;
    return static_cast<T>(static_cast<U>(a) ^ static_cast<U>(b));
}
template<class T> constexpr T enum_bit_not(T a) noexcept
{
    static_assert(std::is_enum<T>::value, "must be enum");
    using U = typename std::underlying_type<T>::type;
    return static_cast<T>(~static_cast<U>(a));
}
} // namespace detail

// partially satisfies named requirement BitmaskType
enum class send_flags : int
{
    none = 0,
    dontwait = ZMQ_DONTWAIT,
    sndmore = ZMQ_SNDMORE
};

```

```
constexpr send_flags operator|(send_flags a, send_flags b) noexcept
{
    return detail::enum_bit_or(a, b);
}
constexpr send_flags operator&(send_flags a, send_flags b) noexcept
{
    return detail::enum_bit_and(a, b);
}
constexpr send_flags operator^(send_flags a, send_flags b) noexcept
{
    return detail::enum_bit_xor(a, b);
}
constexpr send_flags operator~(send_flags a) noexcept
{
    return detail::enum_bit_not(a);
}

// partially satisfies named requirement BitmaskType
enum class recv_flags : int
{
    none = 0,
    dontwait = ZMQ_DONTWAIT
};

constexpr recv_flags operator|(recv_flags a, recv_flags b) noexcept
{
    return detail::enum_bit_or(a, b);
}
constexpr recv_flags operator&(recv_flags a, recv_flags b) noexcept
{
    return detail::enum_bit_and(a, b);
}
constexpr recv_flags operator^(recv_flags a, recv_flags b) noexcept
{
    return detail::enum_bit_xor(a, b);
}
constexpr recv_flags operator~(recv_flags a) noexcept
{
    return detail::enum_bit_not(a);
}


// mutable_buffer, const_buffer and buffer are based on
// the Networking TS specification, draft:
// http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4771.pdf

class mutable_buffer
{
  public:
    constexpr mutable_buffer() noexcept : _data(nullptr), _size(0) {}
    constexpr mutable_buffer(void *p, size_t n) noexcept : _data(p), _size(n)
    {
#ifdef ZMQ_EXTENDED_CONSTEXPR
        assert(p != nullptr || n == 0);
#endif
    }

    constexpr void *data() const noexcept { return _data; }
    constexpr size_t size() const noexcept { return _size; }
    mutable_buffer &operator+=(size_t n) noexcept
    {
        // (std::min) is a workaround for when a min macro is defined
        const auto shift = (std::min)(n, _size);
        _data = static_cast<char *>(_data) + shift;
        _size -= shift;
        return *this;
    }

  private:
    void *_data;
    size_t _size;
};

inline mutable_buffer operator+(const mutable_buffer &mb, size_t n) noexcept
{
    return mutable_buffer(static_cast<char *>(mb.data()) + (std::min)(n, mb.size()),
                          mb.size() - (std::min)(n, mb.size()));
}
inline mutable_buffer operator+(size_t n, const mutable_buffer &mb) noexcept
{
    return mb + n;
}

class const_buffer
{
  public:
    constexpr const_buffer() noexcept : _data(nullptr), _size(0) {}
    constexpr const_buffer(const void *p, size_t n) noexcept : _data(p), _size(n)
    {
#ifdef ZMQ_EXTENDED_CONSTEXPR
        assert(p != nullptr || n == 0);
#endif
    }
    constexpr const_buffer(const mutable_buffer &mb) noexcept :
        _data(mb.data()), _size(mb.size())
    {
    }

    constexpr const void *data() const noexcept { return _data; }
    constexpr size_t size() const noexcept { return _size; }
    const_buffer &operator+=(size_t n) noexcept
    {
        const auto shift = (std::min)(n, _size);
        _data = static_cast<const char *>(_data) + shift;
        _size -= shift;
        return *this;
    }

  private:
    const void *_data;
    size_t _size;
};

inline const_buffer operator+(const const_buffer &cb, size_t n) noexcept
{
    return const_buffer(static_cast<const char *>(cb.data())
                          + (std::min)(n, cb.size()),
                        cb.size() - (std::min)(n, cb.size()));
}
inline const_buffer operator+(size_t n, const const_buffer &cb) noexcept
{
    return cb + n;
}

// buffer creation

constexpr mutable_buffer buffer(void *p, size_t n) noexcept
{
    return mutable_buffer(p, n);
}
constexpr const_buffer buffer(const void *p, size_t n) noexcept
{
    return const_buffer(p, n);
}
constexpr mutable_buffer buffer(const mutable_buffer &mb) noexcept
{
    return mb;
}
inline mutable_buffer buffer(const mutable_buffer &mb, size_t n) noexcept
{
    return mutable_buffer(mb.data(), (std::min)(mb.size(), n));
}
constexpr const_buffer buffer(const const_buffer &cb) noexcept
{
    return cb;
}
inline const_buffer buffer(const const_buffer &cb, size_t n) noexcept
{
    return const_buffer(cb.data(), (std::min)(cb.size(), n));
}

namespace detail
{
template<class T> struct is_buffer
{
    static constexpr bool value =
      std::is_same<T, const_buffer>::value || std::is_same<T, mutable_buffer>::value;
};

template<class T> struct is_pod_like
{
    // NOTE: The networking draft N4771 section 16.11 requires
    // T in the buffer functions below to be
    // trivially copyable OR standard layout.
    // Here we decide to be conservative and require both.
    static constexpr bool value =
      ZMQ_IS_TRIVIALLY_COPYABLE(T) && std::is_standard_layout<T>::value;
};

template<class C> constexpr auto seq_size(const C &c) noexcept -> decltype(c.size())
{
    return c.size();
}
template<class T, size_t N>
```

```
constexpr size_t seq_size(const T (&/*array*/)[N]) noexcept
{
    return N;
}

template<class Seq>
auto buffer_contiguous_sequence(Seq &&seq) noexcept
  -> decltype(buffer(std::addressof(*std::begin(seq)), size_t{}))
{
    using T = typename std::remove_cv<
      typename std::remove_reference<decltype(*std::begin(seq))>::type>::type;
    static_assert(detail::is_pod_like<T>::value, "T must be POD");

    const auto size = seq_size(seq);
    return buffer(size != 0u ? std::addressof(*std::begin(seq)) : nullptr,
                  size * sizeof(T));
}
template<class Seq>
auto buffer_contiguous_sequence(Seq &&seq, size_t n_bytes) noexcept
  -> decltype(buffer_contiguous_sequence(seq))
{
    using T = typename std::remove_cv<
      typename std::remove_reference<decltype(*std::begin(seq))>::type>::type;
    static_assert(detail::is_pod_like<T>::value, "T must be POD");

    const auto size = seq_size(seq);
    return buffer(size != 0u ? std::addressof(*std::begin(seq)) : nullptr,
                  (std::min)(size * sizeof(T), n_bytes));
}

} // namespace detail

// C array
template<class T, size_t N> mutable_buffer buffer(T (&data)[N]) noexcept
{
    return detail::buffer_contiguous_sequence(data);
}
template<class T, size_t N>
mutable_buffer buffer(T (&data)[N], size_t n_bytes) noexcept
{
    return detail::buffer_contiguous_sequence(data, n_bytes);
}
template<class T, size_t N> const_buffer buffer(const T (&data)[N]) noexcept
{
    return detail::buffer_contiguous_sequence(data);
}
template<class T, size_t N>
const_buffer buffer(const T (&data)[N], size_t n_bytes) noexcept
{
    return detail::buffer_contiguous_sequence(data, n_bytes);
}
// std::array
template<class T, size_t N> mutable_buffer buffer(std::array<T, N> &data) noexcept
{
    return detail::buffer_contiguous_sequence(data);
}
template<class T, size_t N>
mutable_buffer buffer(std::array<T, N> &data, size_t n_bytes) noexcept
{
    return detail::buffer_contiguous_sequence(data, n_bytes);
}
template<class T, size_t N>
const_buffer buffer(std::array<const T, N> &data) noexcept
{
    return detail::buffer_contiguous_sequence(data);
}
template<class T, size_t N>
const_buffer buffer(std::array<const T, N> &data, size_t n_bytes) noexcept
{
    return detail::buffer_contiguous_sequence(data, n_bytes);
}
template<class T, size_t N>
const_buffer buffer(const std::array<T, N> &data) noexcept
{
    return detail::buffer_contiguous_sequence(data);
}
template<class T, size_t N>
const_buffer buffer(const std::array<T, N> &data, size_t n_bytes) noexcept
{
    return detail::buffer_contiguous_sequence(data, n_bytes);
}
// std::vector
template<class T, class Allocator>
mutable_buffer buffer(std::vector<T, Allocator> &data) noexcept
{
    return detail::buffer_contiguous_sequence(data);
}
```

```
template<class T, class Allocator>
mutable_buffer buffer(std::vector<T, Allocator> &data, size_t n_bytes) noexcept
{
    return detail::buffer_contiguous_sequence(data, n_bytes);
}
template<class T, class Allocator>
const_buffer buffer(const std::vector<T, Allocator> &data) noexcept
{
    return detail::buffer_contiguous_sequence(data);
}
template<class T, class Allocator>
const_buffer buffer(const std::vector<T, Allocator> &data, size_t n_bytes) noexcept
{
    return detail::buffer_contiguous_sequence(data, n_bytes);
}
// std::basic_string
template<class T, class Traits, class Allocator>
mutable_buffer buffer(std::basic_string<T, Traits, Allocator> &data) noexcept
{
    return detail::buffer_contiguous_sequence(data);
}
template<class T, class Traits, class Allocator>
mutable_buffer buffer(std::basic_string<T, Traits, Allocator> &data,
                      size_t n_bytes) noexcept
{
    return detail::buffer_contiguous_sequence(data, n_bytes);
}
template<class T, class Traits, class Allocator>
const_buffer buffer(const std::basic_string<T, Traits, Allocator> &data) noexcept
{
    return detail::buffer_contiguous_sequence(data);
}
template<class T, class Traits, class Allocator>
const_buffer buffer(const std::basic_string<T, Traits, Allocator> &data,
                    size_t n_bytes) noexcept
{
    return detail::buffer_contiguous_sequence(data, n_bytes);
}

#if CPPZMQ_HAS_STRING_VIEW
// std::basic_string_view
template<class T, class Traits>
const_buffer buffer(std::basic_string_view<T, Traits> data) noexcept
{
    return detail::buffer_contiguous_sequence(data);
}
template<class T, class Traits>
const_buffer buffer(std::basic_string_view<T, Traits> data, size_t n_bytes) noexcept
{
    return detail::buffer_contiguous_sequence(data, n_bytes);
}
#endif

// Buffer for a string literal (null terminated)
// where the buffer size excludes the terminating character.
// Equivalent to zmq::buffer(std::string_view("...")).
template<class Char, size_t N>
constexpr const_buffer str_buffer(const Char (&data)[N]) noexcept
{
    static_assert(detail::is_pod_like<Char>::value, "Char must be POD");
#ifdef ZMQ_EXTENDED_CONSTEXPR
    assert(data[N - 1] == Char{0});
#endif
    return const_buffer(static_cast<const Char *>(data), (N - 1) * sizeof(Char));
}

namespace literals
{
constexpr const_buffer operator"" _zbuf(const char *str, size_t len) noexcept
{
    return const_buffer(str, len * sizeof(char));
}
constexpr const_buffer operator"" _zbuf(const wchar_t *str, size_t len) noexcept
{
    return const_buffer(str, len * sizeof(wchar_t));
}
constexpr const_buffer operator"" _zbuf(const char16_t *str, size_t len) noexcept
{
    return const_buffer(str, len * sizeof(char16_t));
}
constexpr const_buffer operator"" _zbuf(const char32_t *str, size_t len) noexcept
{
    return const_buffer(str, len * sizeof(char32_t));
}
}

#ifdef ZMQ_CPP11
```

```
enum class socket_type : int
{
    req = ZMQ_REQ,
    rep = ZMQ_REP,
    dealer = ZMQ_DEALER,
    router = ZMQ_ROUTER,
    pub = ZMQ_PUB,
    sub = ZMQ_SUB,
    xpub = ZMQ_XPUB,
    xsub = ZMQ_XSUB,
    push = ZMQ_PUSH,
    pull = ZMQ_PULL,
#if defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 0)
    server = ZMQ_SERVER,
    client = ZMQ_CLIENT,
    radio = ZMQ_RADIO,
    dish = ZMQ_DISH,
    gather = ZMQ_GATHER,
    scatter = ZMQ_SCATTER,
    dgram = ZMQ_DGRAM,
#endif
#if defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 3, 3)
    peer = ZMQ_PEER,
    channel = ZMQ_CHANNEL,
#endif
#if ZMQ_VERSION_MAJOR >= 4
    stream = ZMQ_STREAM,
#endif
    pair = ZMQ_PAIR
};
#endif

namespace sockopt
{
// There are two types of options,
// integral type with known compiler time size (int, bool, int64_t, uint64_t)
// and arrays with dynamic size (strings, binary data).

// BoolUnit: if true accepts values of type bool (but passed as T into libzmq)
template<int Opt, class T, bool BoolUnit = false> struct integral_option
{
};

// NullTerm:
// 0: binary data
// 1: null-terminated string (`getsockopt` size includes null)
// 2: binary (size 32) or Z85 encoder string of size 41 (null included)
template<int Opt, int NullTerm = 1> struct array_option
{
};

#define ZMQ_DEFINE_INTEGRAL_OPT(OPT, NAME, TYPE)                                    \
    using NAME##_t = integral_option<OPT, TYPE, false>;                             \
    ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {}
#define ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(OPT, NAME, TYPE)                          \
    using NAME##_t = integral_option<OPT, TYPE, true>;                              \
    ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {}
#define ZMQ_DEFINE_ARRAY_OPT(OPT, NAME)                                             \
    using NAME##_t = array_option<OPT>;                                             \
    ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {}
#define ZMQ_DEFINE_ARRAY_OPT_BINARY(OPT, NAME)                                      \
    using NAME##_t = array_option<OPT, 0>;                                          \
    ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {}
#define ZMQ_DEFINE_ARRAY_OPT_BIN_OR_Z85(OPT, NAME)                                  \
    using NAME##_t = array_option<OPT, 2>;                                          \
    ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {}

// deprecated, use zmq::fd_t
using cppzmq_fd_t = ::zmq::fd_t;

#ifdef ZMQ_AFFINITY
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_AFFINITY, affinity, uint64_t);
#endif
#ifdef ZMQ_BACKLOG
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_BACKLOG, backlog, int);
#endif
#ifdef ZMQ_BINDTODEVICE
ZMQ_DEFINE_ARRAY_OPT_BINARY(ZMQ_BINDTODEVICE, bindtodevice);
#endif
#ifdef ZMQ_CONFLATE
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_CONFLATE, conflate, int);
#endif
#ifdef ZMQ_CONNECT_ROUTING_ID
ZMQ_DEFINE_ARRAY_OPT(ZMQ_CONNECT_ROUTING_ID, connect_routing_id);
#endif
#ifdef ZMQ_CONNECT_TIMEOUT
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_CONNECT_TIMEOUT, connect_timeout, int);
```

```
#endif
#ifdef ZMQ_CURVE_PUBLICKEY
ZMQ_DEFINE_ARRAY_OPT_BIN_OR_Z85(ZMQ_CURVE_PUBLICKEY, curve_publickey);
#endif
#ifdef ZMQ_CURVE_SECRETKEY
ZMQ_DEFINE_ARRAY_OPT_BIN_OR_Z85(ZMQ_CURVE_SECRETKEY, curve_secretkey);
#endif
#ifdef ZMQ_CURVE_SERVER
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_CURVE_SERVER, curve_server, int);
#endif
#ifdef ZMQ_CURVE_SERVERKEY
ZMQ_DEFINE_ARRAY_OPT_BIN_OR_Z85(ZMQ_CURVE_SERVERKEY, curve_serverkey);
#endif
#ifdef ZMQ_DISCONNECT_MSG
ZMQ_DEFINE_ARRAY_OPT_BINARY(ZMQ_DISCONNECT_MSG, disconnect_msg);
#endif
#ifdef ZMQ_EVENTS
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_EVENTS, events, int);
#endif
#ifdef ZMQ_FD
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_FD, fd, ::zmq::fd_t);
#endif
#ifdef ZMQ_GSSAPI_PLAINTEXT
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_GSSAPI_PLAINTEXT, gssapi_plaintext, int);
#endif
#ifdef ZMQ_GSSAPI_SERVER
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_GSSAPI_SERVER, gssapi_server, int);
#endif
#ifdef ZMQ_GSSAPI_SERVICE_PRINCIPAL
ZMQ_DEFINE_ARRAY_OPT(ZMQ_GSSAPI_SERVICE_PRINCIPAL, gssapi_service_principal);
#endif
#ifdef ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAMETYPE
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAMETYPE,
                        gssapi_service_principal_nametype,
                        int);
#endif
#ifdef ZMQ_GSSAPI_PRINCIPAL
ZMQ_DEFINE_ARRAY_OPT(ZMQ_GSSAPI_PRINCIPAL, gssapi_principal);
#endif
#ifdef ZMQ_GSSAPI_PRINCIPAL_NAMETYPE
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_GSSAPI_PRINCIPAL_NAMETYPE,
                        gssapi_principal_nametype,
                        int);
#endif
#ifdef ZMQ_HANDSHAKE_IVL
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_HANDSHAKE_IVL, handshake_ivl, int);
#endif
#ifdef ZMQ_HEARTBEAT_IVL
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_HEARTBEAT_IVL, heartbeat_ivl, int);
#endif
#ifdef ZMQ_HEARTBEAT_TIMEOUT
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_HEARTBEAT_TIMEOUT, heartbeat_timeout, int);
#endif
#ifdef ZMQ_HEARTBEAT_TTL
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_HEARTBEAT_TTL, heartbeat_ttl, int);
#endif
#ifdef ZMQ_HELLO_MSG
ZMQ_DEFINE_ARRAY_OPT_BINARY(ZMQ_HELLO_MSG, hello_msg);
#endif
#ifdef ZMQ_IMMEDIATE
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_IMMEDIATE, immediate, int);
#endif
#ifdef ZMQ_INVERT_MATCHING
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_INVERT_MATCHING, invert_matching, int);
#endif
#ifdef ZMQ_IPV6
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_IPV6, ipv6, int);
#endif
#ifdef ZMQ_LAST_ENDPOINT
ZMQ_DEFINE_ARRAY_OPT(ZMQ_LAST_ENDPOINT, last_endpoint);
#endif
#ifdef ZMQ_LINGER
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_LINGER, linger, int);
#endif
#ifdef ZMQ_MAXMSGSIZE
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_MAXMSGSIZE, maxmsgsize, int64_t);
#endif
#ifdef ZMQ_MECHANISM
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_MECHANISM, mechanism, int);
#endif
#ifdef ZMQ_METADATA
ZMQ_DEFINE_ARRAY_OPT(ZMQ_METADATA, metadata);
#endif
#ifdef ZMQ_MULTICAST_HOPS
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_MULTICAST_HOPS, multicast_hops, int);
#endif
#ifdef ZMQ_MULTICAST_LOOP
```

```
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_MULTICAST_LOOP, multicast_loop, int);
#endif
#ifdef ZMQ_MULTICAST_MAXTPDU
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_MULTICAST_MAXTPDU, multicast_maxtpdu, int);
#endif
#ifdef ZMQ_ONLY_FIRST_SUBSCRIBE
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_ONLY_FIRST_SUBSCRIBE, only_first_subscribe, int);
#endif
#ifdef ZMQ_PLAIN_SERVER
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_PLAIN_SERVER, plain_server, int);
#endif
#ifdef ZMQ_PLAIN_PASSWORD
ZMQ_DEFINE_ARRAY_OPT(ZMQ_PLAIN_PASSWORD, plain_password);
#endif
#ifdef ZMQ_PLAIN_USERNAME
ZMQ_DEFINE_ARRAY_OPT(ZMQ_PLAIN_USERNAME, plain_username);
#endif
#ifdef ZMQ_PRIORITY
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_PRIORITY, priority, int);
#endif
#ifdef ZMQ_USE_FD
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_USE_FD, use_fd, int);
#endif
#ifdef ZMQ_PROBE_ROUTER
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_PROBE_ROUTER, probe_router, int);
#endif
#ifdef ZMQ_RATE
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RATE, rate, int);
#endif
#ifdef ZMQ_RCVBUF
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RCVBUF, rcvbuf, int);
#endif
#ifdef ZMQ_RCVHWM
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RCVHWM, rcvhwm, int);
#endif
#ifdef ZMQ_RCVMORE
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_RCVMORE, rcvmore, int);
#endif
#ifdef ZMQ_RCVTIMEO
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RCVTIMEO, rcvtimeo, int);
#endif
#ifdef ZMQ_RECONNECT_IVL
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RECONNECT_IVL, reconnect_ivl, int);
#endif
#ifdef ZMQ_RECONNECT_IVL_MAX
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RECONNECT_IVL_MAX, reconnect_ivl_max, int);
#endif
#ifdef ZMQ_RECONNECT_STOP
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RECONNECT_STOP, reconnect_stop, int);
#endif
#ifdef ZMQ_RECOVERY_IVL
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RECOVERY_IVL, recovery_ivl, int);
#endif
#ifdef ZMQ_REQ_CORRELATE
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_REQ_CORRELATE, req_correlate, int);
#endif
#ifdef ZMQ_REQ_RELAXED
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_REQ_RELAXED, req_relaxed, int);
#endif
#ifdef ZMQ_ROUTER_HANDOVER
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_ROUTER_HANDOVER, router_handover, int);
#endif
#ifdef ZMQ_ROUTER_MANDATORY
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_ROUTER_MANDATORY, router_mandatory, int);
#endif
#ifdef ZMQ_ROUTER_NOTIFY
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_ROUTER_NOTIFY, router_notify, int);
#endif
#ifdef ZMQ_ROUTING_ID
ZMQ_DEFINE_ARRAY_OPT_BINARY(ZMQ_ROUTING_ID, routing_id);
#endif
#ifdef ZMQ_SNDBUF
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_SNDBUF, sndbuf, int);
#endif
#ifdef ZMQ_SNDHWM
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_SNDHWM, sndhwm, int);
#endif
#ifdef ZMQ_SNDTIMEO
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_SNDTIMEO, sndtimeo, int);
#endif
#ifdef ZMQ_SOCKS_PASSWORD
ZMQ_DEFINE_ARRAY_OPT(ZMQ_SOCKS_PASSWORD, socks_password);
#endif
#ifdef ZMQ_SOCKS_PROXY
ZMQ_DEFINE_ARRAY_OPT(ZMQ_SOCKS_PROXY, socks_proxy);
#endif
#ifdef ZMQ_SOCKS_USERNAME
```

```
ZMQ_DEFINE_ARRAY_OPT(ZMQ_SOCKS_USERNAME, socks_username);
#endif
#ifdef ZMQ_STREAM_NOTIFY
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_STREAM_NOTIFY, stream_notify, int);
#endif
#ifdef ZMQ_SUBSCRIBE
ZMQ_DEFINE_ARRAY_OPT(ZMQ_SUBSCRIBE, subscribe);
#endif
#ifdef ZMQ_TCP_KEEPALIVE
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_KEEPALIVE, tcp_keepalive, int);
#endif
#ifdef ZMQ_TCP_KEEPALIVE_CNT
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_KEEPALIVE_CNT, tcp_keepalive_cnt, int);
#endif
#ifdef ZMQ_TCP_KEEPALIVE_IDLE
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_KEEPALIVE_IDLE, tcp_keepalive_idle, int);
#endif
#ifdef ZMQ_TCP_KEEPALIVE_INTVL
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_KEEPALIVE_INTVL, tcp_keepalive_intvl, int);
#endif
#ifdef ZMQ_TCP_MAXRT
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_MAXRT, tcp_maxrt, int);
#endif
#ifdef ZMQ_THREAD_SAFE
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_THREAD_SAFE, thread_safe, int);
#endif
#ifdef ZMQ_TOS
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TOS, tos, int);
#endif
#ifdef ZMQ_TYPE
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TYPE, type, int);
#ifdef ZMQ_CPP11
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TYPE, socket_type, socket_type);
#endif // ZMQ_CPP11
#endif // ZMQ_TYPE
#ifdef ZMQ_UNSUBSCRIBE
ZMQ_DEFINE_ARRAY_OPT(ZMQ_UNSUBSCRIBE, unsubscribe);
#endif
#ifdef ZMQ_VMCI_BUFFER_SIZE
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_VMCI_BUFFER_SIZE, vmci_buffer_size, uint64_t);
#endif
#ifdef ZMQ_VMCI_BUFFER_MIN_SIZE
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_VMCI_BUFFER_MIN_SIZE, vmci_buffer_min_size, uint64_t);
#endif
#ifdef ZMQ_VMCI_BUFFER_MAX_SIZE
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_VMCI_BUFFER_MAX_SIZE, vmci_buffer_max_size, uint64_t);
#endif
#ifdef ZMQ_VMCI_CONNECT_TIMEOUT
ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_VMCI_CONNECT_TIMEOUT, vmci_connect_timeout, int);
#endif
#ifdef ZMQ_XPUB_VERBOSE
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_VERBOSE, xpub_verbose, int);
#endif
#ifdef ZMQ_XPUB_VERBOSER
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_VERBOSER, xpub_verboser, int);
#endif
#ifdef ZMQ_XPUB_MANUAL
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_MANUAL, xpub_manual, int);
#endif
#ifdef ZMQ_XPUB_MANUAL_LAST_VALUE
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_MANUAL_LAST_VALUE, xpub_manual_last_value, int);
#endif
#ifdef ZMQ_XPUB_NODROP
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_NODROP, xpub_nodrop, int);
#endif
#ifdef ZMQ_XPUB_WELCOME_MSG
ZMQ_DEFINE_ARRAY_OPT(ZMQ_XPUB_WELCOME_MSG, xpub_welcome_msg);
#endif
#ifdef ZMQ_ZAP_ENFORCE_DOMAIN
ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_ZAP_ENFORCE_DOMAIN, zap_enforce_domain, int);
#endif
#ifdef ZMQ_ZAP_DOMAIN
ZMQ_DEFINE_ARRAY_OPT(ZMQ_ZAP_DOMAIN, zap_domain);
#endif

} // namespace sockopt
#endif // ZMQ_CPP11


namespace detail
{
class socket_base
{
  public:
    socket_base() ZMQ_NOTHROW : _handle(ZMQ_NULLPTR) {}
    ZMQ_EXPLICIT socket_base(void *handle) ZMQ_NOTHROW : _handle(handle) {}

    template<typename T>
    ZMQ_CPP11_DEPRECATED("from 4.7.0, use `set` taking option from zmq::sockopt")
    void setsockopt(int option_, T const &optval)
    {
        setsockopt(option_, &optval, sizeof(T));
    }

    ZMQ_CPP11_DEPRECATED("from 4.7.0, use `set` taking option from zmq::sockopt")
    void setsockopt(int option_, const void *optval_, size_t optvallen_)
    {
        int rc = zmq_setsockopt(_handle, option_, optval_, optvallen_);
        if (rc != 0)
            throw error_t();
    }

    ZMQ_CPP11_DEPRECATED("from 4.7.0, use `get` taking option from zmq::sockopt")
    void getsockopt(int option_, void *optval_, size_t *optvallen_) const
    {
        int rc = zmq_getsockopt(_handle, option_, optval_, optvallen_);
        if (rc != 0)
            throw error_t();
    }

    template<typename T>
    ZMQ_CPP11_DEPRECATED("from 4.7.0, use `get` taking option from zmq::sockopt")
    T getsockopt(int option_) const
    {
        T optval;
        size_t optlen = sizeof(T);
        getsockopt(option_, &optval, &optlen);
        return optval;
    }

#ifdef ZMQ_CPP11
    // Set integral socket option, e.g.
    // `socket.set(zmq::sockopt::linger, 0)`
    template<int Opt, class T, bool BoolUnit>
    void set(sockopt::integral_option<Opt, T, BoolUnit>, const T &val)
    {
        static_assert(std::is_integral<T>::value, "T must be integral");
        set_option(Opt, &val, sizeof val);
    }

    // Set integral socket option from boolean, e.g.
    // `socket.set(zmq::sockopt::immediate, false)`
    template<int Opt, class T>
    void set(sockopt::integral_option<Opt, T, true>, bool val)
    {
        static_assert(std::is_integral<T>::value, "T must be integral");
        T rep_val = val;
        set_option(Opt, &rep_val, sizeof rep_val);
    }

    // Set array socket option, e.g.
    // `socket.set(zmq::sockopt::plain_username, "foo123")`
    template<int Opt, int NullTerm>
    void set(sockopt::array_option<Opt, NullTerm>, const char *buf)
    {
        set_option(Opt, buf, std::strlen(buf));
    }

    // Set array socket option, e.g.
    // `socket.set(zmq::sockopt::routing_id, zmq::buffer(id))`
    template<int Opt, int NullTerm>
    void set(sockopt::array_option<Opt, NullTerm>, const_buffer buf)
    {
        set_option(Opt, buf.data(), buf.size());
    }

    // Set array socket option, e.g.
    // `socket.set(zmq::sockopt::routing_id, id_str)`
    template<int Opt, int NullTerm>
    void set(sockopt::array_option<Opt, NullTerm>, const std::string &buf)
    {
        set_option(Opt, buf.data(), buf.size());
    }

#if CPPZMQ_HAS_STRING_VIEW
    // Set array socket option, e.g.
    // `socket.set(zmq::sockopt::routing_id, id_str)`
    template<int Opt, int NullTerm>
    void set(sockopt::array_option<Opt, NullTerm>, std::string_view buf)
    {
        set_option(Opt, buf.data(), buf.size());
    }
#endif

```

```
    // Get scalar socket option, e.g.
    // `auto opt = socket.get(zmq::sockopt::linger)`
    template<int Opt, class T, bool BoolUnit>
    ZMQ_NODISCARD T get(sockopt::integral_option<Opt, T, BoolUnit>) const
    {
        static_assert(std::is_scalar<T>::value, "T must be scalar");
        T val;
        size_t size = sizeof val;
        get_option(Opt, &val, &size);
        assert(size == sizeof val);
        return val;
    }

    // Get array socket option, writes to buf, returns option size in bytes, e.g.
    // `size_t optsize = socket.get(zmq::sockopt::routing_id, zmq::buffer(id))`
    template<int Opt, int NullTerm>
    ZMQ_NODISCARD size_t get(sockopt::array_option<Opt, NullTerm>,
                             mutable_buffer buf) const
    {
        size_t size = buf.size();
        get_option(Opt, buf.data(), &size);
        return size;
    }

    // Get array socket option as string (initializes the string buffer size to init_size) e.g.
    // `auto s = socket.get(zmq::sockopt::routing_id)`
    // Note: removes the null character from null-terminated string options,
    // i.e. the string size excludes the null character.
    template<int Opt, int NullTerm>
    ZMQ_NODISCARD std::string get(sockopt::array_option<Opt, NullTerm>,
                                  size_t init_size = 1024) const
    {
        if ZMQ_CONSTEXPR_IF (NullTerm == 2) {
            if (init_size == 1024) {
                init_size = 41; // get as Z85 string
            }
        }
        std::string str(init_size, '\0');
        size_t size = get(sockopt::array_option<Opt>{}, buffer(str));
        if ZMQ_CONSTEXPR_IF (NullTerm == 1) {
            if (size > 0) {
                assert(str[size - 1] == '\0');
                --size;
            }
        } else if ZMQ_CONSTEXPR_IF (NullTerm == 2) {
            assert(size == 32 || size == 41);
            if (size == 41) {
                assert(str[size - 1] == '\0');
                --size;
            }
        }
        str.resize(size);
        return str;
    }
#endif

    void bind(std::string const &addr) { bind(addr.c_str()); }

    void bind(const char *addr_)
    {
        int rc = zmq_bind(_handle, addr_);
        if (rc != 0)
            throw error_t();
    }

    void unbind(std::string const &addr) { unbind(addr.c_str()); }

    void unbind(const char *addr_)
    {
        int rc = zmq_unbind(_handle, addr_);
        if (rc != 0)
            throw error_t();
    }

    void connect(std::string const &addr) { connect(addr.c_str()); }

    void connect(const char *addr_)
    {
        int rc = zmq_connect(_handle, addr_);
        if (rc != 0)
            throw error_t();
    }

    void disconnect(std::string const &addr) { disconnect(addr.c_str()); }

    void disconnect(const char *addr_)
    {
        int rc = zmq_disconnect(_handle, addr_);
        if (rc != 0)
            throw error_t();
    }

    ZMQ_DEPRECATED("from 4.7.1, use handle() != nullptr or operator bool")
    bool connected() const ZMQ_NOTHROW { return (_handle != ZMQ_NULLPTR); }

    ZMQ_CPP11_DEPRECATED("from 4.3.1, use send taking a const_buffer and send_flags")
    size_t send(const void *buf_, size_t len_, int flags_ = 0)
    {
        int nbytes = zmq_send(_handle, buf_, len_, flags_);
        if (nbytes >= 0)
            return static_cast<size_t>(nbytes);
        if (zmq_errno() == EAGAIN)
            return 0;
        throw error_t();
    }

    ZMQ_CPP11_DEPRECATED("from 4.3.1, use send taking message_t and send_flags")
    bool send(message_t &msg_,
              int flags_ = 0) // default until removed
    {
        int nbytes = zmq_msg_send(msg_.handle(), _handle, flags_);
        if (nbytes >= 0)
            return true;
        if (zmq_errno() == EAGAIN)
            return false;
        throw error_t();
    }

    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, T last, int flags_ = 0)
    {
        zmq::message_t msg(first, last);
        int nbytes = zmq_msg_send(msg.handle(), _handle, flags_);
        if (nbytes >= 0)
            return true;
        if (zmq_errno() == EAGAIN)
            return false;
        throw error_t();
    }

#ifdef ZMQ_HAS_RVALUE_REFS
    ZMQ_CPP11_DEPRECATED("from 4.3.1, use send taking message_t and send_flags")
    bool send(message_t &&msg_,
              int flags_ = 0) // default until removed
    {
#ifdef ZMQ_CPP11
        return send(msg_, static_cast<send_flags>(flags_)).has_value();
#else
        return send(msg_, flags_);
#endif
    }
#endif

#ifdef ZMQ_CPP11
    send_result_t send(const_buffer buf, send_flags flags = send_flags::none)
    {
        const int nbytes =
          zmq_send(_handle, buf.data(), buf.size(), static_cast<int>(flags));
        if (nbytes >= 0)
            return static_cast<size_t>(nbytes);
        if (zmq_errno() == EAGAIN)
            return {};
        throw error_t();
    }

    send_result_t send(message_t &msg, send_flags flags)
    {
        int nbytes = zmq_msg_send(msg.handle(), _handle, static_cast<int>(flags));
        if (nbytes >= 0)
            return static_cast<size_t>(nbytes);
        if (zmq_errno() == EAGAIN)
            return {};
        throw error_t();
    }

    send_result_t send(message_t &&msg, send_flags flags)
    {
        return send(msg, flags);
    }
#endif

```

```
    ZMQ_CPP11_DEPRECATED(
      "from 4.3.1, use recv taking a mutable_buffer and recv_flags")
    size_t recv(void *buf_, size_t len_, int flags_ = 0)
    {
        int nbytes = zmq_recv(_handle, buf_, len_, flags_);
        if (nbytes >= 0)
            return static_cast<size_t>(nbytes);
        if (zmq_errno() == EAGAIN)
            return 0;
        throw error_t();
    }

    ZMQ_CPP11_DEPRECATED(
      "from 4.3.1, use recv taking a reference to message_t and recv_flags")
    bool recv(message_t *msg_, int flags_ = 0)
    {
        int nbytes = zmq_msg_recv(msg_->handle(), _handle, flags_);
        if (nbytes >= 0)
            return true;
        if (zmq_errno() == EAGAIN)
            return false;
        throw error_t();
    }

#ifdef ZMQ_CPP11
    ZMQ_NODISCARD
    recv_buffer_result_t recv(mutable_buffer buf,
                              recv_flags flags = recv_flags::none)
    {
        const int nbytes =
          zmq_recv(_handle, buf.data(), buf.size(), static_cast<int>(flags));
        if (nbytes >= 0) {
            return recv_buffer_size{
              (std::min)(static_cast<size_t>(nbytes), buf.size()),
              static_cast<size_t>(nbytes)};
        }
        if (zmq_errno() == EAGAIN)
            return {};
        throw error_t();
    }

    ZMQ_NODISCARD
    recv_result_t recv(message_t &msg, recv_flags flags = recv_flags::none)
    {
        const int nbytes =
          zmq_msg_recv(msg.handle(), _handle, static_cast<int>(flags));
        if (nbytes >= 0) {
            assert(msg.size() == static_cast<size_t>(nbytes));
            return static_cast<size_t>(nbytes);
        }
        if (zmq_errno() == EAGAIN)
            return {};
        throw error_t();
    }
#endif

#if defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 0)
    void join(const char *group)
    {
        int rc = zmq_join(_handle, group);
        if (rc != 0)
            throw error_t();
    }

    void leave(const char *group)
    {
        int rc = zmq_leave(_handle, group);
        if (rc != 0)
            throw error_t();
    }
#endif

    ZMQ_NODISCARD void *handle() ZMQ_NOTHROW { return _handle; }
    ZMQ_NODISCARD const void *handle() const ZMQ_NOTHROW { return _handle; }

    ZMQ_EXPLICIT operator bool() const ZMQ_NOTHROW { return _handle != ZMQ_NULLPTR; }
    // note: non-const operator bool can be removed once
    // operator void* is removed from socket_t
    ZMQ_EXPLICIT operator bool() ZMQ_NOTHROW { return _handle != ZMQ_NULLPTR; }

  protected:
    void *_handle;

  private:
    void set_option(int option_, const void *optval_, size_t optvallen_)
    {
        int rc = zmq_setsockopt(_handle, option_, optval_, optvallen_);
        if (rc != 0)
            throw error_t();
    }

    void get_option(int option_, void *optval_, size_t *optvallen_) const
    {
        int rc = zmq_getsockopt(_handle, option_, optval_, optvallen_);
        if (rc != 0)
            throw error_t();
    }
};
} // namespace detail

struct from_handle_t
{
    struct _private
    {
    }; // disabling use other than with from_handle
    ZMQ_CONSTEXPR_FN ZMQ_EXPLICIT from_handle_t(_private /*p*/) ZMQ_NOTHROW {}
};

ZMQ_CONSTEXPR_VAR from_handle_t from_handle =
  from_handle_t(from_handle_t::_private());

// A non-owning nullable reference to a socket.
// The reference is invalidated on socket close or destruction.
class socket_ref : public detail::socket_base
{
  public:
    socket_ref() ZMQ_NOTHROW : detail::socket_base() {}
#ifdef ZMQ_CPP11
    socket_ref(std::nullptr_t) ZMQ_NOTHROW : detail::socket_base() {}
#endif
    socket_ref(from_handle_t /*fh*/, void *handle) ZMQ_NOTHROW
        : detail::socket_base(handle)
    {
    }
};

#ifdef ZMQ_CPP11
inline bool operator==(socket_ref sr, std::nullptr_t /*p*/) ZMQ_NOTHROW
{
    return sr.handle() == nullptr;
}
inline bool operator==(std::nullptr_t /*p*/, socket_ref sr) ZMQ_NOTHROW
{
    return sr.handle() == nullptr;
}
inline bool operator!=(socket_ref sr, std::nullptr_t /*p*/) ZMQ_NOTHROW
{
    return !(sr == nullptr);
}
inline bool operator!=(std::nullptr_t /*p*/, socket_ref sr) ZMQ_NOTHROW
{
    return !(sr == nullptr);
}
#endif

inline bool operator==(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
{
    return std::equal_to<const void *>()(a.handle(), b.handle());
}
inline bool operator!=(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
{
    return !(a == b);
}
inline bool operator<(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
{
    return std::less<const void *>()(a.handle(), b.handle());
}
inline bool operator>(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
{
    return b < a;
}
inline bool operator<=(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
{
    return !(a > b);
}
inline bool operator>=(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
{
    return !(a < b);
}

} // namespace zmq

#ifdef ZMQ_CPP11
namespace std
```

```
{
template<> struct hash<zmq::socket_ref>
{
    size_t operator()(zmq::socket_ref sr) const ZMQ_NOTHROW
    {
        return hash<void *>()(sr.handle());
    }
};
} // namespace std
#endif

namespace zmq
{
class socket_t : public detail::socket_base
{
    friend class monitor_t;

  public:
    socket_t() ZMQ_NOTHROW : detail::socket_base(ZMQ_NULLPTR), ctxptr(ZMQ_NULLPTR) {}

    socket_t(context_t &context_, int type_) :
        detail::socket_base(zmq_socket(context_.handle(), type_)),
        ctxptr(context_.handle())
    {
        if (_handle == ZMQ_NULLPTR)
            throw error_t();
    }

#ifdef ZMQ_CPP11
    socket_t(context_t &context_, socket_type type_) :
        socket_t(context_, static_cast<int>(type_))
    {
    }
#endif

#ifdef ZMQ_HAS_RVALUE_REFS
    socket_t(socket_t &&rhs) ZMQ_NOTHROW : detail::socket_base(rhs._handle),
                                           ctxptr(rhs.ctxptr)
    {
        rhs._handle = ZMQ_NULLPTR;
        rhs.ctxptr = ZMQ_NULLPTR;
    }
    socket_t &operator=(socket_t &&rhs) ZMQ_NOTHROW
    {
        close();
        std::swap(_handle, rhs._handle);
        std::swap(ctxptr, rhs.ctxptr);
        return *this;
    }
#endif

    ~socket_t() ZMQ_NOTHROW { close(); }

    operator void *() ZMQ_NOTHROW { return _handle; }

    operator void const *() const ZMQ_NOTHROW { return _handle; }

    void close() ZMQ_NOTHROW
    {
        if (_handle == ZMQ_NULLPTR)
            // already closed
            return;
        int rc = zmq_close(_handle);
        ZMQ_ASSERT(rc == 0);
        _handle = ZMQ_NULLPTR;
        ctxptr = ZMQ_NULLPTR;
    }

    void swap(socket_t &other) ZMQ_NOTHROW
    {
        std::swap(_handle, other._handle);
        std::swap(ctxptr, other.ctxptr);
    }

    operator socket_ref() ZMQ_NOTHROW { return socket_ref(from_handle, _handle); }

  private:
    void *ctxptr;

    socket_t(const socket_t &) ZMQ_DELETED_FUNCTION;
    void operator=(const socket_t &) ZMQ_DELETED_FUNCTION;

    // used by monitor_t
    socket_t(void *context_, int type_) :
        detail::socket_base(zmq_socket(context_, type_)), ctxptr(context_)
    {
        if (_handle == ZMQ_NULLPTR)
```

```
            throw error_t();
        if (ctxptr == ZMQ_NULLPTR)
            throw error_t();
    }
};

inline void swap(socket_t &a, socket_t &b) ZMQ_NOTHROW
{
    a.swap(b);
}

ZMQ_DEPRECATED("from 4.3.1, use proxy taking socket_t objects")
inline void proxy(void *frontend, void *backend, void *capture)
{
    int rc = zmq_proxy(frontend, backend, capture);
    if (rc != 0)
        throw error_t();
}

inline void
proxy(socket_ref frontend, socket_ref backend, socket_ref capture = socket_ref())
{
    int rc = zmq_proxy(frontend.handle(), backend.handle(), capture.handle());
    if (rc != 0)
        throw error_t();
}

#ifdef ZMQ_HAS_PROXY_STEERABLE
ZMQ_DEPRECATED("from 4.3.1, use proxy_steerable taking socket_t objects")
inline void
proxy_steerable(void *frontend, void *backend, void *capture, void *control)
{
    int rc = zmq_proxy_steerable(frontend, backend, capture, control);
    if (rc != 0)
        throw error_t();
}

inline void proxy_steerable(socket_ref frontend,
                            socket_ref backend,
                            socket_ref capture,
                            socket_ref control)
{
    int rc = zmq_proxy_steerable(frontend.handle(), backend.handle(),
                                 capture.handle(), control.handle());
    if (rc != 0)
        throw error_t();
}
#endif

class monitor_t
{
  public:
    monitor_t() : _socket(), _monitor_socket() {}

    virtual ~monitor_t() { close(); }

#ifdef ZMQ_HAS_RVALUE_REFS
    monitor_t(monitor_t &&rhs) ZMQ_NOTHROW : _socket(), _monitor_socket()
    {
        std::swap(_socket, rhs._socket);
        std::swap(_monitor_socket, rhs._monitor_socket);
    }

    monitor_t &operator=(monitor_t &&rhs) ZMQ_NOTHROW
    {
        close();
        _socket = socket_ref();
        std::swap(_socket, rhs._socket);
        std::swap(_monitor_socket, rhs._monitor_socket);
        return *this;
    }
#endif


    void
    monitor(socket_t &socket, std::string const &addr, int events = ZMQ_EVENT_ALL)
    {
        monitor(socket, addr.c_str(), events);
    }

    void monitor(socket_t &socket, const char *addr_, int events = ZMQ_EVENT_ALL)
    {
        init(socket, addr_, events);
        while (true) {
            check_event(-1);
        }
    }

    void init(socket_t &socket, std::string const &addr, int events = ZMQ_EVENT_ALL)
    {
        init(socket, addr.c_str(), events);
    }

    void init(socket_t &socket, const char *addr_, int events = ZMQ_EVENT_ALL)
    {
        int rc = zmq_socket_monitor(socket.handle(), addr_, events);
        if (rc != 0)
            throw error_t();

        _socket = socket;
        _monitor_socket = socket_t(socket.ctxptr, ZMQ_PAIR);
        _monitor_socket.connect(addr_);

        on_monitor_started();
    }

    bool check_event(int timeout = 0)
    {
        assert(_monitor_socket);

        zmq::message_t eventMsg;

        zmq::pollitem_t items[] = {
          {_monitor_socket.handle(), 0, ZMQ_POLLIN, 0},
        };

        #ifdef ZMQ_CPP11
        zmq::poll(&items[0], 1, std::chrono::milliseconds(timeout));
        #else
        zmq::poll(&items[0], 1, timeout);
        #endif

        if (items[0].revents & ZMQ_POLLIN) {
            int rc = zmq_msg_recv(eventMsg.handle(), _monitor_socket.handle(), 0);
            if (rc == -1 && zmq_errno() == ETERM)
                return false;
            assert(rc != -1);

        } else {
            return false;
        }

#if ZMQ_VERSION_MAJOR >= 4
        const char *data = static_cast<const char *>(eventMsg.data());
        zmq_event_t msgEvent;
        memcpy(&msgEvent.event, data, sizeof(uint16_t));
        data += sizeof(uint16_t);
        memcpy(&msgEvent.value, data, sizeof(int32_t));
        zmq_event_t *event = &msgEvent;
#else
        zmq_event_t *event = static_cast<zmq_event_t *>(eventMsg.data());
#endif

#ifdef ZMQ_NEW_MONITOR_EVENT_LAYOUT
        zmq::message_t addrMsg;
        int rc = zmq_msg_recv(addrMsg.handle(), _monitor_socket.handle(), 0);
        if (rc == -1 && zmq_errno() == ETERM) {
            return false;
        }

        assert(rc != -1);
        std::string address = addrMsg.to_string();
#else
        // Bit of a hack, but all events in the zmq_event_t union have the same layout so this will work for all event types.
        std::string address = event->data.connected.addr;
#endif

#ifdef ZMQ_EVENT_MONITOR_STOPPED
        if (event->event == ZMQ_EVENT_MONITOR_STOPPED) {
            return false;
        }

#endif

        switch (event->event) {
            case ZMQ_EVENT_CONNECTED:
                on_event_connected(*event, address.c_str());
                break;
            case ZMQ_EVENT_CONNECT_DELAYED:
                on_event_connect_delayed(*event, address.c_str());
                break;
            case ZMQ_EVENT_CONNECT_RETRIED:
                on_event_connect_retried(*event, address.c_str());
                break;
            case ZMQ_EVENT_LISTENING:
                on_event_listening(*event, address.c_str());
                break;
            case ZMQ_EVENT_BIND_FAILED:
                on_event_bind_failed(*event, address.c_str());
                break;
            case ZMQ_EVENT_ACCEPTED:
                on_event_accepted(*event, address.c_str());
                break;
            case ZMQ_EVENT_ACCEPT_FAILED:
                on_event_accept_failed(*event, address.c_str());
                break;
            case ZMQ_EVENT_CLOSED:
                on_event_closed(*event, address.c_str());
                break;
            case ZMQ_EVENT_CLOSE_FAILED:
                on_event_close_failed(*event, address.c_str());
                break;
            case ZMQ_EVENT_DISCONNECTED:
                on_event_disconnected(*event, address.c_str());
                break;
#if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 3, 0) || (defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >=
ZMQ_MAKE_VERSION(4, 2, 3))
            case ZMQ_EVENT_HANDSHAKE_FAILED_NO_DETAIL:
                on_event_handshake_failed_no_detail(*event, address.c_str());
                break;
            case ZMQ_EVENT_HANDSHAKE_FAILED_PROTOCOL:
                on_event_handshake_failed_protocol(*event, address.c_str());
                break;
            case ZMQ_EVENT_HANDSHAKE_FAILED_AUTH:
                on_event_handshake_failed_auth(*event, address.c_str());
                break;
            case ZMQ_EVENT_HANDSHAKE_SUCCEEDED:
                on_event_handshake_succeeded(*event, address.c_str());
                break;
#elif defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 1)
            case ZMQ_EVENT_HANDSHAKE_FAILED:
                on_event_handshake_failed(*event, address.c_str());
                break;
            case ZMQ_EVENT_HANDSHAKE_SUCCEED:
                on_event_handshake_succeed(*event, address.c_str());
                break;
#endif
            default:
                on_event_unknown(*event, address.c_str());
                break;
        }

        return true;
    }

#ifdef ZMQ_EVENT_MONITOR_STOPPED
    void abort()
    {
        if (_socket)
            zmq_socket_monitor(_socket.handle(), ZMQ_NULLPTR, 0);

        _socket = socket_ref();
    }
#endif
    virtual void on_monitor_started() {}
    virtual void on_event_connected(const zmq_event_t &event_, const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
    virtual void on_event_connect_delayed(const zmq_event_t &event_,
                                          const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
    virtual void on_event_connect_retried(const zmq_event_t &event_,
                                          const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
    virtual void on_event_listening(const zmq_event_t &event_, const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
    virtual void on_event_bind_failed(const zmq_event_t &event_, const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
    virtual void on_event_accepted(const zmq_event_t &event_, const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
    virtual void on_event_accept_failed(const zmq_event_t &event_, const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
    virtual void on_event_closed(const zmq_event_t &event_, const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
    virtual void on_event_close_failed(const zmq_event_t &event_, const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
    virtual void on_event_disconnected(const zmq_event_t &event_, const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
#if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 3)
    virtual void on_event_handshake_failed_no_detail(const zmq_event_t &event_,
                                                     const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
    virtual void on_event_handshake_failed_protocol(const zmq_event_t &event_,
                                                    const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
    virtual void on_event_handshake_failed_auth(const zmq_event_t &event_,
                                                const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
    virtual void on_event_handshake_succeeded(const zmq_event_t &event_,
                                              const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
#elif ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 1)
    virtual void on_event_handshake_failed(const zmq_event_t &event_,
                                           const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
    virtual void on_event_handshake_succeed(const zmq_event_t &event_,
                                            const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }
#endif
    virtual void on_event_unknown(const zmq_event_t &event_, const char *addr_)
    {
        (void) event_;
        (void) addr_;
    }

  private:
    monitor_t(const monitor_t &) ZMQ_DELETED_FUNCTION;
    void operator=(const monitor_t &) ZMQ_DELETED_FUNCTION;

    socket_ref _socket;
    socket_t _monitor_socket;

    void close() ZMQ_NOTHROW
    {
        if (_socket)
            zmq_socket_monitor(_socket.handle(), ZMQ_NULLPTR, 0);
        _monitor_socket.close();
    }
};

```

```
#if defined(ZMQ_BUILD_DRAFT_API) && defined(ZMQ_CPP11) && defined(ZMQ_HAVE_POLLER)

// polling events
enum class event_flags : short
{
    none = 0,
    pollin = ZMQ_POLLIN,
    pollout = ZMQ_POLLOUT,
    pollerr = ZMQ_POLLERR,
    pollpri = ZMQ_POLLPRI
};

constexpr event_flags operator|(event_flags a, event_flags b) noexcept
{
    return detail::enum_bit_or(a, b);
}
constexpr event_flags operator&(event_flags a, event_flags b) noexcept
{
    return detail::enum_bit_and(a, b);
}
constexpr event_flags operator^(event_flags a, event_flags b) noexcept
{
    return detail::enum_bit_xor(a, b);
}
constexpr event_flags operator~(event_flags a) noexcept
{
    return detail::enum_bit_not(a);
}

struct no_user_data;

// layout compatible with zmq_poller_event_t
template<class T = no_user_data> struct poller_event
{
    socket_ref socket;
    ::zmq::fd_t fd;
    T *user_data;
    event_flags events;
};

template<typename T = no_user_data> class poller_t
{
  public:
    using event_type = poller_event<T>;

    poller_t() : poller_ptr(zmq_poller_new())
    {
        if (!poller_ptr)
            throw error_t();
    }

    template<
      typename Dummy = void,
      typename =
        typename std::enable_if<!std::is_same<T, no_user_data>::value, Dummy>::type>
    void add(zmq::socket_ref socket, event_flags events, T *user_data)
    {
        add_impl(socket, events, user_data);
    }

    void add(zmq::socket_ref socket, event_flags events)
    {
        add_impl(socket, events, nullptr);
    }

    template<
      typename Dummy = void,
      typename =
        typename std::enable_if<!std::is_same<T, no_user_data>::value, Dummy>::type>
    void add(fd_t fd, event_flags events, T *user_data)
    {
        add_impl(fd, events, user_data);
    }

    void add(fd_t fd, event_flags events) { add_impl(fd, events, nullptr); }

    void remove(zmq::socket_ref socket)
    {
        if (0 != zmq_poller_remove(poller_ptr.get(), socket.handle())) {
            throw error_t();
        }
    }

    void modify(zmq::socket_ref socket, event_flags events)
    {
        if (0
            != zmq_poller_modify(poller_ptr.get(), socket.handle(),
                                static_cast<short>(events))) {
            throw error_t();
        }
    }

    size_t wait_all(std::vector<event_type> &poller_events,
                    const std::chrono::milliseconds timeout)
    {
        int rc = zmq_poller_wait_all(
          poller_ptr.get(),
          reinterpret_cast<zmq_poller_event_t *>(poller_events.data()),
          static_cast<int>(poller_events.size()),
          static_cast<long>(timeout.count()));
        if (rc > 0)
            return static_cast<size_t>(rc);

#if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 3)
        if (zmq_errno() == EAGAIN)
#else
        if (zmq_errno() == ETIMEDOUT)
#endif
            return 0;

        throw error_t();
    }

#if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 3, 3)
    size_t size() const noexcept
    {
        int rc = zmq_poller_size(const_cast<void *>(poller_ptr.get()));
        ZMQ_ASSERT(rc >= 0);
        return static_cast<size_t>(std::max(rc, 0));
    }
#endif

  private:
    struct destroy_poller_t
    {
        void operator()(void *ptr) noexcept
        {
            int rc = zmq_poller_destroy(&ptr);
            ZMQ_ASSERT(rc == 0);
        }
    };

    std::unique_ptr<void, destroy_poller_t> poller_ptr;

    void add_impl(zmq::socket_ref socket, event_flags events, T *user_data)
    {
        if (0
            != zmq_poller_add(poller_ptr.get(), socket.handle(), user_data,
                              static_cast<short>(events))) {
            throw error_t();
        }
    }

    void add_impl(fd_t fd, event_flags events, T *user_data)
    {
        if (0
            != zmq_poller_add_fd(poller_ptr.get(), fd, user_data,
                                 static_cast<short>(events))) {
            throw error_t();
        }
    }
};
#endif //  defined(ZMQ_BUILD_DRAFT_API) && defined(ZMQ_CPP11) && defined(ZMQ_HAVE_POLLER)

inline std::ostream &operator«(std::ostream &os, const message_t &msg)
{
    return os « msg.str();
}

} // namespace zmq

#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.

```
/*
    Copyright (c) 2016-2017 ZeroMQ community
    Copyright (c) 2016 VOCA AS / Harald Nøkland

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to
    deal in the Software without restriction, including without limitation the
    rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
    sell copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
    FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
    IN THE SOFTWARE.
*/

#ifndef __ZMQ_ADDON_HPP_INCLUDED__
#define __ZMQ_ADDON_HPP_INCLUDED__

#include "zmq.hpp"

#include <deque>
#include <iomanip>
#include <sstream>
#include <stdexcept>
#ifdef ZMQ_CPP11
#include <limits>
#include <functional>
#include <unordered_map>
#endif

namespace zmq
{
#ifdef ZMQ_CPP11

namespace detail
{
template<bool CheckN, class OutputIt>
recv_result_t
recv_multipart_n(socket_ref s, OutputIt out, size_t n, recv_flags flags)
{
    size_t msg_count = 0;
    message_t msg;
    while (true) {
        if ZMQ_CONSTEXPR_IF (CheckN) {
            if (msg_count >= n)
                throw std::runtime_error(
                  "Too many message parts in recv_multipart_n");
        }
        if (!s.recv(msg, flags)) {
            // zmq ensures atomic delivery of messages
            assert(msg_count == 0);
            return {};
        }
        ++msg_count;
        const bool more = msg.more();
        *out++ = std::move(msg);
        if (!more)
            break;
    }
    return msg_count;
}

inline bool is_little_endian()
{
    const uint16_t i = 0x01;
    return *reinterpret_cast<const uint8_t *>(&i) == 0x01;
}

inline void write_network_order(unsigned char *buf, const uint32_t value)
{
    if (is_little_endian()) {
        ZMQ_CONSTEXPR_VAR uint32_t mask = (std::numeric_limits<std::uint8_t>::max)();
        *buf++ = static_cast<unsigned char>((value » 24) & mask);
        *buf++ = static_cast<unsigned char>((value » 16) & mask);
        *buf++ = static_cast<unsigned char>((value » 8) & mask);
        *buf++ = static_cast<unsigned char>(value & mask);
    } else {
        std::memcpy(buf, &value, sizeof(value));
    }
}

inline uint32_t read_u32_network_order(const unsigned char *buf)
{
    if (is_little_endian()) {
        return (static_cast<uint32_t>(buf[0]) « 24)
               + (static_cast<uint32_t>(buf[1]) « 16)
               + (static_cast<uint32_t>(buf[2]) « 8)
               + static_cast<uint32_t>(buf[3]);
    } else {
        uint32_t value;
        std::memcpy(&value, buf, sizeof(value));
        return value;
    }
}
} // namespace detail

/*  Receive a multipart message.

    Writes the zmq::message_t objects to OutputIterator out.
    The out iterator must handle an unspecified number of writes,
    e.g. by using std::back_inserter.

    Returns: the number of messages received or nullopt (on EAGAIN).
    Throws: if recv throws. Any exceptions thrown
    by the out iterator will be propagated and the message
    may have been only partially received with pending
    message parts. It is adviced to close this socket in that event.
*/
template<class OutputIt>
ZMQ_NODISCARD recv_result_t recv_multipart(socket_ref s,
                                           OutputIt out,
                                           recv_flags flags = recv_flags::none)
{
    return detail::recv_multipart_n<false>(s, std::move(out), 0, flags);
}

/*  Receive a multipart message.

    Writes at most n zmq::message_t objects to OutputIterator out.
    If the number of message parts of the incoming message exceeds n
    then an exception will be thrown.

    Returns: the number of messages received or nullopt (on EAGAIN).
    Throws: if recv throws. Throws std::runtime_error if the number
    of message parts exceeds n (exactly n messages will have been written
    to out). Any exceptions thrown
    by the out iterator will be propagated and the message
    may have been only partially received with pending
    message parts. It is adviced to close this socket in that event.
*/
template<class OutputIt>
ZMQ_NODISCARD recv_result_t recv_multipart_n(socket_ref s,
                                             OutputIt out,
                                             size_t n,
                                             recv_flags flags = recv_flags::none)
{
    return detail::recv_multipart_n<true>(s, std::move(out), n, flags);
}

/*  Send a multipart message.

    The range must be a ForwardRange of zmq::message_t,
    zmq::const_buffer or zmq::mutable_buffer.
    The flags may be zmq::send_flags::sndmore if there are
    more message parts to be sent after the call to this function.

    Returns: the number of messages sent (exactly msgs.size()) or nullopt (on EAGAIN).
    Throws: if send throws. Any exceptions thrown
    by the msgs range will be propagated and the message
    may have been only partially sent. It is adviced to close this socket in that event.
*/
```

```
template<class Range
#ifndef ZMQ_CPP11_PARTIAL
         ,
         typename = typename std::enable_if<
           detail::is_range<Range>::value
           && (std::is_same<detail::range_value_t<Range>, message_t>::value
               || detail::is_buffer<detail::range_value_t<Range>>::value)>::type
#endif
         >
send_result_t
send_multipart(socket_ref s, Range &&msgs, send_flags flags = send_flags::none)
{
    using std::begin;
    using std::end;
    auto it = begin(msgs);
    const auto end_it = end(msgs);
    size_t msg_count = 0;
    while (it != end_it) {
        const auto next = std::next(it);
        const auto msg_flags =
          flags | (next == end_it ? send_flags::none : send_flags::sndmore);
        if (!s.send(*it, msg_flags)) {
            // zmq ensures atomic delivery of messages
            assert(it == begin(msgs));
            return {};
        }
        ++msg_count;
        it = next;
    }
    return msg_count;
}

/* Encode a multipart message.

   The range must be a ForwardRange of zmq::message_t.  A
   zmq::multipart_t or STL container may be passed for encoding.

   Returns: a zmq::message_t holding the encoded multipart data.

   Throws: std::range_error is thrown if the size of any single part
   can not fit in an unsigned 32 bit integer.

   The encoding is compatible with that used by the CZMQ function
   zmsg_encode(), see https://rfc.zeromq.org/spec/50/.
   Each part consists of a size followed by the data.
   These are placed contiguously into the output message.  A part of
   size less than 255 bytes will have a single byte size value.
   Larger parts will have a five byte size value with the first byte
   set to 0xFF and the remaining four bytes holding the size of the
   part's data.
*/
template<class Range
#ifndef ZMQ_CPP11_PARTIAL
         ,
         typename = typename std::enable_if<
           detail::is_range<Range>::value
           && (std::is_same<detail::range_value_t<Range>, message_t>::value
               || detail::is_buffer<detail::range_value_t<Range>>::value)>::type
#endif
         >
message_t encode(const Range &parts)
{
    size_t mmsg_size = 0;

    // First pass check sizes
    for (const auto &part : parts) {
        const size_t part_size = part.size();
        if (part_size > (std::numeric_limits<std::uint32_t>::max)()) {
            // Size value must fit into uint32_t.
            throw std::range_error("Invalid size, message part too large");
        }
        const size_t count_size =
          part_size < (std::numeric_limits<std::uint8_t>::max)() ? 1 : 5;
        mmsg_size += part_size + count_size;
    }

    message_t encoded(mmsg_size);
    unsigned char *buf = encoded.data<unsigned char>();
    for (const auto &part : parts) {
        const uint32_t part_size = static_cast<uint32_t>(part.size());
        const unsigned char *part_data =
          static_cast<const unsigned char *>(part.data());

        if (part_size < (std::numeric_limits<std::uint8_t>::max)()) {
            // small part
            *buf++ = (unsigned char) part_size;
        } else {
            // big part
            *buf++ = (std::numeric_limits<uint8_t>::max)();
            detail::write_network_order(buf, part_size);
            buf += sizeof(part_size);
        }
        std::memcpy(buf, part_data, part_size);
        buf += part_size;
    }

    assert(static_cast<size_t>(buf - encoded.data<unsigned char>()) == mmsg_size);
    return encoded;
}

/*  Decode an encoded message to multiple parts.

    The given output iterator must be a ForwardIterator to a container
    holding zmq::message_t such as a zmq::multipart_t or various STL
    containers.

    Returns the ForwardIterator advanced once past the last decoded
    part.

    Throws: a std::out_of_range is thrown if the encoded part sizes
    lead to exceeding the message data bounds.

    The decoding assumes the message is encoded in the manner
    performed by zmq::encode(), see https://rfc.zeromq.org/spec/50/.
 */
template<class OutputIt> OutputIt decode(const message_t &encoded, OutputIt out)
{
    const unsigned char *source = encoded.data<unsigned char>();
    const unsigned char *const limit = source + encoded.size();

    while (source < limit) {
        size_t part_size = *source++;
        if (part_size == (std::numeric_limits<std::uint8_t>::max)()) {
            if (static_cast<size_t>(limit - source) < sizeof(uint32_t)) {
                throw std::out_of_range(
                  "Malformed encoding, overflow in reading size");
            }
            part_size = detail::read_u32_network_order(source);
            // the part size is allowed to be less than 0xFF
            source += sizeof(uint32_t);
        }

        if (static_cast<size_t>(limit - source) < part_size) {
            throw std::out_of_range("Malformed encoding, overflow in reading part");
        }
        *out = message_t(source, part_size);
        ++out;
        source += part_size;
    }

    assert(source == limit);
    return out;
}

#endif


#ifdef ZMQ_HAS_RVALUE_REFS

/*
    This class handles multipart messaging. It is the C++ equivalent of zmsg.h,
    which is part of CZMQ (the high-level C binding). Furthermore, it is a major
    improvement compared to zmsg.hpp, which is part of the examples in the ØMQ
    Guide. Unnecessary copying is avoided by using move semantics to efficiently
    add/remove parts.
*/
class multipart_t
{
  private:
    std::deque<message_t> m_parts;

  public:
    typedef std::deque<message_t>::value_type value_type;

    typedef std::deque<message_t>::iterator iterator;
    typedef std::deque<message_t>::const_iterator const_iterator;

    typedef std::deque<message_t>::reverse_iterator reverse_iterator;
    typedef std::deque<message_t>::const_reverse_iterator const_reverse_iterator;

    // Default constructor
    multipart_t() {}

    // Construct from socket receive
    multipart_t(socket_ref socket) { recv(socket); }

    // Construct from memory block
    multipart_t(const void *src, size_t size) { addmem(src, size); }

    // Construct from string
    multipart_t(const std::string &string) { addstr(string); }

    // Construct from message part
    multipart_t(message_t &&message) { add(std::move(message)); }

    // Move constructor
    multipart_t(multipart_t &&other) ZMQ_NOTHROW { m_parts = std::move(other.m_parts); }

    // Move assignment operator
    multipart_t &operator=(multipart_t &&other) ZMQ_NOTHROW
    {
        m_parts = std::move(other.m_parts);
        return *this;
    }

    // Destructor
    virtual ~multipart_t() { clear(); }

    message_t &operator[](size_t n) { return m_parts[n]; }

    const message_t &operator[](size_t n) const { return m_parts[n]; }

    message_t &at(size_t n) { return m_parts.at(n); }

    const message_t &at(size_t n) const { return m_parts.at(n); }

    iterator begin() { return m_parts.begin(); }

    const_iterator begin() const { return m_parts.begin(); }

    const_iterator cbegin() const { return m_parts.cbegin(); }

    reverse_iterator rbegin() { return m_parts.rbegin(); }

    const_reverse_iterator rbegin() const { return m_parts.rbegin(); }

    iterator end() { return m_parts.end(); }

    const_iterator end() const { return m_parts.end(); }

    const_iterator cend() const { return m_parts.cend(); }

    reverse_iterator rend() { return m_parts.rend(); }

    const_reverse_iterator rend() const { return m_parts.rend(); }

    // Delete all parts
    void clear() { m_parts.clear(); }

    // Get number of parts
    size_t size() const { return m_parts.size(); }

    // Check if number of parts is zero
    bool empty() const { return m_parts.empty(); }

    // Receive multipart message from socket
    bool recv(socket_ref socket, int flags = 0)
    {
        clear();
        bool more = true;
        while (more) {
            message_t message;
#ifdef ZMQ_CPP11
            if (!socket.recv(message, static_cast<recv_flags>(flags)))
                return false;
#else
            if (!socket.recv(&message, flags))
                return false;
#endif
            more = message.more();
            add(std::move(message));
        }
        return true;
    }

    // Send multipart message to socket
    bool send(socket_ref socket, int flags = 0)
    {
        flags &= ~(ZMQ_SNDMORE);
        bool more = size() > 0;
        while (more) {
            message_t message = pop();
            more = size() > 0;
#ifdef ZMQ_CPP11
            if (!socket.send(message, static_cast<send_flags>(
                                         (more ? ZMQ_SNDMORE : 0) | flags)))
                return false;
#else
            if (!socket.send(message, (more ? ZMQ_SNDMORE : 0) | flags))
                return false;
#endif
        }
        clear();
        return true;
    }

    // Concatenate other multipart to front
    void prepend(multipart_t &&other)
    {
        while (!other.empty())
            push(other.remove());
    }

    // Concatenate other multipart to back
    void append(multipart_t &&other)
    {
        while (!other.empty())
            add(other.pop());
    }

    // Push memory block to front
    void pushmem(const void *src, size_t size)
    {
        m_parts.push_front(message_t(src, size));
    }

    // Push memory block to back
    void addmem(const void *src, size_t size)
    {
        m_parts.push_back(message_t(src, size));
    }

    // Push string to front
    void pushstr(const std::string &string)
    {
        m_parts.push_front(message_t(string.data(), string.size()));
    }

    // Push string to back
    void addstr(const std::string &string)
    {
        m_parts.push_back(message_t(string.data(), string.size()));
    }

    // Push type (fixed-size) to front
    template<typename T> void pushtyp(const T &type)
    {
        static_assert(!std::is_same<T, std::string>::value,
                      "Use pushstr() instead of pushtyp<std::string>()");
        m_parts.push_front(message_t(&type, sizeof(type)));
    }

    // Push type (fixed-size) to back
    template<typename T> void addtyp(const T &type)
    {
        static_assert(!std::is_same<T, std::string>::value,
                      "Use addstr() instead of addtyp<std::string>()");
        m_parts.push_back(message_t(&type, sizeof(type)));
    }

    // Push message part to front
    void push(message_t &&message) { m_parts.push_front(std::move(message)); }

    // Push message part to back
    void add(message_t &&message) { m_parts.push_back(std::move(message)); }

    // Alias to allow std::back_inserter()
    void push_back(message_t &&message) { m_parts.push_back(std::move(message)); }

    // Pop string from front
    std::string popstr()
    {
        std::string string(m_parts.front().data<char>(), m_parts.front().size());
        m_parts.pop_front();
        return string;
    }

    // Pop type (fixed-size) from front
    template<typename T> T poptyp()
    {
        static_assert(!std::is_same<T, std::string>::value,
                      "Use popstr() instead of poptyp<std::string>()");
        if (sizeof(T) != m_parts.front().size())
            throw std::runtime_error(
              "Invalid type, size does not match the message size");
        T type = *m_parts.front().data<T>();
        m_parts.pop_front();
        return type;
    }

    // Pop message part from front
    message_t pop()
    {
        message_t message = std::move(m_parts.front());
        m_parts.pop_front();
        return message;
    }

    // Pop message part from back
    message_t remove()
    {
        message_t message = std::move(m_parts.back());
        m_parts.pop_back();
        return message;
    }

    // get message part from front
    const message_t &front() { return m_parts.front(); }

    // get message part from back
    const message_t &back() { return m_parts.back(); }

    // Get pointer to a specific message part
    const message_t *peek(size_t index) const { return &m_parts[index]; }

    // Get a string copy of a specific message part
    std::string peekstr(size_t index) const
    {
        std::string string(m_parts[index].data<char>(), m_parts[index].size());
        return string;
    }

    // Peek type (fixed-size) from front
    template<typename T> T peektyp(size_t index) const
    {
        static_assert(!std::is_same<T, std::string>::value,
                      "Use peekstr() instead of peektyp<std::string>()");
        if (sizeof(T) != m_parts[index].size())
            throw std::runtime_error(
              "Invalid type, size does not match the message size");
        T type = *m_parts[index].data<T>();
        return type;
    }

    // Create multipart from type (fixed-size)
    template<typename T> static multipart_t create(const T &type)
    {
        multipart_t multipart;
        multipart.addtyp(type);
        return multipart;
    }

    // Copy multipart
    multipart_t clone() const
    {
        multipart_t multipart;
        for (size_t i = 0; i < size(); i++)
            multipart.addmem(m_parts[i].data(), m_parts[i].size());
        return multipart;
    }

    // Dump content to string
    std::string str() const
    {
        std::stringstream ss;
        for (size_t i = 0; i < m_parts.size(); i++) {
            const unsigned char *data = m_parts[i].data<unsigned char>();
            size_t size = m_parts[i].size();

            // Dump the message as text or binary
            bool isText = true;
            for (size_t j = 0; j < size; j++) {
                if (data[j] < 32 || data[j] > 127) {
                    isText = false;
                    break;
                }
            }
            ss « "\n[" « std::dec « std::setw(3) « std::setfill('0') « size
               « "] ";
            if (size >= 1000) {
                ss « "... (too big to print)";
                continue;
            }
            for (size_t j = 0; j < size; j++) {
                if (isText)
                    ss « static_cast<char>(data[j]);
                else
                    ss « std::hex « std::setw(2) « std::setfill('0')
                       « static_cast<short>(data[j]);
            }
        }
        return ss.str();
    }

    // Check if equal to other multipart
    bool equal(const multipart_t *other) const ZMQ_NOTHROW
    {
        return *this == *other;
    }

    bool operator==(const multipart_t &other) const ZMQ_NOTHROW
    {
        if (size() != other.size())
            return false;
        for (size_t i = 0; i < size(); i++)
            if (at(i) != other.at(i))
                return false;
        return true;
    }

    bool operator!=(const multipart_t &other) const ZMQ_NOTHROW
    {
        return !(*this == other);
    }

#ifdef ZMQ_CPP11

    // Return single part message_t encoded from this multipart_t.
    message_t encode() const { return zmq::encode(*this); }

    // Decode encoded message into multiple parts and append to self.
    void decode_append(const message_t &encoded)
    {
        zmq::decode(encoded, std::back_inserter(*this));
    }

    // Return a new multipart_t containing the decoded message_t.
    static multipart_t decode(const message_t &encoded)
    {
        multipart_t tmp;
        zmq::decode(encoded, std::back_inserter(tmp));
        return tmp;
    }

#endif

  private:
    // Disable implicit copying (moving is more efficient)
    multipart_t(const multipart_t &other) ZMQ_DELETED_FUNCTION;
    void operator=(const multipart_t &other) ZMQ_DELETED_FUNCTION;
}; // class multipart_t

inline std::ostream &operator«(std::ostream &os, const multipart_t &msg)
{
    return os « msg.str();
}

#endif // ZMQ_HAS_RVALUE_REFS

#if defined(ZMQ_BUILD_DRAFT_API) && defined(ZMQ_CPP11) && defined(ZMQ_HAVE_POLLER)
class active_poller_t
{
  public:
    active_poller_t() = default;
    ~active_poller_t() = default;

    active_poller_t(const active_poller_t &) = delete;
    active_poller_t &operator=(const active_poller_t &) = delete;

    active_poller_t(active_poller_t &&src) = default;
    active_poller_t &operator=(active_poller_t &&src) = default;

    using handler_type = std::function<void(event_flags)>;

    void add(zmq::socket_ref socket, event_flags events, handler_type handler)
    {
        if (!handler)
            throw std::invalid_argument("null handler in active_poller_t::add");
        auto ret = handlers.emplace(
          socket, std::make_shared<handler_type>(std::move(handler)));
        if (!ret.second)
            throw error_t(EINVAL); // already added
        try {
            base_poller.add(socket, events, ret.first->second.get());
            need_rebuild = true;
        }
        catch (...) {
            // rollback
            handlers.erase(socket);
            throw;
        }
    }

    void remove(zmq::socket_ref socket)
    {
        base_poller.remove(socket);
        handlers.erase(socket);
        need_rebuild = true;
    }

    void modify(zmq::socket_ref socket, event_flags events)
    {
        base_poller.modify(socket, events);
    }

    size_t wait(std::chrono::milliseconds timeout)
    {
        if (need_rebuild) {
            poller_events.resize(handlers.size());
            poller_handlers.clear();
            poller_handlers.reserve(handlers.size());
            for (const auto &handler : handlers) {
                poller_handlers.push_back(handler.second);
            }
            need_rebuild = false;
        }
        const auto count = base_poller.wait_all(poller_events, timeout);
        std::for_each(poller_events.begin(),
                      poller_events.begin() + static_cast<ptrdiff_t>(count),
                      [](decltype(base_poller)::event_type &event) {
                          assert(event.user_data != nullptr);
                          (*event.user_data)(event.events);
                      });
        return count;
    }

    ZMQ_NODISCARD bool empty() const noexcept { return handlers.empty(); }

    size_t size() const noexcept { return handlers.size(); }

  private:
    bool need_rebuild{false};

    poller_t<handler_type> base_poller{};
    std::unordered_map<socket_ref, std::shared_ptr<handler_type» handlers{};
    std::vector<decltype(base_poller)::event_type> poller_events{};
    std::vector<std::shared_ptr<handler_type» poller_handlers{};
};     // class active_poller_t
#endif //  defined(ZMQ_BUILD_DRAFT_API) && defined(ZMQ_CPP11) && defined(ZMQ_HAVE_POLLER)


} // namespace zmq

#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.

```
/*****************************************************************************************************************************
 *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.                                  *
 *                                                                                                                              *
 *   Copyright (C) 2023 Degoras Project Team                                                                                    *
 *                       < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >                                     *
 *                       < Jesús Relinque Madroñal >                                                                            *
 *                                                                                                                              *
 *   This file is part of LibZMQUtils.                                                                                          *
 *                                                                                                                              *
 *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of the EUPL license   *
 *   as soon they will be approved by the European Commission (IDABC).                                                          *
 *                                                                                                                              *
 *   This project is free software: you can redistribute it and/or modify it under the terms of the EUPL license as    *
 *   published by the IDABC, either Version 1.2 or, at your option, any later version.                                          *
 *                                                                                                                              *
 *   This project is distributed in the hope that it will be useful. Unless required by applicable law or agreed to in *
 *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND; without even the  *
 *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to check specific   *
 *   language governing permissions and limitations and more details.                                                          *
 *                                                                                                                              *
 *   You should use this project in compliance with the EUPL license. You should have received a copy of the license   *
 *   along with this project. If not, see the license at < https://eupl.eu/ >.
*

*************************************************************************************************************************/

//
======================================================================================================================
#pragma once
//
======================================================================================================================

// C++ INCLUDES
//
======================================================================================================================
#include <future>
#include <map>
//
======================================================================================================================

// ZMQUTILS INCLUDES
//
======================================================================================================================
#include "LibZMQUtils/libzmqutils_global.h"
#include "LibZMQUtils/CommandServerClient/common.h"
//
======================================================================================================================

// ZMQ DECLARATIONS
//
======================================================================================================================
namespace zmq
{
    class context_t;
    class socket_t;
}
//
======================================================================================================================

// ZMQUTILS NAMESPACES
//
======================================================================================================================
namespace zmqutils{
//
======================================================================================================================

//
======================================================================================================================
using common::ServerCommand;
using common::ServerResult;
using common::ClientResult;
using common::CommandReply;
using common::CommandType;
using common::RequestData;
//
======================================================================================================================

class LIBZMQUTILS_EXPORT CommandClientBase
{

public:

    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:

    virtual void onSendCommand(const RequestData&, const zmq::multipart_t&) = 0;

private:


    ClientResult recvFromSocket(CommandReply&);

    void sendAliveCallback();
    zmq::multipart_t prepareMessage(const RequestData &msg);

    // Internal client identification.
    common::HostClient client_info_;

    // Server endpoint.
    std::string server_endpoint_;

    // ZMQ context and socket.
    zmq::context_t *context_;
    zmq::socket_t *client_socket_;

    // Mutex.
    std::mutex mtx_;

    std::future<void> auto_alive_future_;
    std::condition_variable auto_alive_cv_;
    std::atomic_bool auto_alive_working_;

};

} // END NAMESPACES.
//
=====================================================================================================
```

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.

```
/****************************************************************************************************************
 *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities. *
 * *
 *   Copyright (C) 2023 Degoras Project Team *
 *                        < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com > *
 *                        < Jesús Relinque Madroñal > *
 * *
 *   This file is part of LibZMQUtils. *
 * *
 *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of the EUPL license   *
 *   as soon they will be approved by the European Commission (IDABC). *
 * *
 *   This project is free software: you can redistribute it and/or modify it under the terms of the EUPL license as    *
 *   published by the IDABC, either Version 1.2 or, at your option, any later version. *
 * *
 *   This project is distributed in the hope that it will be useful. Unless required by applicable law or agreed to in *
 *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND; without even the  *
 *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to check specific   *
 *   language governing permissions and limitations and more details. *
 * *
 *   You should use this project in compliance with the EUPL license. You should have received a copy of the license   *
 *   along with this project. If not, see the license at < https://eupl.eu/ >. *
 ***************************************************************************************************************/

// =====================================================================================================================
#pragma once
// =====================================================================================================================

// C++ INCLUDES
// =====================================================================================================================
#include <future>
#include <map>
#include <zmq/zmq.hpp>
#include <zmq/zmq_addon.hpp>
// =====================================================================================================================

// ZMQUTILS INCLUDES
// =====================================================================================================================
#include "LibZMQUtils/libzmqutils_global.h"
#include "LibZMQUtils/CommandServerClient/common.h"
#include "LibZMQUtils/utils.h"
// =====================================================================================================================

// ZMQ DECLARATIONS
// =====================================================================================================================
namespace zmq
{
    class context_t;
    class socket_t;
}
// =====================================================================================================================

// ZMQUTILS NAMESPACES
//
=====================================================================================================
namespace zmqutils{
//
=====================================================================================================

//
=====================================================================================================
using common::ServerResultStr;
using common::CommandReply;
using common::CommandRequest;
using common::ServerCommand;
using common::ServerResult;
using common::HostClient;
using utils::NetworkAdapterInfo;
//
=====================================================================================================

class LIBZMQUTILS_EXPORT CommandServerBase
{

public:

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

    const unsigned& getServerPort() const;

    const std::vector<NetworkAdapterInfo> &getServerAddresses() const;

    const std::string& getServerEndpoint() const;

    const std::future<void>& getServerWorkerFuture() const;

    const std::map<std::string, HostClient>& getConnectedClients() const;

    bool isWorking() const{return this->server_working_;}

    void setClientStatusCheck(bool);

    void startServer();

    void stopServer();

    virtual ~CommandServerBase();

protected:

    virtual void onServerStop() = 0;

    virtual void onServerStart() = 0;

    virtual void onWaitingCommand() = 0;

    virtual void onConnected(const HostClient&) = 0;

    virtual void onDisconnected(const HostClient&) = 0;

    virtual void onDeadClient(const HostClient&) = 0;

    virtual void onInvalidMsgReceived(const CommandRequest&) = 0;

    virtual void onCommandReceived(const CommandRequest&) = 0;

    virtual void onCustomCommandReceived(const CommandRequest&, CommandReply&);

    virtual void onServerError(const zmq::error_t &error, const std::string& ext_info = "") = 0;

    virtual void onSendingResponse(const CommandReply&) = 0;

private:

    // Helper for prepare the result message.
    static void prepareCommandResult(ServerResult, std::unique_ptr<uint8_t>& data_out);

    // Helper for check if the base command is valid.
    static bool validateCommand(int raw_command);

    // Server worker. Will be execute asynchronously.
    void serverWorker();

    // Process command class.
    void processCommand(const CommandRequest&, CommandReply&);

    // Client status checker.
    void checkClientsAliveStatus();

    // Update client last connection.
    void updateClientLastConnection(const std::string& id);

    // Update the server timeout.
    void updateServerTimeout();

    // Internal connect execution process.
    ServerResult execReqConnect(const CommandRequest&);

    // Internal disconnect execution process.
    ServerResult execReqDisconnect(const CommandRequest&);

    // Function for receive data from the client.
    ServerResult recvFromSocket(CommandRequest&);

    // Function for reset the socket.
    void resetSocket();

    // ZMQ socket and context.
    zmq::context_t *context_;
    zmq::socket_t* server_socket_;

    // Endpoint data.
    std::string server_endpoint_;
    std::vector<utils::NetworkAdapterInfo> server_listen_adapters_;
    unsigned server_port_;

    // Mutex.
    std::mutex mtx_;

    // Future for the server worker.
    std::future<void> server_worker_future_;

    // Clients container.
    std::map<std::string, HostClient> connected_clients_;

    // Usefull flags.
    std::atomic_bool server_working_;
    std::atomic_bool check_clients_alive_;
};

} // END NAMESPACES.
//
=====================================================================================================================
```

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.

```
/******************************************************************************************************************
 *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
*
 *
*
 *   Copyright (C) 2023 Degoras Project Team
*
 *                       < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
*
 *                       < Jesús Relinque Madroñal >
*
 *
*
 *   This file is part of LibZMQUtils.
*
```

```
 *                                                                                                        *
 *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of the EUPL license   *
 *   as soon they will be approved by the European Commission (IDABC).                                    *
 *                                                                                                        *
 *   This project is free software: you can redistribute it and/or modify it under the terms of the EUPL license as    *
 *   published by the IDABC, either Version 1.2 or, at your option, any later version.                    *
 *                                                                                                        *
 *   This project is distributed in the hope that it will be useful. Unless required by applicable law or agreed to in *
 *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND; without even the  *
 *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to check specific   *
 *   language governing permissions and limitations and more details.                                     *
 *                                                                                                        *
 *   You should use this project in compliance with the EUPL license. You should have received a copy of the license   *
 *   along with this project. If not, see the license at < https://eupl.eu/ >.                            *
 **********************************************************************************************************************/

// =====================================================================================================================
#pragma once
// =====================================================================================================================

// =====================================================================================================================
#if ((defined __WIN32__) || (defined _WIN32)) && (!defined LIBZMQUTILS_STATIC)
#ifdef LIBZMQUTILS_LIBRARY
#define LIBZMQUTILS_EXPORT  __declspec(dllexport)
#else
#define LIBZMQUTILS_EXPORT  __declspec(dllimport)
#endif
#else
/* Static libraries or non-Windows needs no special declaration. */
# define LIBZMQUTILS_EXPORT
#endif
// =====================================================================================================================
```

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.

```

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

```

#include <iostream>

#include <iostream>
#include <string>
#include <cstring>
#include <cstdlib>
#include <algorithm>

#include "LibZMQUtils/CommandServerClient/command_client.h"
#include "LibZMQUtils/utils.h"

// ZMQUTILS NAMESPACES
// ==========================================================================================================================
namespace zmqutils{
// ==========================================================================================================================




CommandClientBase::CommandClientBase(const std::string &server_endpoint) :
    server_endpoint_(server_endpoint),
    context_(nullptr),
    client_socket_(nullptr),
    auto_alive_working_(false)
{

}

CommandClientBase::~CommandClientBase()
{
    if (this->auto_alive_working_)
        this->stopAutoAlive();
    this->stopClient();
}

bool CommandClientBase::startClient(const std::string& interface_name)
{
    // Auxiliar variables.
    std::string ip, name, pid;

    // Get the client ip.
    std::vector<utils::NetworkAdapterInfo> interfcs = utils::getHostIPsWithInterfaces();
    auto it = std::find_if(interfcs.begin(), interfcs.end(), [&interface_name](const utils::NetworkAdapterInfo& info)
                           {return info.name == interface_name;});
    if (it == interfcs.end())
        return false;
    ip = it->ip;

    // Get the host name.
    name = utils::getHostname();

    // Get the current pid.
    pid = std::to_string(utils::getCurrentPID());

    // Store the info.
    this->client_info_ = common::HostClient(ip, name, pid);

    std::cout«client_info_.id«std::endl;

    // If server is already started, do nothing
    if (this->client_socket_)
        return false;

    // Create the ZMQ context.
    if (!this->context_)
        this->context_ = new zmq::context_t(1);

    try
    {
        this->client_socket_ = new zmq::socket_t(*this->context_, zmq::socket_type::req);
        this->client_socket_->connect(this->server_endpoint_);
        // Set timeout so socket will not wait for answer more than client alive timeout.
        this->client_socket_->set(zmq::sockopt::rcvtimeo, common::kDefaultServerAliveTimeoutMsec);
        this->client_socket_->set(zmq::sockopt::linger, 0);
    }
    catch (const zmq::error_t &error)
    {
        delete this->client_socket_;
        this->client_socket_ = nullptr;

        std::cerr « "Error at socket creation: " « error.num();
        // TODO: handle error
        return false;
    }

    // All ok.
    return true;
}

void CommandClientBase::stopClient()
{
    // If server is already stopped, do nothing.
    if (!this->client_socket_)
        return;


    // Destroy the  socket.
    delete this->client_socket_;
    this->client_socket_ = nullptr;

    std::this_thread::sleep_for(std::chrono::milliseconds(1050));

    // Delete context

    if (this->context_)
    {
        delete this->context_;
        this->context_ = nullptr;
    }
}

void CommandClientBase::resetClient()
{
    if (this->client_socket_)
    {
        // Destroy the socket and create again to flush.
        delete this->client_socket_;

        std::this_thread::sleep_for(std::chrono::milliseconds(1050));

        try
        {
            this->client_socket_ = new zmq::socket_t(*this->context_, zmq::socket_type::req);
            this->client_socket_->connect(this->server_endpoint_);
            // Set timeout so socket will not wait for answer more than server alive timeout.
            this->client_socket_->set(zmq::sockopt::rcvtimeo, common::kDefaultServerAliveTimeoutMsec);
            this->client_socket_->set(zmq::sockopt::linger, 0);
        }
        catch (const zmq::error_t &error)
        {
            delete this->client_socket_;
            this->client_socket_ = nullptr;

            std::cerr « "Error at socket creation: " « error.num();
            // TODO: handle error
        }
    }
}

void CommandClientBase::startAutoAlive()
{
    this->auto_alive_working_ = true;
    this->auto_alive_future_ = std::async(std::launch::async, [this]{this->sendAliveCallback();});
}

void CommandClientBase::stopAutoAlive()
{
    if (this->auto_alive_working_)
    {
        this->auto_alive_working_ = false;
        this->auto_alive_cv_.notify_all();
        this->auto_alive_future_.wait();
    }
}

void CommandClientBase::setClientHostIP(const std::string&){}

void CommandClientBase::setClientId(const std::string &){}

ClientResult CommandClientBase::sendCommand(const RequestData& msg, CommandReply& reply)
{
    // Result.
    ClientResult result;

    // Check if we start the client.
    if (!this->client_socket_)
        return ClientResult::CLIENT_STOPPED;

    // Send the command.
    try
    {

        zmq::multipart_t multipart_msg(this->prepareMessage(msg));

        // Internal send callback.
        this->onSendCommand(msg, multipart_msg);

        // Send the multiple messages.
        multipart_msg.send(*this->client_socket_);

    }  catch (const zmq::error_t &error)
    {
        // TODO: handle error
        std::cout«error.what()«std::endl;
        return ClientResult::INTERNAL_ZMQ_ERROR;
    }

    std::cout«"Waiting response"«std::endl;


    // TODO multipart.

    result = this->recvFromSocket(reply);





    if (this->auto_alive_working_)
        this->auto_alive_cv_.notify_one();

    return result;

}

ClientResult CommandClientBase::recvFromSocket(CommandReply& reply)
{
    // Result variable.
    ClientResult result = ClientResult::COMMAND_OK;

    // Containers.
    bool recv_result;
    zmq::multipart_t multipart_msg;

    // Try to receive data. If an execption is thrown, receiving fails and an error code is generated.
    try
    {
        // Call to the internal waiting command callback. TODO
        //this->onWaitingCommand();

        // Wait the command.
        recv_result = multipart_msg.recv(*(this->client_socket_));

        // Store the raw data.
        reply.raw_msg = multipart_msg.clone();
    }
    catch(zmq::error_t& error)
    {
        // Call to error callback. TODO
        //this->onServerError(error, "Error while receiving a request.");

        std::cout«"INTERNAL ERRROR "«error.what()«std::endl;

        return ClientResult::INTERNAL_ZMQ_ERROR;
    }

    // Check for empty msg or timeout reached.
    if (multipart_msg.empty() && !recv_result)
        return ClientResult::TIMEOUT_REACHED;
    else if (multipart_msg.empty())
        return ClientResult::EMPTY_MSG;

    // Check the multipart msg size.
    if (multipart_msg.size() == 1 || multipart_msg.size() == 2)
    {
        // Get the multipart data.
        zmq::message_t message_result = multipart_msg.pop();

        // Get the sizes.
        size_t result_size_bytes = message_result.size();

        // Get the command.
        if (result_size_bytes == sizeof(ServerCommand))
        {
            int raw_result;
            utils::binarySerializeDeserialize(message_result.data(), sizeof(ServerCommand),
&raw_result);
```

```
            reply.result = static_cast<common::ServerResult>(raw_result);
        }
        else
            return ClientResult::INVALID_MSG;

        // If there is still one more part, they are the parameters.
        if (multipart_msg.size() == 1)
        {
            // Get the message and the size.
            zmq::message_t message_params = multipart_msg.pop();
            size_t params_size_bytes = message_params.size();

            std::cout«multipart_msg.str()«std::endl;
            std::cout«params_size_bytes«std::endl;

            // Check the parameters.
            if(params_size_bytes > 0)
            {
                // Get and store the parameters data.
                std::unique_ptr<std::uint8_t> params =
                    std::unique_ptr<std::uint8_t>(new std::uint8_t[params_size_bytes]);
                auto *params_pointer = static_cast<std::uint8_t*>(message_params.data());
                std::copy(params_pointer, params_pointer + params_size_bytes, params.get());
                reply.params = std::move(params);
                reply.params_size = params_size_bytes;
            }
            else
                return ClientResult::EMPTY_PARAMS;
        }
    }
    else
        return ClientResult::INVALID_PARTS;

    // Return the result.
    return result;
}

void CommandClientBase::sendAliveCallback()
{
    /*
    std::mutex m;
    std::unique_lock<std::mutex> lk(m);
    bool send_success = true;
    bool recv_success = true;
    void *data_out;
    size_t out_size;
    zmq::multipart_t msg;
    zmq::socket_t *alive_socket = new zmq::socket_t(*this->context_, zmq::socket_type::req);
    alive_socket->connect(this->server_endpoint_);
    // Set timeout so socket will not wait for answer more than client alive timeout.
    alive_socket->set(zmq::sockopt::rcvtimeo, common::kDefaultServerAliveTimeoutMsec);
    alive_socket->set(zmq::sockopt::linger, 0);

    while(this->auto_alive_working_)
    {
        auto res =
            this->auto_alive_cv_.wait_for(lk,
std::chrono::milliseconds(common::kClientAlivePeriodMsec));

        if (std::cv_status::timeout == res)
        {
            msg = this->prepareMessage(
                            RequestData(static_cast<common::CommandType>(ServerCommand::REQ_ALIVE)));
            try
            {
                msg.send(*alive_socket);
            }  catch (const zmq::error_t &error)
            {
                // TODO: handle error
                std::cerr « "Failed to automatically send alive command with error: " « error.num() «
std::endl;
                send_success = false;
            }

            if (send_success)
            {
                auto recv_result = this->recvFromSocket(alive_socket, data_out, out_size);
                auto *data_bytes = static_cast<std::uint8_t*>(data_out);

                if (0 == recv_result && out_size == sizeof(common::ServerResult))
                {
                    common::ServerResult result;


                    zmqutils::utils::binarySerializeDeserialize(
                                data_bytes, sizeof(common::CommandReply), &result);

                    recv_success = result == common::ServerResult::COMMAND_OK;

                }
                else
                {
                    std::cerr « "Auto alive message answer receive failed" « std::endl;
                    recv_success = false;
                }

                delete[] data_bytes;
            }

            if (!send_success || !recv_success)
            {
                std::cerr « "Failed auto sending alive message. Process will be stopped." « std::endl;
                this->auto_alive_working_ = false;
            }

        }

    }

    delete alive_socket;
    */
}

zmq::multipart_t CommandClientBase::prepareMessage(const RequestData &msg)
{
    // Prepare the ip data.
    zmq::message_t message_ip(this->client_info_.ip.begin(), this->client_info_.ip.end());
    // Prepare the hostname data.
    zmq::message_t message_host(this->client_info_.hostname.begin(),
this->client_info_.hostname.end());
    // Prepare the pid data.
    zmq::message_t message_pid(this->client_info_.pid.begin(), this->client_info_.pid.end());
    // Prepare the command data.
    std::uint8_t command_buffer[sizeof(common::CommandType)];
    zmqutils::utils::binarySerializeDeserialize(&msg.command, sizeof(common::CommandType),
command_buffer);
    zmq::message_t message_command(&command_buffer, sizeof(common::CommandType));


    // Prepare the multipart msg.
    zmq::multipart_t multipart_msg;
    multipart_msg.add(std::move(message_ip));
    multipart_msg.add(std::move(message_host));
    multipart_msg.add(std::move(message_pid));
    multipart_msg.add(std::move(message_command));

    // Add command parameters if they exist
    if (msg.params_size > 0)
    {
        // Prepare the command parameters
        zmq::message_t message_params(msg.params.get(), msg.params_size);
        multipart_msg.add(std::move(message_params));
    }

    return multipart_msg;
}

} // END NAMESPACES.
//
==================================================================================================
```

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.

```
/***********************************************************************************************************
 *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
 *
 *
 *
 *   Copyright (C) 2023 Degoras Project Team
 *
 *                          < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
 *
 *                          < Jesús Relinque Madroñal >
 *
 *
 *
 *   This file is part of LibZMQUtils.
 *
 *
 *
 *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
the EUPL license   *
 *   as soon they will be approved by the European Commission (IDABC).
 *
 *
 *
 *   This project is free software: you can redistribute it and/or modify it under the terms of the
EUPL license as    *
 *   published by the IDABC, either Version 1.2 or, at your option, any later version.
 *
 *
 *
 *   This project is distributed in the hope that it will be useful. Unless required by applicable law
or agreed to in *
 *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
without even the  *
 *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
check specific   *
 *   language governing permissions and limitations and more details.
 *
 *
 *
 *   You should use this project in compliance with the EUPL license. You should have received a copy
of the license   *
 *   along with this project. If not, see the license at < https://eupl.eu/ >.
 *
**********************************************************************************************************/

// C++ INCLUDES
//
=========================================================================================================
#include <iostream>
#include <stdio.h>
```

```cpp
#include <zmq/zmq_addon.hpp>
#include <zmq/zmq.h>
// =====================================================================================================

// ZMQUTILS INCLUDES
// =====================================================================================================
#include "LibZMQUtils/CommandServerClient/command_server.h"
#include "LibZMQUtils/utils.h"
// =====================================================================================================

// ZMQUTILS NAMESPACES
// =====================================================================================================
namespace zmqutils{
// =====================================================================================================

CommandServerBase::CommandServerBase(unsigned int port, const std::string& local_addr) :
    context_(nullptr),
    server_socket_(nullptr),
    server_endpoint_("tcp://" + local_addr + ":" + std::to_string(port)),
    server_port_(port),
    server_working_(false),
    check_clients_alive_(true)
{
    // Get the adapters.
    std::vector<utils::NetworkAdapterInfo> interfcs = utils::getHostIPsWithInterfaces();
    // Store the adapters.
    if(local_addr == "*")
        this->server_listen_adapters_ = interfcs;
    else
    {
        for(const auto& intrfc : interfcs)
        {
            if(intrfc.ip == local_addr)
                this->server_listen_adapters_.push_back(intrfc);
        }
    }
}

const std::future<void> &CommandServerBase::getServerWorkerFuture() const {return this->server_worker_future_;}

const std::map<std::string, HostClient> &CommandServerBase::getConnectedClients() const
{return this->connected_clients_;}

void CommandServerBase::setClientStatusCheck(bool)
{
    // Safe mutex lock
    std::unique_lock<std::mutex> lock(this->mtx_);
    // Disable the client alive checking.
    this->check_clients_alive_ = false;
    if(this->server_socket_)
        this->server_socket_->set(zmq::sockopt::rcvtimeo, -1);
}

const unsigned& CommandServerBase::getServerPort() const {return this->server_port_;}

const std::vector<utils::NetworkAdapterInfo>& CommandServerBase::getServerAddresses() const
{return this->server_listen_adapters_;}

const std::string& CommandServerBase::getServerEndpoint() const {return this->server_endpoint_;}

void CommandServerBase::startServer()
{
    // Safe mutex lock
    std::unique_lock<std::mutex> lock(this->mtx_);

    // If server is already started, do nothing
    if (this->server_working_)
        return;

    // Create the ZMQ context.
    if (!this->context_)
        this->context_ = new zmq::context_t(1);

    // Launch server worker in other thread.
    this->server_worker_future_ = std::async(std::launch::async, &CommandServerBase::serverWorker, this);
}

void CommandServerBase::stopServer()
{
    // Safe mutex lock
    std::unique_lock<std::mutex> lock(this->mtx_);

    // If server is already stopped, do nothing.
    if (!this->server_working_)
        return;

    // Set the shared working flag to false (is atomic).
    this->server_working_ = false;

    // Delete the context.
    if (this->context_)
    {
        delete this->context_;
        context_ = nullptr;
    }

    // Clean the clients.
    this->connected_clients_.clear();
}

CommandServerBase::~CommandServerBase()
{
    // Stop the server (this function also deletes the pointers).
    this->stopServer();
}

ServerResult CommandServerBase::execReqConnect(const CommandRequest& cmd_req)
{
    // Safe mutex lock.
    std::unique_lock<std::mutex> lock(this->mtx_);

    // Check if the client is already connected.
    auto it = this->connected_clients_.find(cmd_req.client.id);
    if(it != this->connected_clients_.end())
        return ServerResult::ALREADY_CONNECTED;

    // Add the new client.
    this->connected_clients_[cmd_req.client.id] = cmd_req.client;

    // Update the timeout of the main socket.
    if(this->check_clients_alive_)
        this->updateServerTimeout();

    // Call to the internal callback.
    this->onConnected(cmd_req.client);

    // All ok.
    return ServerResult::COMMAND_OK;
}

ServerResult CommandServerBase::execReqDisconnect(const CommandRequest& cmd_req)
{
    // Safe mutex lock.
    std::unique_lock<std::mutex> lock(this->mtx_);

    // Get the client.
    auto it = this->connected_clients_.find(cmd_req.client.id);

    // Remove the client from the map of connected clients.
    this->connected_clients_.erase(it);

    // Call to the internal callback.
    this->onDisconnected(cmd_req.client);

    // Update the timeout of the main socket.
    if(this->check_clients_alive_)
        this->updateServerTimeout();

    // All ok.
    return ServerResult::COMMAND_OK;
}

void CommandServerBase::serverWorker()
{
    // Auxiliar variables.
    ServerResult result;

    // Set the working flag to true.
    this->server_working_ = true;

    // Start server socket
    this->resetSocket();

    // Server worker loop.
    // If there is no client connected wait for a client to connect or for an exit message. If there
    // is a client connected set timeout, so if no command comes in time, check the last time
connection
```

```
    // for each client. The loop can be stopped (in a safe way) if using the stopServer() function.
    while(this->server_socket_ && this->server_working_)
    {
        // Message container.
        CommandRequest cmd_request;

        // Result container.
        CommandReply cmd_reply;

        // Receive the data.
        result = this->recvFromSocket(cmd_request);

        // Check all the clients status.
        if(this->check_clients_alive_)
            this->checkClientsAliveStatus();

        // Process the data.
        if(result == ServerResult::COMMAND_OK && !this->server_working_)
        {
            // In this case, we will close the server. Call to the internal callback.
            this->onServerStop();
        }
        else if(result == ServerResult::TIMEOUT_REACHED)
        {
            // DO NOTHING.
        }
        else if (result != ServerResult::COMMAND_OK)
        {
            // Internal callback.
            this->onInvalidMsgReceived(cmd_request);

            // Prepare the message.
            std::uint8_t res_buff[sizeof(ServerResult)];
            utils::binarySerializeDeserialize(&result, sizeof(ServerResult), res_buff);
            zmq::message_t message_res(res_buff, sizeof(ServerResult));

            // Send response callback.
            cmd_reply.result = result;
            this->onSendingResponse(cmd_reply);

            // Send the response.
            try
            {
                this->server_socket_->send(message_res, zmq::send_flags::none);
            }
            catch (const zmq::error_t &error)
            {
                // Check if we want to close the server.
                // The error code is for ZMQ EFSM error.
                if(!(error.num() == common::kZmqEFSMError && !this->server_working_))
                    this->onServerError(error, "Error while sending a response.");
            }
        }
        else if (result == ServerResult::COMMAND_OK)
        {
            // Reply id buffer.
            std::unique_ptr<std::uint8_t> rep_id_buff;

            // Execute the command.
            this->processCommand(cmd_request, cmd_reply);

            // Prepare the command result.
            CommandServerBase::prepareCommandResult(cmd_reply.result, rep_id_buff);
            zmq::message_t message_rep_id(rep_id_buff.get(), sizeof(ServerResult));

            // Prepare the multipart msg.
            zmq::multipart_t multipart_msg;
            multipart_msg.add(std::move(message_rep_id));

            // Specific data.
            if(cmd_reply.result == ServerResult::COMMAND_OK && cmd_reply.params_size != 0)
            {
                // Prepare the custom response.
                zmq::message_t message_rep_custom(cmd_reply.params.get(), cmd_reply.params_size);
                multipart_msg.add(std::move(message_rep_custom));
            }

            // Sending callback.
            this->onSendingResponse(cmd_reply);

            // Send the message.
            try
            {
                multipart_msg.send(*this->server_socket_);
            }
            catch (const zmq::error_t &error)
            {
                // Check if we want to close the server.
                // The error code is for ZMQ EFSM error.
                if(!(error.num() == common::kZmqEFSMError && !this->server_working_))
                    this->onServerError(error, "Error while sending a response.");
            }
        }
    }

    // Delete pointers for clean finish the worker.
    if (this->server_socket_)
    {
        delete this->server_socket_;
        this->server_socket_ = nullptr;
    }
}

ServerResult CommandServerBase::recvFromSocket(CommandRequest& request)
{
    // Result variable.
    ServerResult result = ServerResult::COMMAND_OK;

    // Containers.
    bool recv_result;
    zmq::multipart_t multipart_msg;

    // Try to receive data. If an execption is thrown, receiving fails and an error code is generated.
    try
    {
        // Call to the internal waiting command callback.
        this->onWaitingCommand();

        // Wait the command.
        recv_result = multipart_msg.recv(*(this->server_socket_));

        // Store the raw data.
        request.raw_msg = multipart_msg.clone();
    }
    catch(zmq::error_t& error)
    {
        // Check if we want to close the server.
        // The error code is for ZMQ EFSM error.
        if(error.num() == common::kZmqEFSMError && !this->server_working_)
            return ServerResult::COMMAND_OK;

        // Else, call to error callback.
        this->onServerError(error, "Error while receiving a request.");
        return ServerResult::INTERNAL_ZMQ_ERROR;
    }

    // Check for empty msg or timeout reached.
    if (multipart_msg.empty() && !recv_result)
        return ServerResult::TIMEOUT_REACHED;
    else if (multipart_msg.empty())
        return ServerResult::EMPTY_MSG;

    // Check the multipart msg size.
    if (multipart_msg.size() == 4 || multipart_msg.size() == 5)
    {
        // Auxiliar containers.
        std::string ip;
        std::string hostname;
        std::string pid;

        // Get the multipart data.
        zmq::message_t message_ip = multipart_msg.pop();
        zmq::message_t message_hostname = multipart_msg.pop();
        zmq::message_t message_pid = multipart_msg.pop();
        zmq::message_t message_command = multipart_msg.pop();

        // Get the sizes.
        size_t ip_size_bytes = message_ip.size();
        size_t host_size_bytes = message_hostname.size();
        size_t pid_size_bytes = message_pid.size();
        size_t command_size_bytes = message_command.size();

        // First get the ip data.
        if (ip_size_bytes > 0)
            ip = std::string(static_cast<char*>(message_ip.data()), ip_size_bytes);
        else
            return ServerResult::EMPTY_CLIENT_IP;

        // Get the hostname data.
        if (host_size_bytes > 0)
            hostname = std::string(static_cast<char*>(message_hostname.data()), host_size_bytes);
        else
            return ServerResult::EMPTY_CLIENT_NAME;

        // Get the pid data.
        if (host_size_bytes > 0)
            pid = std::string(static_cast<char*>(message_pid.data()), pid_size_bytes);
        else
            return ServerResult::EMPTY_CLIENT_PID;

        // Update the client info.
        request.client = HostClient(ip, hostname, pid);
        request.client.last_connection = std::chrono::steady_clock::now();

        // Update the last connection if the client is connected.
        this->updateClientLastConnection(request.client.id);

        // Get the command.
        if (command_size_bytes == sizeof(ServerCommand))
        {
            int raw_command;
            utils::binarySerializeDeserialize(message_command.data(), sizeof(ServerCommand),
&raw_command);
            // Validate the command.
            if(CommandServerBase::validateCommand(raw_command))
                request.command = static_cast<ServerCommand>(raw_command);
            else
            {
                request.command = ServerCommand::INVALID_COMMAND;
                return ServerResult::INVALID_MSG;
            }
        }
        else
        {
            request.command = ServerCommand::INVALID_COMMAND;
            return ServerResult::INVALID_MSG;
        }

        // If there is still one more part, they are the parameters.
        if (multipart_msg.size() == 1)
        {
            // Get the message and the size.
            zmq::message_t message_params = multipart_msg.pop();
            size_t params_size_bytes = message_params.size();

            // Check the parameters.
            if(params_size_bytes > 0)
            {
                // Get and store the parameters data.
                std::unique_ptr<std::uint8_t> params =
                    std::unique_ptr<std::uint8_t>(new std::uint8_t[params_size_bytes]);
                auto *params_pointer = static_cast<std::uint8_t*>(message_params.data());
                std::copy(params_pointer, params_pointer + params_size_bytes, params.get());
                request.params = std::move(params);
                request.params_size = params_size_bytes;
            }
            else
                return ServerResult::EMPTY_PARAMS;
        }
    }
    else
        return ServerResult::INVALID_PARTS;

    // Return the result.
    return result;
}

void CommandServerBase::prepareCommandResult(ServerResult result, std::unique_ptr<std::uint8_t>&
data_out)
{
    data_out = std::unique_ptr<std::uint8_t>(new std::uint8_t[sizeof(ServerResult)]);
    utils::binarySerializeDeserialize(&result, sizeof(ServerResult), data_out.get());
}

bool CommandServerBase::validateCommand(int raw_command)
{
    // Auxiliar variables.
    bool result = false;
    int reserved_cmd = static_cast<int>(common::ServerCommand::RESERVED_COMMANDS);
    int end_base_cmd = static_cast<int>(common::ServerCommand::END_BASE_COMMANDS);
    // Check if the command is valid.
    if (raw_command >= common::kMinBaseCmdId && raw_command < reserved_cmd)
        result = true;
    else if(raw_command > end_base_cmd)
        result = true;
    return result;
}

void CommandServerBase::processCommand(const CommandRequest& request, CommandReply& reply)
{
    // First of all, call to the internal callback.
    this->onCommandReceived(request);

    // Process the different commands.
    // 1 - Process is the connect request.
    // 2 - If the command is other, check if the client is connected to the server.
    // 3 - If it is, check if the command is valid.
    // 4 - If valid, process the rest of the base commands or the custom command.
    if (ServerCommand::REQ_CONNECT == request.command)
    {
        reply.result = this->execReqConnect(request);
    }
    else if(this->connected_clients_.find(request.client.id) == this->connected_clients_.end())
    {
        reply.result = ServerResult::CLIENT_NOT_CONNECTED;
    }
    else if (ServerCommand::REQ_DISCONNECT == request.command)
    {
        reply.result = this->execReqDisconnect(request);
    }
    else if (ServerCommand::REQ_ALIVE == request.command)
    {
        reply.result = ServerResult::COMMAND_OK;
    }
    else
    {
        // Custom command, so call the internal callback.
        this->onCustomCommandReceived(request, reply);

        // Chek for an invalid msg.
        if(reply.result == ServerResult::INVALID_MSG)
            this->onInvalidMsgReceived(request);
    }
}

void CommandServerBase::checkClientsAliveStatus()
{
    // Safe mutex lock
    std::unique_lock<std::mutex> lock(this->mtx_);

    // Auxiliar containers.
    std::vector<std::string> dead_clients;
    std::chrono::milliseconds timeout(common::kDefaultClientAliveTimeoutMsec);
    std::chrono::milliseconds min_remaining_time = timeout;

    // Get the current time.
    utils::SCTimePointStd now = std::chrono::steady_clock::now();

    // Check each connection.
    for(auto& client : this->connected_clients_)
    {
        // Get the last connection time.
        const auto& last_conn = client.second.last_connection;
        // Check if the client reaches the timeout checking the last connection time.
        auto since_last_conn = std::chrono::duration_cast<std::chrono::milliseconds>(now - last_conn);
        if(since_last_conn >= timeout)
        {
            // If dead, call the onDead callback and quit the client from the map.
            this->onDeadClient(client.second);
            dead_clients.push_back(client.first);
        }
        else
        {
            // If the client is not dead, check the minor timeout of the client to set
            // with the remain time to reach the timeout.
            min_remaining_time = std::min(min_remaining_time, timeout - since_last_conn);
        }
    }

    // Remove dead clients from the map.
    for(auto& client : dead_clients)
    {
        this->connected_clients_.erase(client);
    }

    // Disable the timeout if no clients remains or set the socket timeout to the
    // minimum remaining time to the timeout among all clients.
    if(this->connected_clients_.empty())
    {
        this->server_socket_->set(zmq::sockopt::rcvtimeo, -1);
    }
    else
    {
        this->server_socket_->set(zmq::sockopt::rcvtimeo,
static_cast<int>(min_remaining_time.count()));
    }
}

```

```cpp
void CommandServerBase::updateClientLastConnection(const std::string &id)
{
    // Safe mutex lock.
    std::unique_lock<std::mutex> lock(this->mtx_);
    // Update the client last connection.
    auto client_itr = this->connected_clients_.find(id);
    if(client_itr != this->connected_clients_.end())
        client_itr->second.last_connection = std::chrono::steady_clock::now();
}

void CommandServerBase::updateServerTimeout()
{
    // Calculate the minor timeout to set it into the socket.
    auto min_timeout = std::min_element(this->connected_clients_.begin(), this->connected_clients_.end(),
        [](const auto& a, const auto& b)
        {
        auto diff_a = std::chrono::duration_cast<std::chrono::milliseconds>(
            std::chrono::steady_clock::now() - a.second.last_connection);
        auto diff_b = std::chrono::duration_cast<std::chrono::milliseconds>(
            std::chrono::steady_clock::now() - b.second.last_connection);
        return diff_a.count() < diff_b.count();
        });

    if (min_timeout != this->connected_clients_.end())
    {
        auto remain_time = common::kDefaultClientAliveTimeoutMsec - std::chrono::duration_cast<std::chrono::milliseconds>(
                                   std::chrono::steady_clock::now() - min_timeout->second.last_connection).count();
        this->server_socket_->set(zmq::sockopt::rcvtimeo, std::max(0, static_cast<int>(remain_time)));
    }
    else
    {
        this->server_socket_->set(zmq::sockopt::rcvtimeo, -1);
    }
}

void CommandServerBase::resetSocket()
{
    // Auxiliar variables.
    int res = 0;
    const zmq::error_t* last_error;
    unsigned reconnect_count = common::kServerReconnTimes;

    // Delete the previous socket.
    if (this->server_socket_)
    {
        delete this->server_socket_;
        this->server_socket_ = nullptr;
    }
    // Try creating a new socket.
    do
    {
        try
        {
            // Create the ZMQ rep socket.
            std::this_thread::sleep_for(std::chrono::microseconds(500));
            this->server_socket_ = new zmq::socket_t(*this->context_, zmq::socket_type::rep);
            this->server_socket_->bind(this->server_endpoint_);
            this->server_socket_->set(zmq::sockopt::linger, 0);
        }
        catch (const zmq::error_t& error)
        {
            // Delete the socket and store the last error.
            delete this->server_socket_;
            this->server_socket_ = nullptr;
            last_error = &error;
        }
        reconnect_count--;
    } while (res == EADDRINUSE && reconnect_count > 0);

    if (!this->server_socket_ )
    {
        // Update the working flag and calls to the callback.
        this->server_working_ = false;
        this->onServerError(*last_error, "Error during socket creation.");
    }
    else
    {
        // Call to the internal callback.
        this->onServerStart();
    }
}

void CommandServerBase::onCustomCommandReceived(const CommandRequest&, CommandReply& rep)
{
    rep.result = ServerResult::NOT_IMPLEMENTED;
}

} // END NAMESPACES.
// =================================================================================================================
```

7.41 sources/CommandServerClient/common.cpp File Reference

`#include "LibZMQUtils/CommandServerClient/common.h"`

7.42 common.cpp

Go to the documentation of this file.

```
/*************************************************************************************************************************
 *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.                         *
 *                                                                                                                     *
 *   Copyright (C) 2023 Degoras Project Team                                                                          *
 *                        < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >                          *
 *                        < Jesús Relinque Madroñal >                                                                  *
 *                                                                                                                     *
 *   This file is part of LibZMQUtils.                                                                                 *
 *                                                                                                                     *
 *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of the EUPL license   *
 *   as soon they will be approved by the European Commission (IDABC).                                                 *
 *                                                                                                                     *
 *   This project is free software: you can redistribute it and/or modify it under the terms of the EUPL license as   *
 *   published by the IDABC, either Version 1.2 or, at your option, any later version.                                 *
 *                                                                                                                     *
 *   This project is distributed in the hope that it will be useful. Unless required by applicable law or agreed to in *
 *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND; without even the  *
 *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to check specific   *
 *   language governing permissions and limitations and more details.                                                  *
 *                                                                                                                     *
 *   You should use this project in compliance with the EUPL license. You should have received a copy of the license   *
 *   along with this project. If not, see the license at < https://eupl.eu/ >.                                         *
 **********************************************************************************************************************/

#include "LibZMQUtils/CommandServerClient/common.h"


zmqutils::common::HostClient::HostClient(const std::string &ip, const std::string &name,
                                                const std::string &pid, const std::string &info) :
    ip(ip),
    hostname(name),
    pid(pid),
    info(info)
{
    // Create the host client internal identification.
    this->id = ip + "//" + name + "//" + pid;
}
```

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↩ _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.

```
/*******************************************************************************************************************
 *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
*
 *
*
 *   Copyright (C) 2023 Degoras Project Team
*
 *                      < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
*
 *                      < Jesús Relinque Madroñal >
*
 *
*
 *   This file is part of LibZMQUtils.
*
 *
*
```

```
 *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of the EUPL license    *
 *   as soon they will be approved by the European Commission (IDABC).                                                 *
 *                                                                                                                     *
 *   This project is free software: you can redistribute it and/or modify it under the terms of the EUPL license as    *
 *   published by the IDABC, either Version 1.2 or, at your option, any later version.                                 *
 *                                                                                                                     *
 *   This project is distributed in the hope that it will be useful. Unless required by applicable law or agreed to in *
 *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND; without even the  *
 *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to check specific   *
 *   language governing permissions and limitations and more details.                                                 *
 *                                                                                                                     *
 *   You should use this project in compliance with the EUPL license. You should have received a copy of the license   *
 *   along with this project. If not, see the license at < https://eupl.eu/ >.                                         *
 **********************************************************************************************************************/



#include <iomanip>
#ifdef _WIN32
#include <winsock2.h>
#include <ws2tcpip.h>
#include <Windows.h>
#else
#include <sys/socket.h>
#include <ifaddrs.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#endif

#include <sstream>
#include <iphlpapi.h>

#include "LibZMQUtils/utils.h"

#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0600
#elif _WIN32_WINNT < 0x0600
#undef _WIN32_WINNT
#define _WIN32_WINNT 0x0600
#endif

// ZMQUTILS NAMESPACES
// =====================================================================================================================
namespace zmqutils{
namespace utils{
// =====================================================================================================================

// =====================================================================================================================
using std::chrono::duration;
using std::chrono::duration_cast;
using std::chrono::high_resolution_clock;
using std::chrono::time_point_cast;
// =====================================================================================================================

std::vector<NetworkAdapterInfo> getHostIPsWithInterfaces()
{
    // Result container.
    std::vector<NetworkAdapterInfo> adapters;

#ifdef _WIN32

    // Buffer size.
    ULONG  buff_size = 0;

    if (GetAdaptersAddresses(AF_INET, GAA_FLAG_SKIP_ANYCAST | GAA_FLAG_SKIP_MULTICAST | GAA_FLAG_SKIP_DNS_SERVER,
                             nullptr, nullptr, &buff_size) != ERROR_BUFFER_OVERFLOW)
    {
        return adapters;
    }

    std::vector<char> buffer(buff_size);

    PIP_ADAPTER_ADDRESSES adapter_addrs = reinterpret_cast<PIP_ADAPTER_ADDRESSES>(&buffer[0]);

    if (GetAdaptersAddresses(AF_INET, GAA_FLAG_SKIP_ANYCAST | GAA_FLAG_SKIP_MULTICAST | GAA_FLAG_SKIP_DNS_SERVER,
                             nullptr, adapter_addrs, &buff_size) != NO_ERROR)
    {
        return adapters;
    }

    while (adapter_addrs != nullptr)
    {
        if (adapter_addrs->OperStatus == IfOperStatusUp)
        {
            PIP_ADAPTER_UNICAST_ADDRESS unicast_addrs = adapter_addrs->FirstUnicastAddress;
            while (unicast_addrs != nullptr)
            {
                sockaddr_in* sockaddr = reinterpret_cast<sockaddr_in*>(unicast_addrs->Address.lpSockaddr);
                char* ip = inet_ntoa(sockaddr->sin_addr);
                char f_name_ch[260];
                char desc_ch[260];
                char df_char = ' ';

                WideCharToMultiByte(CP_ACP,0,adapter_addrs->FriendlyName,-1, f_name_ch,260, &df_char, NULL);
                WideCharToMultiByte(CP_ACP,0,adapter_addrs->Description,-1, desc_ch,260, &df_char, NULL);

                NetworkAdapterInfo adaptr;
                adaptr.id = std::string(adapter_addrs->AdapterName);
                adaptr.name = std::string(f_name_ch);
                adaptr.descr = std::string(desc_ch);
                adaptr.ip = std::string(ip);
                adapters.push_back(adaptr);
                unicast_addrs = unicast_addrs->Next;
            }
        }

        adapter_addrs = adapter_addrs->Next;
    }
#else
  // TODO
#endif

    // Return the ip interface maps.
    return adapters;
}

std::string getHostname()
{
    std::string name;

#ifdef _WIN32

    WSADATA wsaData;

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
        return "";

    char buffer[256];
    if (gethostname(buffer, sizeof(buffer)) != 0)
        WSACleanup();

    // Clear.
    WSACleanup();

    // Store the data.
    name = std::string(buffer);

#else
    // TODO
#endif

    // Return the hostname.
    return name;
}

void binarySerializeDeserialize(const void *data, size_t data_size_bytes, void *dest)
{
    const std::uint8_t* data_byes = reinterpret_cast<const std::uint8_t *>(data);
    std::uint8_t* dest_byes = reinterpret_cast<std::uint8_t*>(dest);
    std::reverse_copy(data_byes, data_byes + data_size_bytes, dest_byes);
```

```
}

std::string timePointToString(const HRTimePointStd &tp, const std::string &format, bool add_ms, bool add_ns, bool utc)
{
    // Stream to hold the formatted string and the return container.
    std::ostringstream ss;
    // Convert the time point to a duration and get the different time fractions.
    HRTimePointStd::duration dur = tp.time_since_epoch();
    const time_t secs = duration_cast<std::chrono::seconds>(dur).count();
    const long long mill = duration_cast<std::chrono::milliseconds>(dur).count();
    const unsigned long long ns = duration_cast<std::chrono::nanoseconds>(dur).count();
    const unsigned long long s_ns = secs * 1e9;
    const unsigned long long t_ns = (ns - s_ns);
    // Format the duration.
    if (const std::tm *tm = (utc ? std::gmtime(&secs) : std::localtime(&secs)))
    {
        ss « std::put_time(tm, format.c_str());
        if(add_ms && !add_ns)
            ss « '.' « std::setw(3) « std::setfill('0') « (mill - secs * 1e3);
        else if(add_ns)
            ss « '.' « std::setw(9) « std::setfill('0') « t_ns;
    }
    else
    {
        // If error, return an empty string.
        return std::string();
    }
    // Return the container.
    return ss.str();
}

std::string timePointToIso8601(const HRTimePointStd &tp, bool add_ms, bool add_ns)
{
    // Return the ISO 8601 datetime.
    return timePointToString(tp, "%Y-%m-%dT%H:%M:%S", add_ms, add_ns) + 'Z';
}

std::string currentISO8601Date(bool add_ms)
{
    auto now = high_resolution_clock::now();
    return timePointToIso8601(now, add_ms);
}

unsigned getCurrentPID()
{
#if defined(_WIN32)
    return GetCurrentProcessId();
#elif defined(__unix__) || defined(__APPLE__) || defined(__linux__)
    return getpid();
#else
// Unsupported or unknown platform
    return 0;
#endif
}

}} // END NAMESPACES.
// =====================================================================================================================
```

Index