LibZMQUtils

v2307.1

Generated by Doxygen 1.9.7

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 zmq::from_handle_t::_private Struct Reference

4.1.1 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

4.2 amelas::common::AltAzPos Struct Reference

Public Member Functions

- AltAzPos (double az, double el)

Public Attributes

- double az
- double el

4.2.1 Detailed Description

Definition at line 36 of file common.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 AltAzPos() [1/2]

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

Definition at line 38 of file common.h.

4.2.2.2 AltAzPos() [2/2]

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

Definition at line 41 of file common.h.

4.2.3 Member Data Documentation

4.2.3.1 az

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

Definition at line 43 of file common.h.

4.2.3.2 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

4.3 amelas::AmelasController Class Reference

Public Member Functions

- ControllerError setHomePosition (const AltAzPos &pos)
- ControllerError getHomePosition (AltAzPos &pos)
- ControllerError getDatetime (std::string &)

4.3.1 Detailed Description

Definition at line 52 of file amelas_controller.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 AmelasController()

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

Definition at line 57 of file amelas_controller.h.

4.3.3 Member Function Documentation

4.3.3.1 getDatetime()

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

Definition at line 100 of file amelas_controller.h.

4.3.3.2 getHomePosition()

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

Definition at line 87 of file amelas_controller.h.

4.3.3.3 setHomePosition()

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

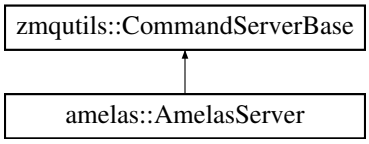
Definition at line 61 of file amelas_controller.h.

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

- examples/ExampleZMQCommanServerAmelas/AmelasExampleController/amelas_controller.h

4.4 amelas::AmelasServer Class Reference

Inheritance diagram for amelas::AmelasServer:



Public Member Functions

- AmelasServer (unsigned port, const std::string &local_addr="*")
- 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...))

Public Member Functions inherited from zmqutils::CommandServerBase

- 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.

Additional Inherited Members

Protected Member Functions inherited from zmqutils::CommandServerBase

- 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.

4.4.1 Detailed Description

Definition at line 31 of file amelas_server.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 AmelasServer()

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

Definition at line 18 of file amelas_server.cpp.

4.4.3 Member Function Documentation

4.4.3.1 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 43 of file amelas_server.h.

4.4.3.2 setCallback() [2/2]

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

Definition at line 37 of file amelas_server.h.

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

- examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/amelas_server.h
- examples/ExampleZMQCommanServerAmelas/AmelasExampleServer/amelas_server.cpp

4.5 zmqutils::CommandClientBase Class Reference

Public Types

- enum class **CommandError** : std::uint32_t {
  NOT_ERROR , **NO_COMMAND** , **NOT_CONNECTED** , **ALREADY_DISCONNECTED** ,
  ALREADY_CONNECTED , **BAD_PARAMETERS** , **COMMAND_FAILED** , **NOT_IMPLEMENTED** }

Public Member Functions

- CommandClientBase (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)
- virtual int sendCommand (const CommandData &msg, void ∗&data_out, size_t &out_bytes)
- int sendBadCommand1 (void ∗&data_out, size_t &out_bytes)

Static Public Attributes

- static const int kClientAliveTimeoutMsec = 5000
- static const int kClientSendAlivePeriodMsec = 3000

4.5.1 Detailed Description

Definition at line 79 of file command_client.h.

4.5.2 Member Enumeration Documentation

4.5.2.1 CommandError

```
enum class zmqutils::CommandClientBase::CommandError :  std::uint32_t  [strong]
```

Definition at line 89 of file command_client.h.

4.5.3 Constructor & Destructor Documentation

4.5.3.1 CommandClientBase()

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

Definition at line 25 of file command_client.cpp.

4.5.3.2 ∼CommandClientBase()

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

Definition at line 34 of file command_client.cpp.

4.5.4 Member Function Documentation

4.5.4.1 resetClient()

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

Definition at line 117 of file command_client.cpp.

4.5.4.2 sendBadCommand1()

```
int zmqutils::CommandClientBase::sendBadCommand1 (
            void *& data_out,
            size_t & out_bytes )
```

Definition at line 196 of file command_client.cpp.

4.5.4.3 sendCommand()

```
int zmqutils::CommandClientBase::sendCommand (
            const CommandData & msg,
            void *& data_out,
            size_t & out_bytes )  [virtual]
```

Definition at line 165 of file command_client.cpp.

4.5.4.4 setClientHostIP()

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

Definition at line 161 of file command_client.cpp.

4.5.4.5 setClientId()

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

Definition at line 163 of file command_client.cpp.

4.5.4.6 startAutoAlive()

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

Definition at line 145 of file command_client.cpp.

4.5.4.7 startClient()

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

Definition at line 41 of file command_client.cpp.

4.5.4.8 stopAutoAlive()

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

Definition at line 151 of file command_client.cpp.

4.5.4.9 stopClient()

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

Definition at line 95 of file command_client.cpp.

4.5.5 Member Data Documentation

4.5.5.1 kClientAliveTimeoutMsec

```
const int zmqutils::CommandClientBase::kClientAliveTimeoutMsec = 5000  [static]
```

Definition at line 86 of file command_client.h.

4.5.5.2 kClientSendAlivePeriodMsec

```
const int zmqutils::CommandClientBase::kClientSendAlivePeriodMsec = 3000  [static]
```

Definition at line 87 of file command_client.h.

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

- includes/LibZMQUtils/CommandServerClient/command_client.h
- sources/command_client.cpp

4.6 zmqutils::CommandData Struct Reference

Public Member Functions

- CommandData (CommandType id)

Public Attributes

- CommandType command_id
- std::unique_ptr< std::uint8_t > params
- size_t params_size

4.6.1 Detailed Description

Definition at line 67 of file command_client.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 CommandData()

```
zmqutils::CommandData::CommandData (
            CommandType id )  [inline]
```

Definition at line 69 of file command_client.h.

4.6.3 Member Data Documentation

4.6.3.1 command_id

```
CommandType zmqutils::CommandData::command_id
```

Definition at line 74 of file command_client.h.

4.6.3.2 params

```
std::unique_ptr<std::uint8_t> zmqutils::CommandData::params
```

Definition at line 75 of file command_client.h.

4.6.3.3 params_size

```
size_t zmqutils::CommandData::params_size
```

Definition at line 76 of file command_client.h.

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

- includes/LibZMQUtils/CommandServerClient/command_client.h

4.7 zmqutils::common::CommandReply Struct Reference

Public Attributes

- BaseServerCommand request_cmd
- std::unique_ptr< std::uint8_t > params
- size_t params_size
- BaseServerResult result

4.7.1 Detailed Description

Definition at line 206 of file common.h.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 CommandReply()

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

Definition at line 208 of file common.h.

4.7.3 Member Data Documentation

4.7.3.1 params

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

Definition at line 216 of file common.h.

4.7.3.2 params_size

```
size_t zmqutils::common::CommandReply::params_size
```

Definition at line 217 of file common.h.

4.7.3.3 request_cmd

```
BaseServerCommand zmqutils::common::CommandReply::request_cmd
```

Definition at line 215 of file common.h.

4.7.3.4 result

```
BaseServerResult zmqutils::common::CommandReply::result
```

Definition at line 218 of file common.h.

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

- includes/LibZMQUtils/CommandServerClient/common.h

4.8 zmqutils::common::CommandRequest Struct Reference

Public Attributes

- HostClient client
- BaseServerCommand command
- std::unique_ptr< std::uint8_t > params
- zmq::multipart_t raw_msg
- size_t params_size

4.8.1 Detailed Description

Definition at line 191 of file common.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 CommandRequest()

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

Definition at line 193 of file common.h.

4.8.3 Member Data Documentation

4.8.3.1 client

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

Definition at line 199 of file common.h.

4.8.3.2 command

```
BaseServerCommand zmqutils::common::CommandRequest::command
```

Definition at line 200 of file common.h.

4.8.3.3 params

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

Definition at line 201 of file common.h.

4.8.3.4 params_size

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

Definition at line 203 of file common.h.

4.8.3.5 raw_msg

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

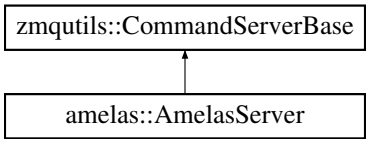
Definition at line 202 of file common.h.

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

- includes/LibZMQUtils/CommandServerClient/common.h

4.9 zmqutils::CommandServerBase Class Reference

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.

4.9.1 Detailed Description

Definition at line 79 of file command_server.h.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 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.

4.9.2.2 ~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.

4.9.3 Member Function Documentation

4.9.3.1 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.

4.9.3.2 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.

4.9.3.3 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.

4.9.3.4 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.

4.9.3.5 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.

4.9.3.6 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 164 of file command_server.h.

4.9.3.7 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.

4.9.3.8 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.

4.9.3.9 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 636 of file command_server.cpp.

4.9.3.10 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.

4.9.3.11 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.

4.9.3.12 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.

4.9.3.13 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.

4.9.3.14 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.

4.9.3.15 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.

4.9.3.16 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.

4.9.3.17 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.

4.9.3.18 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.

4.9.3.19 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.

4.9.3.20 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/command_server.cpp

4.10 zmq::context_t Class Reference

Public Member Functions

- context_t (int io_threads_, int max_sockets_=ZMQ_MAX_SOCKETS_DFLT)
- 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

4.10.1 Detailed Description

Definition at line 798 of file zmq.hpp.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 context_t() [1/2]

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

Definition at line 801 of file zmq.hpp.

4.10.2.2 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.

4.10.2.3 ~context_t()

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

Definition at line 832 of file zmq.hpp.

4.10.3 Member Function Documentation

4.10.3.1 close()

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

Definition at line 866 of file zmq.hpp.

4.10.3.2 getctxopt()

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

Definition at line 843 of file zmq.hpp.

4.10.3.3 handle()

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

Definition at line 898 of file zmq.hpp.

4.10.3.4 operator bool()

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

Definition at line 901 of file zmq.hpp.

4.10.3.5 operator void *()

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

Definition at line 894 of file zmq.hpp.

4.10.3.6 operator void const *()

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

Definition at line 896 of file zmq.hpp.

4.10.3.7 setctxopt()

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

Definition at line 835 of file zmq.hpp.

4.10.3.8 shutdown()

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

Definition at line 883 of file zmq.hpp.

4.10.3.9 swap()

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

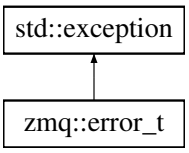
Definition at line 903 of file zmq.hpp.

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

- external/zmq/includes/zmq/zmq.hpp

4.11 zmq::error_t Class Reference

Inheritance diagram for zmq::error_t:



Public Member Functions

- error_t (int err) ZMQ_NOTHROW
- virtual const char * what () const ZMQ_NOTHROW ZMQ_OVERRIDE
- int num () const ZMQ_NOTHROW

4.11.1 Detailed Description

Definition at line 289 of file zmq.hpp.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 error_t() [1/2]

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

Definition at line 292 of file zmq.hpp.

4.11.2.2 error_t() [2/2]

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

Definition at line 293 of file zmq.hpp.

4.11.3 Member Function Documentation

4.11.3.1 num()

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

Definition at line 298 of file zmq.hpp.

4.11.3.2 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

4.12 zmq::from_handle_t Struct Reference

Classes

- struct _private

Public Member Functions

- ZMQ_CONSTEXPR_FN ZMQ_EXPLICIT from_handle_t (_private) ZMQ_NOTHROW

4.12.1 Detailed Description

Definition at line 2094 of file zmq.hpp.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 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

4.13 zmqutils::common::HostClient Struct Reference

Public Member Functions

- **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.

4.13.1 Detailed Description

Definition at line 167 of file common.h.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 HostClient()

```
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.

4.13.3 Member Data Documentation

4.13.3.1 hostname

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

Host client name.

Definition at line 185 of file common.h.

4.13.3.2 id

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

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

Definition at line 183 of file common.h.

4.13.3.3 info

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

Host client information.

Definition at line 187 of file common.h.

4.13.3.4 ip

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

Host client ip.

Definition at line 184 of file common.h.

4.13.3.5 last_connection

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

Host client last connection time.

Definition at line 188 of file common.h.

4.13.3.6 pid

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

PID of the host client process.

Definition at line 186 of file common.h.

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

- includes/LibZMQUtils/CommandServerClient/common.h
- sources/common.cpp

4.14 zmq::message_t Class Reference

Public Member Functions

- 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)
- 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

4.14.1 Detailed Description

Definition at line 408 of file zmq.hpp.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 message_t() [1/5]

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

Definition at line 411 of file zmq.hpp.

4.14.2.2 message_t() [2/5]

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

Definition at line 417 of file zmq.hpp.

4.14.2.3 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.

4.14.2.4 message_t() [4/5]

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

Definition at line 437 of file zmq.hpp.

4.14.2.5 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.

4.14.2.6 ~message_t()

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

Definition at line 506 of file zmq.hpp.

4.14.3 Member Function Documentation

4.14.3.1 copy() [1/2]

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

Definition at line 580 of file zmq.hpp.

4.14.3.2 copy() [2/2]

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

Definition at line 573 of file zmq.hpp.

4.14.3.3 data() [1/4]

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

Definition at line 595 of file zmq.hpp.

4.14.3.4 data() [2/4]

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

Definition at line 609 of file zmq.hpp.

4.14.3.5 data() [3/4]

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

Definition at line 593 of file zmq.hpp.

4.14.3.6 data() [4/4]

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

Definition at line 607 of file zmq.hpp.

4.14.3.7 empty()

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

Definition at line 605 of file zmq.hpp.

4.14.3.8 equal()

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

Definition at line 615 of file zmq.hpp.

4.14.3.9 get()

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

Definition at line 629 of file zmq.hpp.

4.14.3.10 gets()

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

Definition at line 639 of file zmq.hpp.

4.14.3.11 handle() [1/2]

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

Definition at line 736 of file zmq.hpp.

4.14.3.12 handle() [2/2]

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

Definition at line 735 of file zmq.hpp.

4.14.3.13 more()

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

Definition at line 587 of file zmq.hpp.

4.14.3.14 move() [1/2]

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

Definition at line 565 of file zmq.hpp.

4.14.3.15 move() [2/2]

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

Definition at line 558 of file zmq.hpp.

4.14.3.16 operator"!=()

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

Definition at line 623 of file zmq.hpp.

4.14.3.17 operator==()

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

Definition at line 617 of file zmq.hpp.

4.14.3.18 rebuild() [1/5]

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

Definition at line 512 of file zmq.hpp.

4.14.3.19 rebuild() [2/5]

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

Definition at line 542 of file zmq.hpp.

4.14.3.20 rebuild() [3/5]

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

Definition at line 531 of file zmq.hpp.

4.14.3.21 rebuild() [4/5]

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

Definition at line 521 of file zmq.hpp.

4.14.3.22 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.

4.14.3.23 size()

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

Definition at line 600 of file zmq.hpp.

4.14.3.24 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.

4.14.3.25 swap()

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

Definition at line 729 of file zmq.hpp.

4.14.3.26 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

4.15 zmq::monitor_t Class Reference

Public Member Functions

- 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_)

4.15.1 Detailed Description

Definition at line 2304 of file zmq.hpp.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 monitor_t()

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

Definition at line 2307 of file zmq.hpp.

4.15.2.2 ~monitor_t()

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

Definition at line 2309 of file zmq.hpp.

4.15.3 Member Function Documentation

4.15.3.1 abort()

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

Definition at line 2480 of file zmq.hpp.

4.15.3.2 check_event()

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

Definition at line 2361 of file zmq.hpp.

4.15.3.3 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.

4.15.3.4 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.

4.15.3.5 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.

4.15.3.6 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.

4.15.3.7 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.

4.15.3.8 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.

4.15.3.9 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.

4.15.3.10 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.

4.15.3.11 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.

4.15.3.12 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.

4.15.3.13 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.

4.15.3.14 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.

4.15.3.15 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.

4.15.3.16 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.

4.15.3.17 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.

4.15.3.18 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.

4.15.3.19 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.

4.15.3.20 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.

4.15.3.21 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.

4.15.3.22 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

4.16 zmqutils::utils::NetworkAdapterInfo Struct Reference

Public Attributes

- std::string id
- std::string name
- std::string descr
- std::string ip

4.16.1 Detailed Description

Definition at line 78 of file utils.h.

4.16.2 Member Data Documentation

4.16.2.1 descr

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

Definition at line 82 of file utils.h.

4.16.2.2 id

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

Definition at line 80 of file utils.h.

4.16.2.3 ip

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

Definition at line 83 of file utils.h.

4.16.2.4 name

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

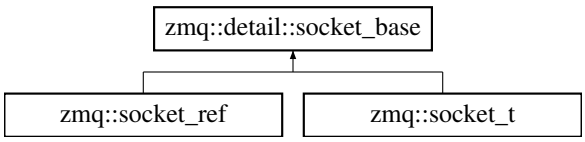
Definition at line 81 of file utils.h.

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

- includes/LibZMQUtils/utils.h

4.17 zmq::detail::socket_base Class Reference

Inheritance diagram for zmq::detail::socket_base:



Public Member Functions

- 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_

4.17.1 Detailed Description

Definition at line 1727 of file zmq.hpp.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 socket_base() [1/2]

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

Definition at line 1730 of file zmq.hpp.

4.17.2.2 socket_base() [2/2]

```
ZMQ_EXPLICIT zmq::detail::socket_base::socket_base (
            void * handle )  [inline]
```

Definition at line 1731 of file zmq.hpp.

4.17.3 Member Function Documentation

4.17.3.1 bind() [1/2]

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

Definition at line 1878 of file zmq.hpp.

4.17.3.2 bind() [2/2]

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

Definition at line 1876 of file zmq.hpp.

4.17.3.3 connect() [1/2]

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

Definition at line 1896 of file zmq.hpp.

4.17.3.4 connect() [2/2]

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

Definition at line 1894 of file zmq.hpp.

4.17.3.5 connected()

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

Definition at line 1913 of file zmq.hpp.

4.17.3.6 disconnect() [1/2]

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

Definition at line 1905 of file zmq.hpp.

4.17.3.7 disconnect() [2/2]

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

Definition at line 1903 of file zmq.hpp.

4.17.3.8 getsockopt() [1/2]

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

Definition at line 1758 of file zmq.hpp.

4.17.3.9 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.

4.17.3.10 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.

4.17.3.11 send() [2/2]

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

Definition at line 1927 of file zmq.hpp.

4.17.3.12 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.

4.17.3.13 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.

4.17.3.14 unbind() [1/2]

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

Definition at line 1887 of file zmq.hpp.

4.17.3.15 unbind() [2/2]

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

Definition at line 1885 of file zmq.hpp.

4.17.4 Member Data Documentation

4.17.4.1 flags_

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

Definition at line 1942 of file zmq.hpp.

4.17.4.2 last

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

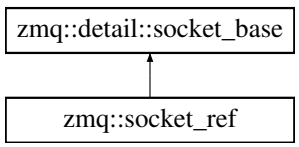
Definition at line 1942 of file zmq.hpp.

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

- external/zmq/includes/zmq/zmq.hpp

4.18 zmq::socket_ref Class Reference

Inheritance diagram for zmq::socket_ref:



Public Member Functions

- socket_ref (from_handle_t, void *handle) ZMQ_NOTHROW

Public Member Functions inherited from zmq::detail::socket_base

- 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

Additional Inherited Members

Public Attributes inherited from zmq::detail::socket_base

- T last
- T int flags_

4.18.1 Detailed Description

Definition at line 2107 of file zmq.hpp.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 socket_ref() [1/2]

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

Definition at line 2110 of file zmq.hpp.

4.18.2.2 socket_ref() [2/2]

```
zmq::socket_ref::socket_ref (
            from_handle_t ,
            void * handle )  [inline]
```

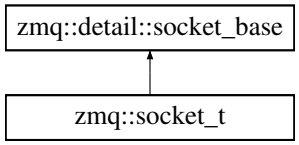
Definition at line 2114 of file zmq.hpp.

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

- external/zmq/includes/zmq/zmq.hpp

4.19 zmq::socket_t Class Reference

Inheritance diagram for zmq::socket_t:



Public Member Functions

- socket_t (context_t &context_, int type_)
- 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

Public Member Functions inherited from zmq::detail::socket_base

- 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

Friends

- class monitor_t

Additional Inherited Members

Public Attributes inherited from zmq::detail::socket_base

- T last
- T int flags_

4.19.1 Detailed Description

Definition at line 2181 of file zmq.hpp.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 socket_t() [1/2]

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

Definition at line 2186 of file zmq.hpp.

4.19.2.2 socket_t() [2/2]

```
zmq::socket_t::socket_t (
            context_t & context_,
            int type_ )  [inline]
```

Definition at line 2188 of file zmq.hpp.

4.19.2.3 ~socket_t()

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

Definition at line 2219 of file zmq.hpp.

4.19.3 Member Function Documentation

4.19.3.1 close()

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

Definition at line 2225 of file zmq.hpp.

4.19.3.2 operator socket_ref()

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

Definition at line 2242 of file zmq.hpp.

4.19.3.3 operator void *()

```
zmq::socket_t::operator void * ( )  [inline]
```

Definition at line 2221 of file zmq.hpp.

4.19.3.4 operator void const *()

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

Definition at line 2223 of file zmq.hpp.

4.19.3.5 swap()

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

Definition at line 2236 of file zmq.hpp.

4.19.4 Friends And Related Symbol Documentation

4.19.4.1 monitor_t

```
friend class monitor_t  [friend]
```

Definition at line 2183 of file zmq.hpp.

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

- external/zmq/includes/zmq/zmq.hpp

4.20 zmq_event_t Struct Reference

Public Attributes

- uint16_t event
- int32_t value

4.20.1 Detailed Description

Definition at line 207 of file zmq.hpp.

4.20.2 Member Data Documentation

4.20.2.1 event

```
uint16_t zmq_event_t::event
```

Definition at line 209 of file zmq.hpp.

4.20.2.2 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

4.21 zmq_msg_t Struct Reference

Public Attributes

- unsigned char _ [64]

4.21.1 Detailed Description

Definition at line 251 of file zmq.h.

4.21.2 Member Data Documentation

4.21.2.1 _

```
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

4.22 zmq_pollitem_t Struct Reference

Public Attributes

- void * socket
- zmq_fd_t fd
- short events
- short revents

4.22.1 Detailed Description

Definition at line 520 of file zmq.h.

4.22.2 Member Data Documentation

4.22.2.1 events

```
short zmq_pollitem_t::events
```

Definition at line 524 of file zmq.h.

4.22.2.2 fd

```
zmq_fd_t zmq_pollitem_t::fd
```

Definition at line 523 of file zmq.h.

4.22.2.3 revents

```
short zmq_pollitem_t::revents
```

Definition at line 525 of file zmq.h.

4.22.2.4 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 5

File Documentation

5.1 ExampleZMQClientAmelas.cpp

```
00001
00002
00003 #include <iostream>
00004 #include <cstring>
00005
00006 #include <LibZMQUtils/CommandClient>
00007
00008
00009
00010 using namespace zmqutils;
00011
00012 // Specific subclass commands (0 to 4 are reserved for the base server).
00013 // WARNING: In our approach, the server commands must be always in order.
00014 enum class AmelasServerCommand : common::CommandType
00015 {
00016     REQ_SET_DATETIME      = 11,
00017     REQ_GET_DATETIME      = 12,
00018     REQ_SET_HOME_POSITION = 13,
00019     REQ_GET_HOME_POSITION = 14,
00020     END_AMELAS_COMMANDS
00021 };
00022
00023 // Specific subclass errors (0 to 15 are reserved for the base server).
00024 enum class AmelasServerResult : common::CommandType
00025 {
00026     INVALID_DATETIME = 16,
00027     INVALID_POSITION = 17
00028 };
00029
00030 void parseCommand(CommandClientBase &client, const std::string &command)
00031 {
00032     void *data_out = nullptr;
00033     size_t out_size_bytes = 0;
00034     int send_result = 0;
00035
00036     char *command_str = new char[command.size()];
00037     std::copy(command.begin(), command.end(), command_str);
00038
00039     char *token = std::strtok(command_str, " ");
00040
00041     if (token)
00042     {
00043         common::CommandType command_id;
00044
00045         try
00046         {
00047             command_id = static_cast<common::CommandType>(std::stoi(token));
00048         }
00049         catch (...)
00050         {
00051             std::cerr « "Failed at sending command." « std::endl;
00052             delete[] command_str;
00053             return;
00054         }
00055
00056         CommandData command_msg(command_id);
00057         bool valid = true;
00058
```

```
00059         if (command_id == static_cast<common::CommandType>(common::BaseServerCommand::REQ_CONNECT))
00060         {
00061             std::cout « "Sending connect message" « std::endl;
00062         }
00063         else if (command_id ==
      static_cast<common::CommandType>(common::BaseServerCommand::REQ_DISCONNECT))
00064         {
00065             std::cout « "Sending disconnect message" « std::endl;
00066         }
00067         else if (command_id == static_cast<common::CommandType>(common::BaseServerCommand::REQ_ALIVE))
00068         {
00069             std::cout « "Sending keepalive command." « std::endl;
00070         }
00071         else if (command_id ==
      static_cast<common::CommandType>(AmelasServerCommand::REQ_GET_DATETIME))
00072         {
00073             std::cout « "Get datetime command not implemented yet." « std::endl;
00074             valid = false;
00075         }
00076         else if (command_id ==
      static_cast<common::CommandType>(AmelasServerCommand::REQ_SET_DATETIME))
00077         {
00078             std::cout « "Set datetime command not implemented yet." « std::endl;
00079             valid = false;
00080         }
00081         else if (command_id ==
      static_cast<common::CommandType>(AmelasServerCommand::REQ_GET_HOME_POSITION))
00082         {
00083             std::cout « "Sending get home position command." « std::endl;
00084         }
00085         else if (command_id ==
      static_cast<common::CommandType>(AmelasServerCommand::REQ_SET_HOME_POSITION))
00086         {
00087             std::cout « "Sending get home position command." « std::endl;
00088
00089             bool valid_params = true;
00090             double az = 0., el = 0.;
00091             char *param_token = std::strtok(nullptr, " ");
00092
00093             try
00094             {
00095                 az = std::stod(param_token);
00096             }
00097             catch (...)
00098             {
00099                 std::cerr « "Bad parameter azimuth issued.";
00100                 valid_params = false;
00101             }
00102
00103             if (valid_params)
00104             {
00105                 param_token = std::strtok(nullptr, " ");
00106
00107                 try
00108                 {
00109                     el = std::stod(param_token);
00110                 }
00111                 catch (...)
00112                 {
00113                     std::cerr « "Bad parameter elevation issued.";
00114                     valid_params = false;
00115                 }
00116             }
00117
00118             if (valid_params)
00119             {
00120                 std::cout«"Sending: " « az «" "«el«std::endl;
00121
00122                 command_msg.params = std::unique_ptr<std::uint8_t>(new std::uint8_t[16]);
00123                 command_msg.params_size = 16;
00124
00125                 zmqutils::utils::binarySerializeDeserialize(&az, 8, command_msg.params.get());
00126                 zmqutils::utils::binarySerializeDeserialize(&el, 8, command_msg.params.get() + 8);
00127             }
00128
00129             valid = valid_params;
00130
00131         }
00132         else
00133         {
00134             valid = false;
00135         }
00136
00137         if (valid)
00138         {
00139             send_result = client.sendCommand(command_msg, data_out, out_size_bytes);
00140
```

```
00141             if (send_result != 0)
00142             {
00143                 std::cerr « "Command sending failed with code: " « send_result « std::endl;
00144                 // Restart client if sending fails
00145                 client.resetClient();
00146             }
00147 
00148             else if (out_size_bytes >= sizeof(CommandClientBase::CommandError))
00149             {
00150                 CommandClientBase::CommandError error_response;
00151 
00152                 auto *data_bytes = static_cast<std::uint8_t*>(data_out);
00153                 zmqutils::utils::binarySerializeDeserialize(
00154                             data_bytes, sizeof(CommandClientBase::CommandError), &error_response);
00155                 std::cout « "Response code from server: "« static_cast<std::uint32_t>(error_response)
    « std::endl;
00156 
00157                 if (command_id ==
    static_cast<common::CommandType>(AmelasServerCommand::REQ_GET_HOME_POSITION))
00158                 {
00159                     double az, el;
00160                     if (out_size_bytes == sizeof(CommandClientBase::CommandError) + 16)
00161                     {
00162                         zmqutils::utils::binarySerializeDeserialize(
00163                                     data_bytes + sizeof(CommandClientBase::CommandError), 8, &az);
00164 
00165                         zmqutils::utils::binarySerializeDeserialize(
00166                                     data_bytes + sizeof(CommandClientBase::CommandError) + 8, 8, &el);
00167 
00168                         std::cout « "Get home position command result is (az,el): " « az « ", " « el «
    std::endl;
00169                     }
00170                     else
00171                         std::cerr « "Get home position command answer is incorrect. Params size is: "
    « out_size_bytes «  std::endl;
00172 
00173                 }
00174 
00175                 delete[] data_bytes;
00176             }
00177         }
00178         else
00179         {
00180             std::cerr « "Command is not implemented or valid" « std::endl;
00181         }
00182 
00183     }
00184     else
00185     {
00186         std::cerr « "Not a valid command" « std::endl;
00187     }
00188 
00189 
00190     delete[] command_str;
00191 }
00192 
00193 
00194 int main(int argc, char**argv)
00195 {
00196 
00197     int port = 9999;
00198     std::string ip = "127.0.0.1";
00199 
00200     if (argc == 2)
00201     {
00202         ip = argv[1];
00203     }
00204     if (argc == 3)
00205     {
00206         ip = argv[1];
00207         try
00208         {
00209             port = std::stoi(argv[2]);
00210         }  catch (...)
00211         {
00212             std::cerr « "Not recognized port in input: " « argv[2] « std::endl;
00213             return -1;
00214         }
00215 
00216     }
00217     else if (argc > 3)
00218     {
00219         std::cout « "Usage: ZMQClient [ip] [port]" « std::endl;
00220         return 0;
00221     }
00222 
00223     std::string endpoint = "tcp://" + ip + ":" + std::to_string(port);
```

```
00224     CommandClientBase client(endpoint);
00225     client.startClient("Ethernet");
00226     //client.setClientHostIP("");
00227     std::cout « "Connecting to endpoint: " «  endpoint « std::endl;
00228     //client.startAutoAlive();
00229     std::string command;
00230
00231     while(true)
00232     {
00233         std::cout«"Write a command: ";
00234         std::getline(std::cin, command);
00235
00236         if (command == "exit")
00237             break;
00238
00239         parseCommand(client, command);
00240     }
00241
00242     std::cout « "Requested client to stop. Bye." « std::endl;
00243
00244     client.stopClient();
00245
00246
00247     return 0;
00248 }
```

5.2 amelas_controller.h

```
00001
      /*****************************************************************************************************************************
00002  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
      *
00003  *
      *
00004  *   Copyright (C) 2023 Degoras Project Team
      *
00005  *                       < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
      *
00006  *                       < Jesús Relinque Madroñal >
      *
00007  *
      *
00008  *   This file is part of LibZMQUtils.
      *
00009  *
      *
00010  *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
      the EUPL license   *
00011  *   as soon they will be approved by the European Commission (IDABC).
      *
00012  *
      *
00013  *   This project is free software: you can redistribute it and/or modify it under the terms of the
      EUPL license as    *
00014  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
      *
00015  *
      *
00016  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
      or agreed to in *
00017  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
      without even the   *
00018  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
      check specific   *
00019  *   language governing permissions and limitations and more details.
      *
00020  *
      *
00021  *   You should use this project in compliance with the EUPL license. You should have received a copy
      of the license   *
00022  *   along with this project. If not, see the license at < https://eupl.eu/ >.
      *
00023
      *****************************************************************************************************************************/
00024
00025 // C++ INCLUDES
00026 //
      ============================================================================================================================
00027 #include <map>
00028 #include <string>
00029 //
      ============================================================================================================================
00030
```

```
00031 // ZMQUTILS INCLUDES
00032 //
      ========================================================================================================================
00033 #include <LibZMQUtils/CommandServer>
00034 #include <LibZMQUtils/Utils>
00035 //
      ========================================================================================================================
00036
00037 // PROJECT INCLUDES
00038 //
      ========================================================================================================================
00039 #include "common.h"
00040 //
      ========================================================================================================================
00041
00042
00043
00044 // AMELAS NAMESPACES
00045 //
      ========================================================================================================================
00046 namespace amelas{
00047 //
      ========================================================================================================================
00048
00049 using amelas::common::ControllerError;
00050 using amelas::common::AltAzPos;
00051
00052 class AmelasController
00053 {
00054 public:
00055
00056
00057     AmelasController() :
00058         home_pos_({0,0})
00059     {}
00060
00061     ControllerError setHomePosition(const AltAzPos& pos)
00062     {
00063         // Auxiliar result.
00064         ControllerError error = ControllerError::SUCCESS;
00065
00066         // Check the provided values.
00067         if (pos.az >= 360.0 ||  pos.az < 0.0 || pos.el >= 90. || pos.el < 0.)
00068         {
00069             error = ControllerError::INVALID_POSITION;
00070         }
00071         else
00072         {
00073             this->home_pos_ = pos;
00074         }
00075
00076         std::cout « std::string(80, '-') « std::endl;
00077         std::cout«"<AMELAS CONTROLLER>"«std::endl;
00078         std::cout«"-> SET_HOME_POSITION"«std::endl;
00079         std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
00080         std::cout«"Az: "«pos.az«std::endl;
00081         std::cout«"El: "«pos.el«std::endl;
00082         std::cout « std::string(80, '-') « std::endl;
00083
00084         return error;
00085     }
00086
00087     ControllerError getHomePosition(AltAzPos& pos)
00088     {
00089         pos = this->home_pos_;
00090
00091         std::cout « std::string(80, '-') « std::endl;
00092         std::cout«"<AMELAS CONTROLLER>"«std::endl;
00093         std::cout«"-> GET_HOME_POSITION"«std::endl;
00094         std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
00095         std::cout « std::string(80, '-') « std::endl;
00096
00097         return ControllerError::SUCCESS;
00098     }
00099
00100     ControllerError getDatetime(std::string&)
00101     {
00102         return ControllerError::SUCCESS;
00103     }
00104
00105 private:
00106
00107     AltAzPos home_pos_;
00108
00109 };
00110
00111 } // END NAMESPACES.
```

```
00112 //
      ==========================================================================================================================
00113
```

5.3 common.h

```
00001
00002 //
      ==========================================================================================================================
00003 #pragma once
00004 //
      ==========================================================================================================================
00005
00006 // C++ INCLUDES
00007 //
      ==========================================================================================================================
00008 #include <string>
00009 #include <map>
00010 #include <vector>
00011 #include <variant>
00012 #include <functional>
00013 //
      ==========================================================================================================================
00014
00015 // AMELAS NAMESPACES
00016 //
      ==========================================================================================================================
00017 namespace amelas{
00018 namespace common{
00019 //
      ==========================================================================================================================
00020
00021 // CONSTANTS
00022 //
      ==========================================================================================================================
00023
00024 //
      ==========================================================================================================================
00025
00026 // CONVENIENT ALIAS, ENUMERATIONS AND CONSTEXPR
00027 //
      ==========================================================================================================================
00028
00029 enum class ControllerError : std::uint32_t
00030 {
00031     SUCCESS = 0,
00032     INVALID_POSITION = 1,
00033     UNSAFE_POSITION = 2
00034 };
00035
00036 struct AltAzPos
00037 {
00038     AltAzPos(double az, double el):
00039         az(az), el(el){}
00040
00041     AltAzPos(): az(-1), el(-1){}
00042
00043     double az;
00044     double el;
00045 };
00046
00047 // Callback function type aliases
00048 using SetHomePositionCallback = std::function<ControllerError(const AltAzPos&)>;
00049 using GetHomePositionCallback = std::function<ControllerError(AltAzPos&)>;
00050 using GetDatetimeCallback = std::function<ControllerError(std::string&)>;
00051
00052 // Callback variant.
00053 using ControllerCallback = std::variant<SetHomePositionCallback,
00054                                         GetHomePositionCallback,
00055                                         GetDatetimeCallback>;
00056
00057
00058
00059
00060
00061 //
      ==========================================================================================================================
00062
00063 }} // END NAMESPACES.
00064 //
      ==========================================================================================================================
```

5.4 common.h

```
00001
00002 #include <functional>
00003 #include <any>
00004
00005 #include <LibZMQUtils/Utils>
00006
00007 //
      ============================================================================================================================
00008 #pragma once
00009 //
      ============================================================================================================================
00010
00011 // AMELAS NAMESPACES
00012 //
      ============================================================================================================================
00013 namespace amelas{
00014 namespace common{
00015 //
      ============================================================================================================================
00016
00017 // Specific subclass commands (0 to 4 are reserved for the base server).
00018 // WARNING: In our approach, the server commands must be always in order.
00019 enum class AmelasServerCommand : zmqutils::common::CommandType
00020 {
00021     REQ_SET_DATETIME      = 11,
00022     REQ_GET_DATETIME      = 12,
00023     REQ_SET_HOME_POSITION = 13,
00024     REQ_GET_HOME_POSITION = 14,
00025     END_AMELAS_COMMANDS
00026 };
00027
00028 // Specific subclass errors (0 to 15 are reserved for the base server).
00029 enum class AmelasServerResult : zmqutils::common::ResultType
00030 {
00031
00032 };
00033
00034 // Extend the base command strings with those of the subclass.
00035 static constexpr auto AmelasServerCommandStr = zmqutils::utils::joinArraysConstexpr(
00036     zmqutils::common::BaseServerCommandStr,
00037     std::array<const char*, 5>
00038     {
00039         "REQ_SET_DATETIME",
00040         "REQ_GET_DATETIME",
00041         "REQ_SET_HOME_POSITION",
00042         "REQ_GET_HOME_POSITION",
00043         "END_DRGG_COMMANDS"
00044     });
00045
00046 // Extend the base result strings with those of the subclass.
00047 static constexpr auto AmelasServerResultStr = zmqutils::utils::joinArraysConstexpr(
00048     zmqutils::common::BaseServerResultStr,
00049     std::array<const char*, 2>
00050     {
00051         "INVALID_DATETIME - Datetime provided is invalid.",
00052         "INVALID_POSITION - Position (az, el) provided is invalid."
00053     });
00054
00055 // Usefull const expressions.
00056 constexpr int kMinCmdId = static_cast<int>(zmqutils::common::BaseServerCommand::END_BASE_COMMANDS) +
      1;
00057 constexpr int kMaxCmdId = static_cast<int>(AmelasServerCommand::END_AMELAS_COMMANDS) - 1;
00058
00059 }} // END NAMESPACES.
00060 //
      ============================================================================================================================
```

5.5 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

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::BaseServerCommand : CommandType {
  INVALID_COMMAND = 0 , REQ_CONNECT = 1 , REQ_DISCONNECT = 2 , REQ_ALIVE = 3 ,
  RESERVED_COMMANDS = 4 , END_BASE_COMMANDS = 10 }

  Enumerates the possible commands of a base command server. They can be extended in a subclass.
- enum class zmqutils::common::BaseServerResult : CommandType {
  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 }

Variables

- constexpr int zmqutils::common::kDefaultClientAliveTimeoutMsec = 8000

  Default timeout for consider a client dead.
- constexpr unsigned zmqutils::common::kServerReconnTimes = 10

  Server reconnection default number of attempts.
- constexpr int zmqutils::common::kZmqEFSMError = 156384765

  ZMQ EFSM error.
- constexpr int zmqutils::common::kMinBaseCmdId = static_cast<int>(BaseServerCommand::INVALID_COMMAND) + 1
- constexpr int zmqutils::common::kMaxBaseCmdId = static_cast<int>(BaseServerCommand::END_BASE_COMMANDS) - 1

5.5.1 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.

5.5.2 Typedef Documentation

5.5.2.1 CommandType

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

Type used for the BaseServerCommand enumeration.
Definition at line 71 of file common.h.

5.5.2.2 ResultType

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

Type used for the BaseServerResult enumeration.
Definition at line 72 of file common.h.

5.5.3 Enumeration Type Documentation

5.5.3.1 BaseServerCommand

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

Enumerates the possible commands of a base command server. They can be extended in a subclass.

Warning

Commands 0 to 10 ids must not be used for custom commands, they are special and reserved.

Only positive commands will be acepted by the server.

Messages with the command 0, sentinel value or a reserved commands are considered invalid.

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 81 of file common.h.

5.5.3.2 BaseServerResult

```
enum class zmqutils::common::BaseServerResult : CommandType [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.
INVALID_PARTS	The command has invalid parts.
UNKNOWN_COMMAND	The command is not recognized.
INVALID_MSG	The command 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 96 of file common.h.

5.5.4 Variable Documentation

5.5.4.1 kDefaultClientAliveTimeoutMsec

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

Default timeout for consider a client dead.
Definition at line 63 of file common.h.

5.5.4.2 kMaxBaseCmdId

```
constexpr int zmqutils::common::kMaxBaseCmdId = static_cast<int>(BaseServerCommand::END_BASE↩
_COMMANDS) - 1 [constexpr]
```

Definition at line 120 of file common.h.

5.5.4.3 kMinBaseCmdId

```
constexpr int zmqutils::common::kMinBaseCmdId = static_cast<int>(BaseServerCommand::INVALID_↩
COMMAND) + 1 [constexpr]
```

Definition at line 119 of file common.h.

5.5.4.4 kServerReconnTimes

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

Server reconnection default number of attempts.
Definition at line 64 of file common.h.

5.5.4.5 kZmqEFSMError

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

ZMQ EFSM error.
Definition at line 65 of file common.h.

5.6 common.h

Go to the documentation of this file.

```
00001
00002 /*****************************************************************************************************************
 *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
00003 *
 *
00004 *   Copyright (C) 2023 Degoras Project Team
 *
00005 *                      < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
 *
00006 *                      < Jesús Relinque Madroñal >
 *
00007 *
 *
00008 *   This file is part of LibZMQUtils.
 *
00009 *
 *
00010 *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
the EUPL license   *
00011 *   as soon they will be approved by the European Commission (IDABC).
 *
00012 *
 *
00013 *   This project is free software: you can redistribute it and/or modify it under the terms of the
EUPL license as    *
00014 *   published by the IDABC, either Version 1.2 or, at your option, any later version.
 *
00015 *
 *
00016 *   This project is distributed in the hope that it will be useful. Unless required by applicable law
or agreed to in *
00017 *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
without even the  *
00018 *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
check specific   *
00019 *   language governing permissions and limitations and more details.
 *
00020 *
 *
00021 *   You should use this project in compliance with the EUPL license. You should have received a copy
of the license   *
00022 *   along with this project. If not, see the license at < https://eupl.eu/ >.
 *
00023
*****************************************************************************************************************/
00024
00033 //
      ============================================================================================================
00034 #pragma once
00035 //
      ============================================================================================================
00036
00037 // C++ INCLUDES
00038 //
      ============================================================================================================
00039 #include <string>
00040 #include <iostream>
00041 #include <map>
00042 #include <vector>
00043 #include <cstring>
00044 #include <memory>
00045 #include <zmq/zmq.hpp>
00046 #include <zmq/zmq_addon.hpp>
00047 //
      ============================================================================================================
00048
00049 // ZMQUTILS INCLUDES
00050 //
      ============================================================================================================
00051 #include "LibZMQUtils/libzmqutils_global.h"
00052 #include "LibZMQUtils/utils.h"
00053 //
      ============================================================================================================
00054
00055 // ZMQUTILS NAMESPACES
00056 //
      ============================================================================================================
```

```
00057 namespace zmqutils{
00058 namespace common{
00059 //
      ================================================================================================================
00060
00061 // CONSTANTS
00062 //
      ================================================================================================================
00063 constexpr int kDefaultClientAliveTimeoutMsec = 8000;
00064 constexpr unsigned kServerReconnTimes = 10;
00065 constexpr int kZmqEFSMError = 156384765;
00066 //
      ================================================================================================================
00067
00068 // CONVENIENT ALIAS, ENUMERATIONS AND CONSTEXPR
00069 //
      ================================================================================================================
00070
00071 using CommandType = std::uint32_t;
00072 using ResultType = std::uint32_t;
00073
00081 enum class BaseServerCommand : CommandType
00082 {
00083     INVALID_COMMAND   = 0,
00084     REQ_CONNECT       = 1,
00085     REQ_DISCONNECT    = 2,
00086     REQ_ALIVE         = 3,
00087     RESERVED_COMMANDS = 4,
00088     END_BASE_COMMANDS = 10
00089 };
00090
00096 enum class BaseServerResult : CommandType
00097 {
00098     COMMAND_OK             = 0,
00099     INTERNAL_ZMQ_ERROR     = 1,
00100     EMPTY_MSG              = 2,
00101     EMPTY_CLIENT_IP        = 3,
00102     EMPTY_CLIENT_NAME      = 4,
00103     EMPTY_CLIENT_PID       = 5,
00104     EMPTY_PARAMS           = 6,
00105     TIMEOUT_REACHED        = 7,
00106     INVALID_PARTS          = 8,
00107     UNKNOWN_COMMAND        = 9,
00108     INVALID_MSG            = 10,
00109     CLIENT_NOT_CONNECTED   = 11,
00110     ALREADY_CONNECTED      = 12,
00111     BAD_PARAMETERS         = 13,
00112     COMMAND_FAILED         = 14,
00113     NOT_IMPLEMENTED        = 15,
00114     BAD_NO_PARAMETERS      = 16,
00115     END_BASE_ERRORS        = 20
00116 };
00117
00118 // Usefull const expressions.
00119 constexpr int kMinBaseCmdId = static_cast<int>(BaseServerCommand::INVALID_COMMAND) + 1;
00120 constexpr int kMaxBaseCmdId = static_cast<int>(BaseServerCommand::END_BASE_COMMANDS) - 1;
00121
00122 static constexpr std::array<const char*, 11>  BaseServerCommandStr
00123 {
00124     "INVALID_COMMAND",
00125     "REQ_CONNECT",
00126     "REQ_DISCONNECT",
00127     "REQ_ALIVE",
00128     "RESERVED_BASE_COMMAND",
00129     "RESERVED_BASE_COMMAND",
00130     "RESERVED_BASE_COMMAND",
00131     "RESERVED_BASE_COMMAND",
00132     "RESERVED_BASE_COMMAND",
00133     "RESERVED_BASE_COMMAND",
00134     "END_BASE_COMMANDS"
00135 };
00136
00137 static constexpr std::array<const char*, 21>  BaseServerResultStr
00138 {
00139     "COMMAND_OK - Command executed.",
00140     "INTERNAL_ZMQ_ERROR - Internal ZeroMQ error.",
00141     "EMPTY_MSG - Message is empty.",
00142     "EMPTY_CLIENT_IP - Client IP missing or empty.",
00143     "EMPTY_CLIENT_NAME - Client name missing or empty.",
00144     "EMPTY_CLIENT_PID - Client pid missing or empty.",
00145     "EMPTY_PARAMS - Command parameters missing or empty.",
00146     "TIMEOUT_REACHED - Operation timed out.",
00147     "INVALID_PARTS - Command has invalid parts.",
00148     "UNKNOWN_COMMAND - Command is not recognized.",
00149     "INVALID_COMMAND - Command is invalid.",
00150     "NOT_CONNECTED - Not connected to the server.",
00151     "ALREADY_CONNECTED - Already connected to the server.",
```

```
00152     "BAD_PARAMETERS - Provided parameters are invalid.",
00153     "COMMAND_FAILED - Command execution failed.",
00154     "NOT_IMPLEMENTED - Command is not implemented.",
00155     "RESERVED_BASE_ERROR",
00156     "RESERVED_BASE_ERROR",
00157     "RESERVED_BASE_ERROR",
00158     "RESERVED_BASE_ERROR",
00159     "RESERVED_BASE_ERROR"
00160 };
00161
00162 //
    ============================================================================================================================
00163
00164 // COMMON STRUCTS
00165 //
    ============================================================================================================================
00166
00167 struct LIBZMQUTILS_EXPORT HostClient
00168 {
00169     HostClient() = default;
00170
00171     HostClient(const HostClient&) = default;
00172
00173     HostClient(HostClient&&) = default;
00174
00175     HostClient& operator=(const HostClient&) = default;
00176
00177     HostClient& operator=(HostClient&&) = default;
00178
00179     HostClient(const std::string& ip, const std::string& name,
00180                const std::string& pid, const std::string& info = "");
00181
00182     // Struct members.
00183     std::string id;
00184     std::string ip;
00185     std::string hostname;
00186     std::string pid;
00187     std::string info;
00188     utils::SCTimePointStd last_connection;
00189 };
00190
00191 struct CommandRequest
00192 {
00193     CommandRequest():
00194         command(BaseServerCommand::INVALID_COMMAND),
00195         params(nullptr),
00196         params_size(0)
00197     {}
00198
00199     HostClient client;
00200     BaseServerCommand command;
00201     std::unique_ptr<std::uint8_t> params;
00202     zmq::multipart_t raw_msg;
00203     size_t params_size;
00204 };
00205
00206 struct CommandReply
00207 {
00208     CommandReply():
00209         params(nullptr),
00210         params_size(0),
00211         result(BaseServerResult::COMMAND_OK),
00212         request_cmd(BaseServerCommand::INVALID_COMMAND)
00213     {}
00214
00215     BaseServerCommand request_cmd;
00216     std::unique_ptr<std::uint8_t> params;
00217     size_t params_size;
00218     BaseServerResult result;
00219 };
00220
00221 //
    ============================================================================================================================
00222
00223 }} // END NAMESPACES.
00224 //
    ============================================================================================================================
```

5.7 utils.h

```
00001
00002 //
    ============================================================================================================================
00003 #pragma once
```

```
00004 //
      ================================================================================================================
00005
00006 // C++ INCLUDES
00007 //
      ================================================================================================================
00008 #include <string>
00009 #include <map>
00010 #include <vector>
00011 #include <functional>
00012 //
      ================================================================================================================
00013
00014 // AMELAS NAMESPACES
00015 //
      ================================================================================================================
00016 namespace amelas{
00017 namespace utils{
00018 //
      ================================================================================================================
00019
00020 template<typename ClassType, typename ReturnType, typename... Args>
00021 static std::function<ReturnType(Args...)> makeCallback(ClassType* object,
00022
      ReturnType(ClassType::*memberFunction)(Args...))
00023 {
00024     return [object, memberFunction](Args... args) -> ReturnType
00025     {
00026         return (object->*memberFunction)(std::forward<Args>(args)...);
00027     };
00028 }
00029
00030 }} // END NAMESPACES.
00031 //
      ================================================================================================================
```

5.8 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

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)

5.8.1 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.

5.8.2 Macro Definition Documentation

5.8.2.1 MKGMTIME

```
#define MKGMTIME timegm
```

Definition at line 60 of file utils.h.

5.8.3 Typedef Documentation

5.8.3.1 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.

5.8.3.2 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.

5.8.4 Function Documentation

5.8.4.1 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.

5.8.4.2 currentISO8601Date()

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

Definition at line 198 of file utils.cpp.

5.8.4.3 getCurrentPID()

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

Definition at line 204 of file utils.cpp.

5.8.4.4 getHostIPsWithInterfaces()

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

Definition at line 65 of file utils.cpp.

5.8.4.5 getHostname()

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

Definition at line 127 of file utils.cpp.

5.8.4.6 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.

5.8.4.7 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.

5.8.4.8 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.

5.8.4.9 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 utils.h

Go to the documentation of this file.

```
00001
/*********************************************************************************************************************
00002  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
       *
00003  *
       *
00004  *   Copyright (C) 2023 Degoras Project Team
       *
00005  *                      < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
       *
00006  *                      < Jesús Relinque Madroñal >
       *
00007  *
       *
00008  *   This file is part of LibZMQUtils.
       *
00009  *
       *
00010  *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
       the EUPL license   *
00011  *   as soon they will be approved by the European Commission (IDABC).
       *
00012  *
       *
00013  *   This project is free software: you can redistribute it and/or modify it under the terms of the
       EUPL license as    *
00014  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
       *
00015  *
       *
00016  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
       or agreed to in *
00017  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
       without even the  *
00018  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
       check specific   *
00019  *   language governing permissions and limitations and more details.
       *
00020  *
       *
00021  *   You should use this project in compliance with the EUPL license. You should have received a copy
       of the license   *
00022  *   along with this project. If not, see the license at < https://eupl.eu/ >.
       *
00023
*********************************************************************************************************************/
00024
00033 //
      ================================================================================================================
00034 #pragma once
00035 //
      ================================================================================================================
00036
00037 // C++ INCLUDES
00038 //
      ================================================================================================================
```

```
00039 #include <algorithm>
00040 #include <string>
00041 #include <iostream>
00042 #include <map>
00043 #include <vector>
00044 #include <cstring>
00045 #include <chrono>
00046 #include <array>
00047 #include <utility>
00048 //
      ================================================================================================================
00049
00050 // ZMQUTILS INCLUDES
00051 //
      ================================================================================================================
00052 #include "LibZMQUtils/libzmqutils_global.h"
00053 //
      ================================================================================================================
00054
00055 // DEFINITIONS
00056 //
      ================================================================================================================
00057 #if defined(__MINGW32__) || defined(_MSC_VER)
00058 #define MKGMTIME _mkgmtime
00059 #else
00060 #define MKGMTIME timegm
00061 #endif
00062 //
      ================================================================================================================
00063
00064 // ZMQUTILS NAMESPACES
00065 //
      ================================================================================================================
00066 namespace zmqutils{
00067 namespace utils{
00068 //
      ================================================================================================================
00069
00070 // CONVENIENT ALIAS AND ENUMERATIONS
00071 //
      ================================================================================================================
00073 using HRTimePointStd = std::chrono::time_point<std::chrono::high_resolution_clock>;
00075 using SCTimePointStd =  std::chrono::steady_clock::time_point;
00076 //
      ================================================================================================================
00077
00078 struct LIBZMQUTILS_EXPORT NetworkAdapterInfo
00079 {
00080     std::string id;
00081     std::string name;
00082     std::string descr;
00083     std::string ip;
00084 };
00085
00097 LIBZMQUTILS_EXPORT void binarySerializeDeserialize(const void* data, size_t data_size_bytes, void*
      dest);
00098
00099 LIBZMQUTILS_EXPORT std::vector<NetworkAdapterInfo> getHostIPsWithInterfaces();
00100
00101 LIBZMQUTILS_EXPORT std::string getHostname();
00102
00103 LIBZMQUTILS_EXPORT unsigned getCurrentPID();
00104
00105 LIBZMQUTILS_EXPORT std::string timePointToString(const HRTimePointStd& tp,
00106                                                  const std::string& format = "%Y-%m-%dT%H:%M:%S",
00107                                                  bool add_ms = true, bool add_ns = false, bool utc =
      true);
00108
00109 LIBZMQUTILS_EXPORT std::string timePointToIso8601(const HRTimePointStd& tp, bool add_ms = true, bool
      add_ns = false);
00110
00111 LIBZMQUTILS_EXPORT std::string currentISO8601Date(bool add_ms = true);
00112
00113 namespace internal
00114 {
00115 template <typename T, std::size_t... Is1, std::size_t... Is2>
00116 constexpr std::array<T, sizeof...(Is1) + sizeof...(Is2)>
00117 joinArrays(const std::array<T, sizeof...(Is1)>& a1, const std::array<T, sizeof...(Is2)>& a2,
      std::index_sequence<Is1...>, std::index_sequence<Is2...>)
00118 {
00119     return { a1[Is1]..., a2[Is2]... };
00120 }
00121 }
00122
00123 template <typename T, std::size_t N1, std::size_t N2>
00124 constexpr std::array<T, N1 + N2> joinArraysConstexpr(const std::array<T, N1>& a1, const std::array<T,
      N2>& a2)
```

```
00125 {
00126     return internal::joinArrays(a1, a2, std::make_index_sequence<N1>(),
      std::make_index_sequence<N2>());
00127 }
00128
00129 }} // END NAMESPACES.
00130 //
      ============================================================================================================================
```

5.10 amelas_server.cpp

```
00001 #include "amelas_server.h"
00002
00003 // AMELAS NAMESPACES
00004 //
      ============================================================================================================================
00005 namespace amelas{
00006 //
      ============================================================================================================================
00007
00008 using common::AmelasServerCommandStr;
00009 using common::AmelasServerResultStr;
00010 using common::ControllerError;
00011 using common::AmelasServerCommand;
00012 using common::AmelasServerResult;
00013 using zmqutils::common::BaseServerCommand;
00014 using zmqutils::common::BaseServerResult;
00015 using zmqutils::common::ResultType;
00016
00017
00018 AmelasServer::AmelasServer(unsigned int port, const std::string &local_addr) :
00019     CommandServerBase(port, local_addr)
00020 {}
00021
00022 void AmelasServer::processSetHomePosition(const CommandRequest& request, CommandReply& reply)
00023 {
00024     ControllerError controller_err;
00025
00026     // Auxilar variables.
00027     double az, el;
00028     constexpr std::size_t double_sz = sizeof(double);
00029
00030     // Check the request parameters size.
00031     if (request.params_size == 0)
00032     {
00033         reply.result = BaseServerResult::EMPTY_PARAMS;
00034         return;
00035     }
00036     else if (request.params_size != double_sz*2)
00037     {
00038         reply.result = BaseServerResult::BAD_PARAMETERS;
00039         return;
00040     }
00041
00042     // Deserialize the parameters.
00043     zmqutils::utils::binarySerializeDeserialize(request.params.get(), double_sz, &az);
00044     zmqutils::utils::binarySerializeDeserialize(request.params.get() + double_sz, double_sz, &el);
00045
00046     // Generate the struct.
00047     common::AltAzPos pos = {az, el};
00048
00049     // Process the command.
00050     controller_err = this->invokeCallback<common::SetHomePositionCallback>(
00051                                     AmelasServerCommand::REQ_SET_HOME_POSITION, pos);
00052
00053     // Store the amelas error.
00054     reply.params = std::unique_ptr<std::uint8_t>(new std::uint8_t[sizeof(ResultType)]);
00055     ResultType amelas_res = static_cast<ResultType>(controller_err);
00056     zmqutils::utils::binarySerializeDeserialize(&amelas_res, sizeof(ResultType), reply.params.get());
00057     reply.params_size = sizeof(ResultType);
00058 }
00059
00060 void AmelasServer::processGetHomePosition(const CommandRequest &, CommandReply &reply)
00061 {
00062     // Auxilar variables.
00063     constexpr std::size_t res_sz = sizeof(ResultType);
00064     constexpr std::size_t double_sz = sizeof(double);
00065     ControllerError amelas_err = ControllerError::SUCCESS;
00066     common::AltAzPos pos;
00067
00068     // Process the command.
00069     amelas_err = this->invokeCallback<common::GetHomePositionCallback>(
00070                                 AmelasServerCommand::REQ_GET_HOME_POSITION, pos);
00071
00072     // Serialize parameters
```

```
00073     reply.params = std::unique_ptr<std::uint8_t>(new std::uint8_t[res_sz + 2*double_sz]);
00074     reply.params_size = res_sz + 2*double_sz;
00075
00076     zmqutils::utils::binarySerializeDeserialize(&amelas_err, res_sz, reply.params.get());
00077     zmqutils::utils::binarySerializeDeserialize(&pos.az, double_sz, reply.params.get() + res_sz);
00078     zmqutils::utils::binarySerializeDeserialize(&pos.el, double_sz, reply.params.get() + res_sz +
      double_sz);
00079
00080     // Store the server result.
00081     reply.result = BaseServerResult::COMMAND_OK;
00082
00083     std::cout « "Size of params is " « reply.params_size « std::endl;
00084 }
00085
00086 void AmelasServer::processAmelasCommand(const CommandRequest& request, CommandReply& reply)
00087 {
00088     AmelasServerCommand command = static_cast<AmelasServerCommand>(request.command);
00089
00090     if(command == AmelasServerCommand::REQ_SET_HOME_POSITION)
00091     {
00092         this->processSetHomePosition(request, reply);
00093     }
00094     else if (command == AmelasServerCommand::REQ_GET_HOME_POSITION)
00095     {
00096         this->processGetHomePosition(request, reply);
00097     }
00098     else
00099     {
00100         reply.result = BaseServerResult::NOT_IMPLEMENTED;
00101     }
00102 }
00103
00104 void AmelasServer::onCustomCommandReceived(const CommandRequest& request, CommandReply& reply)
00105 {
00106     // Get the command.
00107     AmelasServerCommand command = static_cast<AmelasServerCommand>(request.command);
00108
00109     // Get the command string.
00110     std::string cmd_str;
00111     std::uint32_t cmd_uint = static_cast<std::uint32_t>(request.command);
00112     cmd_str = (cmd_uint < AmelasServerCommandStr.size()) ? AmelasServerCommandStr[cmd_uint] : "Unknown
      command";
00113
00114     // Log the command.
00115     std::cout « std::string(80, '-') « std::endl;
00116     std::cout«"ON CUSTOM COMMAND RECEIVED: "«std::endl;
00117     std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
00118     std::cout«"Client Id: "«request.client.id«std::endl;
00119     std::cout«"Command: "«cmd_uint«" ("«cmd_str«")"«std::endl;
00120     std::cout « std::string(80, '-') « std::endl;
00121
00122     // Process the command if it is implemented.
00123     if(command == AmelasServerCommand::END_AMELAS_COMMANDS)
00124     {
00125         // Update the result.
00126         reply.result = BaseServerResult::INVALID_MSG;
00127     }
00128     else if(AmelasServer::validateAmelasCommand(command))
00129     {
00130         this->processAmelasCommand(request, reply);
00131     }
00132     else
00133     {
00134         // Call to the base function.
00135         CommandServerBase::onCustomCommandReceived(request, reply);
00136     }
00137 }
00138
00139 void AmelasServer::onServerStart()
00140 {
00141     // Ips.
00142     std::string ips;
00143
00144     // Get listen interfaces ips.
00145     for(const auto& intrfc : this->getServerAddresses())
00146     {
00147         ips.append(intrfc.ip);
00148         ips.append(" - ");
00149     }
00150     ips.pop_back();
00151     ips.pop_back();
00152
00153     // Log.
00154     std::cout « std::string(80, '-') « std::endl;
00155     std::cout«"<AMELAS SERVER>"«std::endl;
00156     std::cout«"-> ON SERVER START: "«std::endl;
00157     std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
```

```
00158     std::cout«"Addresses: "«ips«std::endl;
00159     std::cout«"Port: "«this->getServerPort()«std::endl;
00160     std::cout « std::string(80, '-') « std::endl;
00161 }
00162
00163 void AmelasServer::onServerStop()
00164 {
00165     // Log.
00166     std::cout « std::string(80, '-') « std::endl;
00167     std::cout«"<AMELAS SERVER>"«std::endl;
00168     std::cout«"-> ON SERVER CLOSE: "«std::endl;
00169     std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
00170     std::cout « std::string(80, '-') « std::endl;
00171 }
00172
00173 void AmelasServer::onWaitingCommand()
00174 {
00175     // Log.
00176     std::cout « std::string(80, '-') « std::endl;
00177     std::cout«"<AMELAS SERVER>"«std::endl;
00178     std::cout«"-> ON WAITING COMMAND: "«std::endl;
00179     std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
00180     std::cout « std::string(80, '-') « std::endl;
00181 }
00182
00183 void AmelasServer::onDeadClient(const HostClient& client)
00184 {
00185     // Log.
00186     std::cout « std::string(80, '-') « std::endl;
00187     std::cout«"<AMELAS SERVER>"«std::endl;
00188     std::cout«"-> ON DEAD CLIENT: "«std::endl;
00189     std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
00190     std::cout«"Current Clients: "«this->getConnectedClients().size()«std::endl;
00191     std::cout«"Client Id: "«client.id«std::endl;
00192     std::cout«"Client Ip: "«client.ip«std::endl;
00193     std::cout«"Client Host: "«client.hostname«std::endl;
00194     std::cout«"Client Process: "«client.pid«std::endl;
00195     std::cout « std::string(80, '-') « std::endl;
00196 }
00197
00198 void AmelasServer::onConnected(const HostClient& client)
00199 {
00200     // Log.
00201     std::cout « std::string(80, '-') « std::endl;
00202     std::cout«"<AMELAS SERVER>"«std::endl;
00203     std::cout«"-> ON CONNECTED: "«std::endl;
00204     std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
00205     std::cout«"Current Clients: "«this->getConnectedClients().size()«std::endl;
00206     std::cout«"Client Id: "«client.id«std::endl;
00207     std::cout«"Client Ip: "«client.ip«std::endl;
00208     std::cout«"Client Host: "«client.hostname«std::endl;
00209     std::cout«"Client Process: "«client.pid«std::endl;
00210     std::cout « std::string(80, '-') « std::endl;
00211 }
00212
00213 void AmelasServer::onDisconnected(const HostClient& client)
00214 {
00215     // Log.
00216     std::cout « std::string(80, '-') « std::endl;
00217     std::cout«"<AMELAS SERVER>"«std::endl;
00218     std::cout«"-> ON DISCONNECTED: "«std::endl;
00219     std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
00220     std::cout«"Current Clients: "«this->getConnectedClients().size()«std::endl;
00221     std::cout«"Client Id: "«client.id«std::endl;
00222     std::cout«"Client Ip: "«client.ip«std::endl;
00223     std::cout«"Client Host: "«client.hostname«std::endl;
00224     std::cout«"Client Process: "«client.pid«std::endl;
00225     std::cout « std::string(80, '-') « std::endl;
00226 }
00227
00228 void AmelasServer::onServerError(const zmq::error_t &error, const std::string &ext_info)
00229 {
00230     // Log.
00231     std::cout « std::string(80, '-') « std::endl;
00232     std::cout«"<AMELAS SERVER>"«std::endl;
00233     std::cout«"-> ON SERVER ERROR: "«std::endl;
00234     std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
00235     std::cout«"Code: "«error.num()«std::endl;
00236     std::cout«"Error: "«error.what()«std::endl;
00237     std::cout«"Info: "«ext_info«std::endl;
00238     std::cout « std::string(80, '-') « std::endl;
00239 }
00240
00241 void AmelasServer::onCommandReceived(const CommandRequest &cmd_req)
00242 {
00243     // Get the command string.
00244     std::string cmd_str;
```

```
00245     std::uint32_t command = static_cast<std::uint32_t>(cmd_req.command);
00246     cmd_str = (command < AmelasServerCommandStr.size()) ? AmelasServerCommandStr[command] : "Unknown
    command";
00247     // Log.
00248     std::cout « std::string(80, '-') « std::endl;
00249     std::cout«"<AMELAS SERVER>"«std::endl;
00250     std::cout«"-> ON COMMAND RECEIVED: "«std::endl;
00251     std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
00252     std::cout«"Client Id: "«cmd_req.client.id«std::endl;
00253     std::cout«"Command: "«command«" ("«cmd_str«")"«std::endl;
00254     std::cout « std::string(80, '-') « std::endl;
00255 }
00256
00257 void AmelasServer::onInvalidMsgReceived(const CommandRequest &cmd_req)
00258 {
00259     // Log.
00260     std::cout « std::string(80, '-') « std::endl;
00261     std::cout«"<AMELAS SERVER>"«std::endl;
00262     std::cout«"-> ON BAD COMMAND RECEIVED: "«std::endl;
00263     std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
00264     std::cout«"Raw Str: "«cmd_req.raw_msg.str()«std::endl;
00265     std::cout«"Client Id: "«cmd_req.client.id«std::endl;
00266     std::cout«"Client Ip: "«cmd_req.client.ip«std::endl;
00267     std::cout«"Client Host: "«cmd_req.client.hostname«std::endl;
00268     std::cout«"Client Process: "«cmd_req.client.pid«std::endl;
00269     std::cout«"Command: "«static_cast<int>(cmd_req.command)«std::endl;
00270     std::cout«"Params Size: "«cmd_req.params_size«std::endl;
00271     std::cout « std::string(80, '-') « std::endl;
00272 }
00273
00274 void AmelasServer::onSendingResponse(const CommandReply &cmd_rep)
00275 {
00276     // Log.
00277     int result = static_cast<int>(cmd_rep.result);
00278     std::cout « std::string(80, '-') « std::endl;
00279     std::cout«"<AMELAS SERVER>"«std::endl;
00280     std::cout«"-> ON SENDING RESPONSE: "«std::endl;
00281     std::cout«"Time: "«zmqutils::utils::currentISO8601Date()«std::endl;
00282     std::cout«"Result: "«result«" ("«AmelasServerResultStr[result]«")"«std::endl;
00283     std::cout«"Params Size: "«cmd_rep.params_size«std::endl;
00284     std::cout « std::string(80, '-') « std::endl;
00285 }
00286
00287 bool AmelasServer::validateAmelasCommand(AmelasServerCommand command)
00288 {
00289     // Auxiliar variables.
00290     bool result = false;
00291     zmqutils::common::CommandType cmd = static_cast<zmqutils::common::CommandType>(command);
00292     // Check if the command is within the range of implemented custom commands.
00293     if (cmd >= common::kMinCmdId && cmd <= common::kMaxCmdId)
00294         result = true;
00295     return result;
00296 }
00297
00298 } // END NAMESPACES.
00299 //
    ============================================================================================================================
00300
```

5.11 amelas_server.h

```
00001
00002 // C++ INCLUDES
00003 //
    ============================================================================================================================
00004 #include <unordered_map>
00005 #include <string>
00006 #include <any>
00007 #include <variant>
00008 //
    ============================================================================================================================
00009
00010 // ZMQUTILS INCLUDES
00011 //
    ============================================================================================================================
00012 #include <LibZMQUtils/CommandServer>
00013 #include <LibZMQUtils/Utils>
00014 //
    ============================================================================================================================
00015
00016 // PROJECT INCLUDES
00017 //
    ============================================================================================================================
00018 #include "AmelasExampleController/common.h"
00019 #include "AmelasExampleController/utils.h"
```

```
00020 #include "common.h"
00021 //
      ============================================================================================================================
00022
00023 // AMELAS NAMESPACES
00024 //
      ============================================================================================================================
00025 namespace amelas{
00026 //
      ============================================================================================================================
00027
00028 using namespace zmqutils;
00029
00030 // Example of creating a command server from the base.
00031 class AmelasServer : public CommandServerBase
00032 {
00033 public:
00034
00035     AmelasServer(unsigned port, const std::string& local_addr = "*");
00036
00037     void setCallback(common::AmelasServerCommand command, common::ControllerCallback callback)
00038     {
00039         callback_map_[command] = callback;
00040     }
00041
00042     template<typename ClassT = void, typename ReturnT = void, typename... Args>
00043     void setCallback(common::AmelasServerCommand command,
00044                      ClassT* object,
00045                      ReturnT(ClassT::*callback)(Args...))
00046     {
00047         callback_map_[command] = utils::makeCallback(object, callback);
00048     }
00049
00050 private:
00051
00052     template <typename CallbackType, typename... Args>
00053     common::ControllerError invokeCallback(common::AmelasServerCommand command, Args&&... args)
00054     {
00055         if (auto callback = std::get_if<CallbackType>(&callback_map_[command]))
00056         {
00057             return (*callback)(std::forward<Args>(args)...);
00058         }
00059         throw std::runtime_error("Invalid command or incorrect callback type");
00060     }
00061
00062     // Helper to check if the custom command is valid.
00063     static bool validateAmelasCommand(common::AmelasServerCommand command);
00064
00065     // Process the specific commands.
00066     void processAmelasCommand(const CommandRequest&, CommandReply&);
00067     void processSetHomePosition(const CommandRequest&, CommandReply&);
00068     void processGetHomePosition(const CommandRequest&, CommandReply&);
00069
00070     // Internal overrided custom command received callback.
00071     // WARNING The most important part.
00072     virtual void onCustomCommandReceived(const CommandRequest&, CommandReply&) final;
00073
00074     // Internal overrided start callback.
00075     virtual void onServerStart() final;
00076
00077     // Internal overrided close callback.
00078     virtual void onServerStop() final;
00079
00080     // Internal waiting command callback.
00081     virtual void onWaitingCommand() final;
00082
00083     // Internal dead client callback.
00084     virtual void onDeadClient(const HostClient&) final;
00085
00086     // Internal overrided connect callback.
00087     virtual void onConnected(const HostClient&) final;
00088
00089     // Internal overrided disconnect callback.
00090     virtual void onDisconnected(const HostClient&) final;
00091
00092     // Internal overrided command received callback.
00093     virtual void onCommandReceived(const CommandRequest&) final;
00094
00095     // Internal overrided bad command received callback.
00096     virtual void onInvalidMsgReceived(const CommandRequest&) final;
00097
00098     // Internal overrided sending response callback.
00099     virtual void onSendingResponse(const CommandReply&) final;
00100
00101     // Internal overrided server error callback.
00102     virtual void onServerError(const zmq::error_t&, const std::string& ext_info) final;
00103
```

```
00104     // External callbacks map.
00105     std::map<common::AmelasServerCommand, common::ControllerCallback> callback_map_;
00106 };
00107
00108 } // END NAMESPACES.
00109 //
```

5.12 ExampleZMQServerAmelas.cpp

```
00001
00002
00003 // C++ INCLUDES
00004 //
00005 #ifdef _WIN32
00006 #include <Windows.h>
00007 #endif
00008 #include <iostream>
00009 #include <chrono>
00010 #include <thread>
00011 #include <csignal>
00012 #include <limits>
00013
00014 //
00015
00016 // ZMQUTILS INCLUDES
00017 //
00018 #include <LibZMQUtils/CommandServer>
00019 #include <LibZMQUtils/Utils>
00020 //
00021
00022 // PROJECT INCLUDES
00023 //
00024 #include "AmelasExampleServer/amelas_server.h"
00025 #include "AmelasExampleController/amelas_controller.h"
00026 //
00027
00028 //
00029
00030 // Global variables for safety ending.
00031 volatile sig_atomic_t gSignInterrupt = 0;
00032 std::condition_variable gExitCv;
00033 std::mutex gMtx;
00034
00035 // Signal handler for safety ending.
00036 #ifdef _WIN32
00037 BOOL WINAPI ConsoleCtrlHandler(DWORD dwCtrlType)
00038 {
00039     std::lock_guard<std::mutex> lock(gMtx);
00040     if (dwCtrlType == CTRL_C_EVENT || dwCtrlType == CTRL_BREAK_EVENT)
00041     {
00042         if (!gSignInterrupt)
00043         {
00044             gSignInterrupt = 1;
00045             gExitCv.notify_all();
00046         }
00047         return TRUE;
00048     }
00049     return FALSE;
00050 }
00051 #else
00052 // TODO
00053 #endif
00054
00055 //
00056
00057 // Main function.
00058 //
00059 // In the main we will create an AmelasController and an AmelasServer that will
00060 // work together thanks to the callbacks. For safe finish, press ctrl-c.
00061 //
00062 int main(int argc, char**argv)
00063 {
00064     // Using.
00065     using amelas::common::AmelasServerCommand;
00066
00067     // Set up the Windows Console Control Handler
```

```
00068     SetConsoleCtrlHandler(ConsoleCtrlHandler, TRUE);
00069
00070     // Configuration variables.
00071     unsigned port = 9999;
00072     bool client_status_check = false;
00073
00074     // Get the port.
00075     if (argc == 2)
00076     {
00077         try
00078         {
00079            port = std::stoul(argv[1]);
00080         }  catch (...)
00081         {
00082             std::cerr « "Not recognized port in input: " « argv[1] « std::endl;
00083             return -1;
00084         }
00085
00086     }
00087     else if (argc > 2)
00088     {
00089         std::cout « "Usage: ZMQServer [port]" « std::endl;
00090         return 0;
00091     }
00092
00093     // Instantiate the Amelas controller.
00094     amelas::AmelasController amelas_controller;
00095
00096     // Instantiate the server.
00097     amelas::AmelasServer amelas_server(port);
00098
00099     // Disable or enables the client status checking.
00100     amelas_server.setClientStatusCheck(client_status_check);
00101
00102     // ---------------------------------------
00103     // Set the controller callbacks in the server.
00104
00105     amelas_server.setCallback(AmelasServerCommand::REQ_SET_HOME_POSITION,
00106                               &amelas_controller,
00107                               &amelas::AmelasController::setHomePosition);
00108
00109     amelas_server.setCallback(AmelasServerCommand::REQ_GET_HOME_POSITION,
00110                               &amelas_controller,
00111                               &amelas::AmelasController::getHomePosition);
00112
00113     // ---------------------------------------
00114
00115     // Start the server.
00116     amelas_server.startServer();
00117
00118     // Use the condition variable as an infinite loop until ctrl-c.
00119     std::unique_lock<std::mutex> lock(gMtx);
00120     gExitCv.wait(lock, [] { return gSignInterrupt == 1; });
00121
00122     // Stop the server and wait the future.
00123     amelas_server.stopServer();
00124
00125     // Final log.
00126     std::cout « "Server stoped. Press Enter to exit!" « std::endl;
00127     std::cin.ignore(std::numeric_limits<std::streamsize>::max(),'\n');
00128
00129     // Return.
00130     return 0;
00131 }
00132
00133 //
```

5.13 zmq.h

```
00001 /*
00002     Copyright (c) 2007-2016 Contributors as noted in the AUTHORS file
00003
00004     This file is part of libzmq, the ZeroMQ core engine in C++.
00005
00006     libzmq is free software; you can redistribute it and/or modify it under
00007     the terms of the GNU Lesser General Public License (LGPL) as published
00008     by the Free Software Foundation; either version 3 of the License, or
00009     (at your option) any later version.
00010
00011     As a special exception, the Contributors give you permission to link
00012     this library with independent modules to produce an executable,
00013     regardless of the license terms of these independent modules, and to
00014     copy and distribute the resulting executable under terms of your choice,
00015     provided that you also meet, for each linked independent module, the
```

```
00016     terms and conditions of the license of that module. An independent
00017     module is a module which is not derived from or based on this library.
00018     If you modify this library, you must extend this exception to your
00019     version of the library.
00020
00021     libzmq is distributed in the hope that it will be useful, but WITHOUT
00022     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
00023     FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
00024     License for more details.
00025
00026     You should have received a copy of the GNU Lesser General Public License
00027     along with this program.  If not, see <http://www.gnu.org/licenses/>.
00028
00029     ************************************************************************
00030     NOTE to contributors. This file comprises the principal public contract
00031     for ZeroMQ API users. Any change to this file supplied in a stable
00032     release SHOULD not break existing applications.
00033     In practice this means that the value of constants must not change, and
00034     that old values may not be reused for new constants.
00035     ************************************************************************
00036 */
00037
00038 #ifndef __ZMQ_H_INCLUDED__
00039 #define __ZMQ_H_INCLUDED__
00040
00041 /*  Version macros for compile-time API version detection                     */
00042 #define ZMQ_VERSION_MAJOR 4
00043 #define ZMQ_VERSION_MINOR 3
00044 #define ZMQ_VERSION_PATCH 4
00045
00046 #define ZMQ_MAKE_VERSION(major, minor, patch)                                \
00047     ((major) *10000 + (minor) *100 + (patch))
00048 #define ZMQ_VERSION                                                          \
00049     ZMQ_MAKE_VERSION (ZMQ_VERSION_MAJOR, ZMQ_VERSION_MINOR, ZMQ_VERSION_PATCH)
00050
00051 #ifdef __cplusplus
00052 extern "C" {
00053 #endif
00054
00055 #if !defined _WIN32_WCE
00056 #include <errno.h>
00057 #endif
00058 #include <stddef.h>
00059 #include <stdio.h>
00060 #if defined _WIN32
00061 //  Set target version to Windows Server 2008, Windows Vista or higher.
00062 //  Windows XP (0x0501) is supported but without client & server socket types.
00063 #ifndef _WIN32_WINNT
00064 #define _WIN32_WINNT 0x0600
00065 #endif
00066
00067 #ifdef __MINGW32__
00068 //  Require Windows XP or higher with MinGW for getaddrinfo().
00069 #if (_WIN32_WINNT >= 0x0501)
00070 #else
00071 #error You need at least Windows XP target
00072 #endif
00073 #endif
00074 #endif
00075
00076 /*  Handle DSO symbol visibility                                             */
00077 #if defined _WIN32
00078 #if defined ZMQ_STATIC
00079 #define ZMQ_EXPORT
00080 #elif defined DLL_EXPORT
00081 #define ZMQ_EXPORT __declspec(dllexport)
00082 #else
00083 #define ZMQ_EXPORT __declspec(dllimport)
00084 #endif
00085 #else
00086 #if defined __SUNPRO_C || defined __SUNPRO_CC
00087 #define ZMQ_EXPORT __global
00088 #elif (defined __GNUC__ && __GNUC__ >= 4) || defined __INTEL_COMPILER
00089 #define ZMQ_EXPORT __attribute__ ((visibility ("default")))
00090 #else
00091 #define ZMQ_EXPORT
00092 #endif
00093 #endif
00094
00095 /*  Define integer types needed for event interface                          */
00096 #define ZMQ_DEFINED_STDINT 1
00097 #if defined ZMQ_HAVE_SOLARIS || defined ZMQ_HAVE_OPENVMS
00098 #include <inttypes.h>
00099 #elif defined _MSC_VER && _MSC_VER < 1600
00100 #ifndef uint64_t
00101 typedef unsigned __int64 uint64_t;
00102 #endif
```

```
00103 #ifndef int32_t
00104 typedef __int32 int32_t;
00105 #endif
00106 #ifndef uint32_t
00107 typedef unsigned __int32 uint32_t;
00108 #endif
00109 #ifndef uint16_t
00110 typedef unsigned __int16 uint16_t;
00111 #endif
00112 #ifndef uint8_t
00113 typedef unsigned __int8 uint8_t;
00114 #endif
00115 #else
00116 #include <stdint.h>
00117 #endif
00118
00119 //  32-bit AIX's pollfd struct members are called reqevents and rtnevents so it
00120 //  defines compatibility macros for them. Need to include that header first to
00121 //  stop build failures since zmq_pollset_t defines them as events and revents.
00122 #ifdef ZMQ_HAVE_AIX
00123 #include <poll.h>
00124 #endif
00125
00126
00127 /******************************************************************************/
00128 /*  0MQ errors.                                                               */
00129 /******************************************************************************/
00130
00131 /*  A number random enough not to collide with different errno ranges on      */
00132 /*  different OSes. The assumption is that error_t is at least 32-bit type.   */
00133 #define ZMQ_HAUSNUMERO 156384712
00134
00135 /*  On Windows platform some of the standard POSIX errnos are not defined.    */
00136 #ifndef ENOTSUP
00137 #define ENOTSUP (ZMQ_HAUSNUMERO + 1)
00138 #endif
00139 #ifndef EPROTONOSUPPORT
00140 #define EPROTONOSUPPORT (ZMQ_HAUSNUMERO + 2)
00141 #endif
00142 #ifndef ENOBUFS
00143 #define ENOBUFS (ZMQ_HAUSNUMERO + 3)
00144 #endif
00145 #ifndef ENETDOWN
00146 #define ENETDOWN (ZMQ_HAUSNUMERO + 4)
00147 #endif
00148 #ifndef EADDRINUSE
00149 #define EADDRINUSE (ZMQ_HAUSNUMERO + 5)
00150 #endif
00151 #ifndef EADDRNOTAVAIL
00152 #define EADDRNOTAVAIL (ZMQ_HAUSNUMERO + 6)
00153 #endif
00154 #ifndef ECONNREFUSED
00155 #define ECONNREFUSED (ZMQ_HAUSNUMERO + 7)
00156 #endif
00157 #ifndef EINPROGRESS
00158 #define EINPROGRESS (ZMQ_HAUSNUMERO + 8)
00159 #endif
00160 #ifndef ENOTSOCK
00161 #define ENOTSOCK (ZMQ_HAUSNUMERO + 9)
00162 #endif
00163 #ifndef EMSGSIZE
00164 #define EMSGSIZE (ZMQ_HAUSNUMERO + 10)
00165 #endif
00166 #ifndef EAFNOSUPPORT
00167 #define EAFNOSUPPORT (ZMQ_HAUSNUMERO + 11)
00168 #endif
00169 #ifndef ENETUNREACH
00170 #define ENETUNREACH (ZMQ_HAUSNUMERO + 12)
00171 #endif
00172 #ifndef ECONNABORTED
00173 #define ECONNABORTED (ZMQ_HAUSNUMERO + 13)
00174 #endif
00175 #ifndef ECONNRESET
00176 #define ECONNRESET (ZMQ_HAUSNUMERO + 14)
00177 #endif
00178 #ifndef ENOTCONN
00179 #define ENOTCONN (ZMQ_HAUSNUMERO + 15)
00180 #endif
00181 #ifndef ETIMEDOUT
00182 #define ETIMEDOUT (ZMQ_HAUSNUMERO + 16)
00183 #endif
00184 #ifndef EHOSTUNREACH
00185 #define EHOSTUNREACH (ZMQ_HAUSNUMERO + 17)
00186 #endif
00187 #ifndef ENETRESET
00188 #define ENETRESET (ZMQ_HAUSNUMERO + 18)
00189 #endif
```

```
00190
00191 /*  Native 0MQ error codes.                                             */
00192 #define EFSM (ZMQ_HAUSNUMERO + 51)
00193 #define ENOCOMPATPROTO (ZMQ_HAUSNUMERO + 52)
00194 #define ETERM (ZMQ_HAUSNUMERO + 53)
00195 #define EMTHREAD (ZMQ_HAUSNUMERO + 54)
00196
00197 /*  This function retrieves the errno as it is known to 0MQ library. The goal */
00198 /*  of this function is to make the code 100% portable, including where 0MQ   */
00199 /*  compiled with certain CRT library (on Windows) is linked to an            */
00200 /*  application that uses different CRT library.                              */
00201 ZMQ_EXPORT int zmq_errno (void);
00202
00203 /*  Resolves system errors and 0MQ errors to human-readable string.           */
00204 ZMQ_EXPORT const char *zmq_strerror (int errnum_);
00205
00206 /*  Run-time API version detection                                            */
00207 ZMQ_EXPORT void zmq_version (int *major_, int *minor_, int *patch_);
00208
00209 /******************************************************************************/
00210 /*  0MQ infrastructure (a.k.a. context) initialisation & termination.         */
00211 /******************************************************************************/
00212
00213 /*  Context options                                                           */
00214 #define ZMQ_IO_THREADS 1
00215 #define ZMQ_MAX_SOCKETS 2
00216 #define ZMQ_SOCKET_LIMIT 3
00217 #define ZMQ_THREAD_PRIORITY 3
00218 #define ZMQ_THREAD_SCHED_POLICY 4
00219 #define ZMQ_MAX_MSGSZ 5
00220 #define ZMQ_MSG_T_SIZE 6
00221 #define ZMQ_THREAD_AFFINITY_CPU_ADD 7
00222 #define ZMQ_THREAD_AFFINITY_CPU_REMOVE 8
00223 #define ZMQ_THREAD_NAME_PREFIX 9
00224
00225 /*  Default for new contexts                                                  */
00226 #define ZMQ_IO_THREADS_DFLT 1
00227 #define ZMQ_MAX_SOCKETS_DFLT 1023
00228 #define ZMQ_THREAD_PRIORITY_DFLT -1
00229 #define ZMQ_THREAD_SCHED_POLICY_DFLT -1
00230
00231 ZMQ_EXPORT void *zmq_ctx_new (void);
00232 ZMQ_EXPORT int zmq_ctx_term (void *context_);
00233 ZMQ_EXPORT int zmq_ctx_shutdown (void *context_);
00234 ZMQ_EXPORT int zmq_ctx_set (void *context_, int option_, int optval_);
00235 ZMQ_EXPORT int zmq_ctx_get (void *context_, int option_);
00236
00237 /*  Old (legacy) API                                                          */
00238 ZMQ_EXPORT void *zmq_init (int io_threads_);
00239 ZMQ_EXPORT int zmq_term (void *context_);
00240 ZMQ_EXPORT int zmq_ctx_destroy (void *context_);
00241
00242
00243 /******************************************************************************/
00244 /*  0MQ message definition.                                                   */
00245 /******************************************************************************/
00246
00247 /* Some architectures, like sparc64 and some variants of aarch64, enforce pointer
00248  * alignment and raise sigbus on violations. Make sure applications allocate
00249  * zmq_msg_t on addresses aligned on a pointer-size boundary to avoid this issue.
00250  */
00251 typedef struct zmq_msg_t
00252 {
00253 #if defined(_MSC_VER) && (defined(_M_X64) || defined(_M_ARM64))
00254     __declspec(align (8)) unsigned char _[64];
00255 #elif defined(_MSC_VER)                                                    \
00256   && (defined(_M_IX86) || defined(_M_ARM_ARMV7VE) || defined(_M_ARM))
00257     __declspec(align (4)) unsigned char _[64];
00258 #elif defined(__GNUC__) || defined(__INTEL_COMPILER)                      \
00259   || (defined(__SUNPRO_C) && __SUNPRO_C >= 0x590)                          \
00260   || (defined(__SUNPRO_CC) && __SUNPRO_CC >= 0x590)
00261     unsigned char _[64] __attribute__ ((aligned (sizeof (void *))));
00262 #else
00263     unsigned char _[64];
00264 #endif
00265 } zmq_msg_t;
00266
00267 typedef void(zmq_free_fn) (void *data_, void *hint_);
00268
00269 ZMQ_EXPORT int zmq_msg_init (zmq_msg_t *msg_);
00270 ZMQ_EXPORT int zmq_msg_init_size (zmq_msg_t *msg_, size_t size_);
00271 ZMQ_EXPORT int zmq_msg_init_data (
00272   zmq_msg_t *msg_, void *data_, size_t size_, zmq_free_fn *ffn_, void *hint_);
00273 ZMQ_EXPORT int zmq_msg_send (zmq_msg_t *msg_, void *s_, int flags_);
00274 ZMQ_EXPORT int zmq_msg_recv (zmq_msg_t *msg_, void *s_, int flags_);
00275 ZMQ_EXPORT int zmq_msg_close (zmq_msg_t *msg_);
00276 ZMQ_EXPORT int zmq_msg_move (zmq_msg_t *dest_, zmq_msg_t *src_);
```

```
00277 ZMQ_EXPORT int zmq_msg_copy (zmq_msg_t *dest_, zmq_msg_t *src_);
00278 ZMQ_EXPORT void *zmq_msg_data (zmq_msg_t *msg_);
00279 ZMQ_EXPORT size_t zmq_msg_size (const zmq_msg_t *msg_);
00280 ZMQ_EXPORT int zmq_msg_more (const zmq_msg_t *msg_);
00281 ZMQ_EXPORT int zmq_msg_get (const zmq_msg_t *msg_, int property_);
00282 ZMQ_EXPORT int zmq_msg_set (zmq_msg_t *msg_, int property_, int optval_);
00283 ZMQ_EXPORT const char *zmq_msg_gets (const zmq_msg_t *msg_,
00284                                      const char *property_);
00285
00286 /******************************************************************************/
00287 /*  0MQ socket definition.                                                    */
00288 /******************************************************************************/
00289
00290 /*  Socket types.                                                             */
00291 #define ZMQ_PAIR 0
00292 #define ZMQ_PUB 1
00293 #define ZMQ_SUB 2
00294 #define ZMQ_REQ 3
00295 #define ZMQ_REP 4
00296 #define ZMQ_DEALER 5
00297 #define ZMQ_ROUTER 6
00298 #define ZMQ_PULL 7
00299 #define ZMQ_PUSH 8
00300 #define ZMQ_XPUB 9
00301 #define ZMQ_XSUB 10
00302 #define ZMQ_STREAM 11
00303
00304 /*  Deprecated aliases                                                        */
00305 #define ZMQ_XREQ ZMQ_DEALER
00306 #define ZMQ_XREP ZMQ_ROUTER
00307
00308 /*  Socket options.                                                           */
00309 #define ZMQ_AFFINITY 4
00310 #define ZMQ_ROUTING_ID 5
00311 #define ZMQ_SUBSCRIBE 6
00312 #define ZMQ_UNSUBSCRIBE 7
00313 #define ZMQ_RATE 8
00314 #define ZMQ_RECOVERY_IVL 9
00315 #define ZMQ_SNDBUF 11
00316 #define ZMQ_RCVBUF 12
00317 #define ZMQ_RCVMORE 13
00318 #define ZMQ_FD 14
00319 #define ZMQ_EVENTS 15
00320 #define ZMQ_TYPE 16
00321 #define ZMQ_LINGER 17
00322 #define ZMQ_RECONNECT_IVL 18
00323 #define ZMQ_BACKLOG 19
00324 #define ZMQ_RECONNECT_IVL_MAX 21
00325 #define ZMQ_MAXMSGSIZE 22
00326 #define ZMQ_SNDHWM 23
00327 #define ZMQ_RCVHWM 24
00328 #define ZMQ_MULTICAST_HOPS 25
00329 #define ZMQ_RCVTIMEO 27
00330 #define ZMQ_SNDTIMEO 28
00331 #define ZMQ_LAST_ENDPOINT 32
00332 #define ZMQ_ROUTER_MANDATORY 33
00333 #define ZMQ_TCP_KEEPALIVE 34
00334 #define ZMQ_TCP_KEEPALIVE_CNT 35
00335 #define ZMQ_TCP_KEEPALIVE_IDLE 36
00336 #define ZMQ_TCP_KEEPALIVE_INTVL 37
00337 #define ZMQ_IMMEDIATE 39
00338 #define ZMQ_XPUB_VERBOSE 40
00339 #define ZMQ_ROUTER_RAW 41
00340 #define ZMQ_IPV6 42
00341 #define ZMQ_MECHANISM 43
00342 #define ZMQ_PLAIN_SERVER 44
00343 #define ZMQ_PLAIN_USERNAME 45
00344 #define ZMQ_PLAIN_PASSWORD 46
00345 #define ZMQ_CURVE_SERVER 47
00346 #define ZMQ_CURVE_PUBLICKEY 48
00347 #define ZMQ_CURVE_SECRETKEY 49
00348 #define ZMQ_CURVE_SERVERKEY 50
00349 #define ZMQ_PROBE_ROUTER 51
00350 #define ZMQ_REQ_CORRELATE 52
00351 #define ZMQ_REQ_RELAXED 53
00352 #define ZMQ_CONFLATE 54
00353 #define ZMQ_ZAP_DOMAIN 55
00354 #define ZMQ_ROUTER_HANDOVER 56
00355 #define ZMQ_TOS 57
00356 #define ZMQ_CONNECT_ROUTING_ID 61
00357 #define ZMQ_GSSAPI_SERVER 62
00358 #define ZMQ_GSSAPI_PRINCIPAL 63
00359 #define ZMQ_GSSAPI_SERVICE_PRINCIPAL 64
00360 #define ZMQ_GSSAPI_PLAINTEXT 65
00361 #define ZMQ_HANDSHAKE_IVL 66
00362 #define ZMQ_SOCKS_PROXY 68
00363 #define ZMQ_XPUB_NODROP 69
```

```
00364 #define ZMQ_BLOCKY 70
00365 #define ZMQ_XPUB_MANUAL 71
00366 #define ZMQ_XPUB_WELCOME_MSG 72
00367 #define ZMQ_STREAM_NOTIFY 73
00368 #define ZMQ_INVERT_MATCHING 74
00369 #define ZMQ_HEARTBEAT_IVL 75
00370 #define ZMQ_HEARTBEAT_TTL 76
00371 #define ZMQ_HEARTBEAT_TIMEOUT 77
00372 #define ZMQ_XPUB_VERBOSER 78
00373 #define ZMQ_CONNECT_TIMEOUT 79
00374 #define ZMQ_TCP_MAXRT 80
00375 #define ZMQ_THREAD_SAFE 81
00376 #define ZMQ_MULTICAST_MAXTPDU 84
00377 #define ZMQ_VMCI_BUFFER_SIZE 85
00378 #define ZMQ_VMCI_BUFFER_MIN_SIZE 86
00379 #define ZMQ_VMCI_BUFFER_MAX_SIZE 87
00380 #define ZMQ_VMCI_CONNECT_TIMEOUT 88
00381 #define ZMQ_USE_FD 89
00382 #define ZMQ_GSSAPI_PRINCIPAL_NAMETYPE 90
00383 #define ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAMETYPE 91
00384 #define ZMQ_BINDTODEVICE 92
00385
00386 /*  Message options                                                 */
00387 #define ZMQ_MORE 1
00388 #define ZMQ_SHARED 3
00389
00390 /*  Send/recv options.                                              */
00391 #define ZMQ_DONTWAIT 1
00392 #define ZMQ_SNDMORE 2
00393
00394 /*  Security mechanisms                                             */
00395 #define ZMQ_NULL 0
00396 #define ZMQ_PLAIN 1
00397 #define ZMQ_CURVE 2
00398 #define ZMQ_GSSAPI 3
00399
00400 /*  RADIO-DISH protocol                                             */
00401 #define ZMQ_GROUP_MAX_LENGTH 255
00402
00403 /*  Deprecated options and aliases                                  */
00404 #define ZMQ_IDENTITY ZMQ_ROUTING_ID
00405 #define ZMQ_CONNECT_RID ZMQ_CONNECT_ROUTING_ID
00406 #define ZMQ_TCP_ACCEPT_FILTER 38
00407 #define ZMQ_IPC_FILTER_PID 58
00408 #define ZMQ_IPC_FILTER_UID 59
00409 #define ZMQ_IPC_FILTER_GID 60
00410 #define ZMQ_IPV4ONLY 31
00411 #define ZMQ_DELAY_ATTACH_ON_CONNECT ZMQ_IMMEDIATE
00412 #define ZMQ_NOBLOCK ZMQ_DONTWAIT
00413 #define ZMQ_FAIL_UNROUTABLE ZMQ_ROUTER_MANDATORY
00414 #define ZMQ_ROUTER_BEHAVIOR ZMQ_ROUTER_MANDATORY
00415
00416 /*  Deprecated Message options                                      */
00417 #define ZMQ_SRCFD 2
00418
00419 /******************************************************************************/
00420 /*  GSSAPI definitions                                              */
00421 /******************************************************************************/
00422
00423 /*  GSSAPI principal name types                                     */
00424 #define ZMQ_GSSAPI_NT_HOSTBASED 0
00425 #define ZMQ_GSSAPI_NT_USER_NAME 1
00426 #define ZMQ_GSSAPI_NT_KRB5_PRINCIPAL 2
00427
00428 /******************************************************************************/
00429 /*  0MQ socket events and monitoring                                */
00430 /******************************************************************************/
00431
00432 /*  Socket transport events (TCP, IPC and TIPC only)                */
00433
00434 #define ZMQ_EVENT_CONNECTED 0x0001
00435 #define ZMQ_EVENT_CONNECT_DELAYED 0x0002
00436 #define ZMQ_EVENT_CONNECT_RETRIED 0x0004
00437 #define ZMQ_EVENT_LISTENING 0x0008
00438 #define ZMQ_EVENT_BIND_FAILED 0x0010
00439 #define ZMQ_EVENT_ACCEPTED 0x0020
00440 #define ZMQ_EVENT_ACCEPT_FAILED 0x0040
00441 #define ZMQ_EVENT_CLOSED 0x0080
00442 #define ZMQ_EVENT_CLOSE_FAILED 0x0100
00443 #define ZMQ_EVENT_DISCONNECTED 0x0200
00444 #define ZMQ_EVENT_MONITOR_STOPPED 0x0400
00445 #define ZMQ_EVENT_ALL 0xFFFF
00446 /*  Unspecified system errors during handshake. Event value is an errno.      */
00447 #define ZMQ_EVENT_HANDSHAKE_FAILED_NO_DETAIL 0x0800
00448 /*  Handshake complete successfully with successful authentication (if        *
00449  *  enabled). Event value is unused.                                */
00450 #define ZMQ_EVENT_HANDSHAKE_SUCCEEDED 0x1000
```

```
00451 /*  Protocol errors between ZMTP peers or between server and ZAP handler.     *
00452  *  Event value is one of ZMQ_PROTOCOL_ERROR_*                                */
00453 #define ZMQ_EVENT_HANDSHAKE_FAILED_PROTOCOL 0x2000
00454 /*  Failed authentication requests. Event value is the numeric ZAP status     *
00455  *  code, i.e. 300, 400 or 500.                                               */
00456 #define ZMQ_EVENT_HANDSHAKE_FAILED_AUTH 0x4000
00457 #define ZMQ_PROTOCOL_ERROR_ZMTP_UNSPECIFIED 0x10000000
00458 #define ZMQ_PROTOCOL_ERROR_ZMTP_UNEXPECTED_COMMAND 0x10000001
00459 #define ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_SEQUENCE 0x10000002
00460 #define ZMQ_PROTOCOL_ERROR_ZMTP_KEY_EXCHANGE 0x10000003
00461 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_UNSPECIFIED 0x10000011
00462 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_MESSAGE 0x10000012
00463 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_HELLO 0x10000013
00464 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_INITIATE 0x10000014
00465 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_ERROR 0x10000015
00466 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_READY 0x10000016
00467 #define ZMQ_PROTOCOL_ERROR_ZMTP_MALFORMED_COMMAND_WELCOME 0x10000017
00468 #define ZMQ_PROTOCOL_ERROR_ZMTP_INVALID_METADATA 0x10000018
00469 // the following two may be due to erroneous configuration of a peer
00470 #define ZMQ_PROTOCOL_ERROR_ZMTP_CRYPTOGRAPHIC 0x11000001
00471 #define ZMQ_PROTOCOL_ERROR_ZMTP_MECHANISM_MISMATCH 0x11000002
00472 #define ZMQ_PROTOCOL_ERROR_ZAP_UNSPECIFIED 0x20000000
00473 #define ZMQ_PROTOCOL_ERROR_ZAP_MALFORMED_REPLY 0x20000001
00474 #define ZMQ_PROTOCOL_ERROR_ZAP_BAD_REQUEST_ID 0x20000002
00475 #define ZMQ_PROTOCOL_ERROR_ZAP_BAD_VERSION 0x20000003
00476 #define ZMQ_PROTOCOL_ERROR_ZAP_INVALID_STATUS_CODE 0x20000004
00477 #define ZMQ_PROTOCOL_ERROR_ZAP_INVALID_METADATA 0x20000005
00478 #define ZMQ_PROTOCOL_ERROR_WS_UNSPECIFIED 0x30000000
00479
00480 ZMQ_EXPORT void *zmq_socket (void *, int type_);
00481 ZMQ_EXPORT int zmq_close (void *s_);
00482 ZMQ_EXPORT int
00483 zmq_setsockopt (void *s_, int option_, const void *optval_, size_t optvallen_);
00484 ZMQ_EXPORT int
00485 zmq_getsockopt (void *s_, int option_, void *optval_, size_t *optvallen_);
00486 ZMQ_EXPORT int zmq_bind (void *s_, const char *addr_);
00487 ZMQ_EXPORT int zmq_connect (void *s_, const char *addr_);
00488 ZMQ_EXPORT int zmq_unbind (void *s_, const char *addr_);
00489 ZMQ_EXPORT int zmq_disconnect (void *s_, const char *addr_);
00490 ZMQ_EXPORT int zmq_send (void *s_, const void *buf_, size_t len_, int flags_);
00491 ZMQ_EXPORT int
00492 zmq_send_const (void *s_, const void *buf_, size_t len_, int flags_);
00493 ZMQ_EXPORT int zmq_recv (void *s_, void *buf_, size_t len_, int flags_);
00494 ZMQ_EXPORT int zmq_socket_monitor (void *s_, const char *addr_, int events_);
00495
00496 /******************************************************************************/
00497 /*  Hide socket fd type; this was before zmq_poller_event_t typedef below     */
00498 /******************************************************************************/
00499
00500 #if defined _WIN32
00501 // Windows uses a pointer-sized unsigned integer to store the socket fd.
00502 #if defined _WIN64
00503 typedef unsigned __int64 zmq_fd_t;
00504 #else
00505 typedef unsigned int zmq_fd_t;
00506 #endif
00507 #else
00508 typedef int zmq_fd_t;
00509 #endif
00510
00511 /******************************************************************************/
00512 /*  Deprecated I/O multiplexing. Prefer using zmq_poller API                  */
00513 /******************************************************************************/
00514
00515 #define ZMQ_POLLIN 1
00516 #define ZMQ_POLLOUT 2
00517 #define ZMQ_POLLERR 4
00518 #define ZMQ_POLLPRI 8
00519
00520 typedef struct zmq_pollitem_t
00521 {
00522     void *socket;
00523     zmq_fd_t fd;
00524     short events;
00525     short revents;
00526 } zmq_pollitem_t;
00527
00528 #define ZMQ_POLLITEMS_DFLT 16
00529
00530 ZMQ_EXPORT int zmq_poll (zmq_pollitem_t *items_, int nitems_, long timeout_);
00531
00532 /******************************************************************************/
00533 /*  Message proxying                                                          */
00534 /******************************************************************************/
00535
00536 ZMQ_EXPORT int zmq_proxy (void *frontend_, void *backend_, void *capture_);
00537 ZMQ_EXPORT int zmq_proxy_steerable (void *frontend_,
```

```
00538                                   void *backend_,
00539                                   void *capture_,
00540                                   void *control_);
00541
00542 /******************************************************************************/
00543 /*  Probe library capabilities                                                */
00544 /******************************************************************************/
00545
00546 #define ZMQ_HAS_CAPABILITIES 1
00547 ZMQ_EXPORT int zmq_has (const char *capability_);
00548
00549 /*  Deprecated aliases */
00550 #define ZMQ_STREAMER 1
00551 #define ZMQ_FORWARDER 2
00552 #define ZMQ_QUEUE 3
00553
00554 /*  Deprecated methods */
00555 ZMQ_EXPORT int zmq_device (int type_, void *frontend_, void *backend_);
00556 ZMQ_EXPORT int zmq_sendmsg (void *s_, zmq_msg_t *msg_, int flags_);
00557 ZMQ_EXPORT int zmq_recvmsg (void *s_, zmq_msg_t *msg_, int flags_);
00558 struct iovec;
00559 ZMQ_EXPORT int
00560 zmq_sendiov (void *s_, struct iovec *iov_, size_t count_, int flags_);
00561 ZMQ_EXPORT int
00562 zmq_recviov (void *s_, struct iovec *iov_, size_t *count_, int flags_);
00563
00564 /******************************************************************************/
00565 /*  Encryption functions                                                      */
00566 /******************************************************************************/
00567
00568 /*  Encode data with Z85 encoding. Returns encoded data                       */
00569 ZMQ_EXPORT char *
00570 zmq_z85_encode (char *dest_, const uint8_t *data_, size_t size_);
00571
00572 /*  Decode data with Z85 encoding. Returns decoded data                       */
00573 ZMQ_EXPORT uint8_t *zmq_z85_decode (uint8_t *dest_, const char *string_);
00574
00575 /*  Generate z85-encoded public and private keypair with tweetnacl/libsodium. */
00576 /*  Returns 0 on success.                                                     */
00577 ZMQ_EXPORT int zmq_curve_keypair (char *z85_public_key_, char *z85_secret_key_);
00578
00579 /*  Derive the z85-encoded public key from the z85-encoded secret key.        */
00580 /*  Returns 0 on success.                                                     */
00581 ZMQ_EXPORT int zmq_curve_public (char *z85_public_key_,
00582                                  const char *z85_secret_key_);
00583
00584 /******************************************************************************/
00585 /*  Atomic utility methods                                                    */
00586 /******************************************************************************/
00587
00588 ZMQ_EXPORT void *zmq_atomic_counter_new (void);
00589 ZMQ_EXPORT void zmq_atomic_counter_set (void *counter_, int value_);
00590 ZMQ_EXPORT int zmq_atomic_counter_inc (void *counter_);
00591 ZMQ_EXPORT int zmq_atomic_counter_dec (void *counter_);
00592 ZMQ_EXPORT int zmq_atomic_counter_value (void *counter_);
00593 ZMQ_EXPORT void zmq_atomic_counter_destroy (void **counter_p_);
00594
00595 /******************************************************************************/
00596 /*  Scheduling timers                                                         */
00597 /******************************************************************************/
00598
00599 #define ZMQ_HAVE_TIMERS
00600
00601 typedef void(zmq_timer_fn) (int timer_id, void *arg);
00602
00603 ZMQ_EXPORT void *zmq_timers_new (void);
00604 ZMQ_EXPORT int zmq_timers_destroy (void **timers_p);
00605 ZMQ_EXPORT int
00606 zmq_timers_add (void *timers, size_t interval, zmq_timer_fn handler, void *arg);
00607 ZMQ_EXPORT int zmq_timers_cancel (void *timers, int timer_id);
00608 ZMQ_EXPORT int
00609 zmq_timers_set_interval (void *timers, int timer_id, size_t interval);
00610 ZMQ_EXPORT int zmq_timers_reset (void *timers, int timer_id);
00611 ZMQ_EXPORT long zmq_timers_timeout (void *timers);
00612 ZMQ_EXPORT int zmq_timers_execute (void *timers);
00613
00614
00615 /******************************************************************************/
00616 /*  These functions are not documented by man pages -- use at your own risk.  */
00617 /*  If you need these to be part of the formal ZMQ API, then (a) write a man  */
00618 /*  page, and (b) write a test case in tests.                                 */
00619 /******************************************************************************/
00620
00621 /*  Helper functions are used by perf tests so that they don't have to care   */
00622 /*  about minutiae of time-related functions on different OS platforms.       */
00623
00624 /*  Starts the stopwatch. Returns the handle to the watch.                    */
```

```
00625 ZMQ_EXPORT void *zmq_stopwatch_start (void);
00626
00627 /*  Returns the number of microseconds elapsed since the stopwatch was        */
00628 /*  started, but does not stop or deallocate the stopwatch.                   */
00629 ZMQ_EXPORT unsigned long zmq_stopwatch_intermediate (void *watch_);
00630
00631 /*  Stops the stopwatch. Returns the number of microseconds elapsed since     */
00632 /*  the stopwatch was started, and deallocates that watch.                    */
00633 ZMQ_EXPORT unsigned long zmq_stopwatch_stop (void *watch_);
00634
00635 /*  Sleeps for specified number of seconds.                                   */
00636 ZMQ_EXPORT void zmq_sleep (int seconds_);
00637
00638 typedef void(zmq_thread_fn) (void *);
00639
00640 /* Start a thread. Returns a handle to the thread.                            */
00641 ZMQ_EXPORT void *zmq_threadstart (zmq_thread_fn *func_, void *arg_);
00642
00643 /* Wait for thread to complete then free up resources.                        */
00644 ZMQ_EXPORT void zmq_threadclose (void *thread_);
00645
00646
00647 /******************************************************************************/
00648 /*  These functions are DRAFT and disabled in stable releases, and subject to */
00649 /*  change at ANY time until declared stable.                                 */
00650 /******************************************************************************/
00651
00652 #ifdef ZMQ_BUILD_DRAFT_API
00653
00654 /*  DRAFT Socket types.                                                       */
00655 #define ZMQ_SERVER 12
00656 #define ZMQ_CLIENT 13
00657 #define ZMQ_RADIO 14
00658 #define ZMQ_DISH 15
00659 #define ZMQ_GATHER 16
00660 #define ZMQ_SCATTER 17
00661 #define ZMQ_DGRAM 18
00662 #define ZMQ_PEER 19
00663 #define ZMQ_CHANNEL 20
00664
00665 /*  DRAFT Socket options.                                                     */
00666 #define ZMQ_ZAP_ENFORCE_DOMAIN 93
00667 #define ZMQ_LOOPBACK_FASTPATH 94
00668 #define ZMQ_METADATA 95
00669 #define ZMQ_MULTICAST_LOOP 96
00670 #define ZMQ_ROUTER_NOTIFY 97
00671 #define ZMQ_XPUB_MANUAL_LAST_VALUE 98
00672 #define ZMQ_SOCKS_USERNAME 99
00673 #define ZMQ_SOCKS_PASSWORD 100
00674 #define ZMQ_IN_BATCH_SIZE 101
00675 #define ZMQ_OUT_BATCH_SIZE 102
00676 #define ZMQ_WSS_KEY_PEM 103
00677 #define ZMQ_WSS_CERT_PEM 104
00678 #define ZMQ_WSS_TRUST_PEM 105
00679 #define ZMQ_WSS_HOSTNAME 106
00680 #define ZMQ_WSS_TRUST_SYSTEM 107
00681 #define ZMQ_ONLY_FIRST_SUBSCRIBE 108
00682 #define ZMQ_RECONNECT_STOP 109
00683 #define ZMQ_HELLO_MSG 110
00684 #define ZMQ_DISCONNECT_MSG 111
00685 #define ZMQ_PRIORITY 112
00686
00687 /*  DRAFT ZMQ_RECONNECT_STOP options                                          */
00688 #define ZMQ_RECONNECT_STOP_CONN_REFUSED 0x1
00689 #define ZMQ_RECONNECT_STOP_HANDSHAKE_FAILED 0x2
00690 #define ZMQ_RECONNECT_STOP_AFTER_DISCONNECT 0x3
00691
00692 /*  DRAFT Context options                                                     */
00693 #define ZMQ_ZERO_COPY_RECV 10
00694
00695 /*  DRAFT Context methods.                                                    */
00696 ZMQ_EXPORT int zmq_ctx_set_ext (void *context_,
00697                                 int option_,
00698                                 const void *optval_,
00699                                 size_t optvallen_);
00700 ZMQ_EXPORT int zmq_ctx_get_ext (void *context_,
00701                                 int option_,
00702                                 void *optval_,
00703                                 size_t *optvallen_);
00704
00705 /*  DRAFT Socket methods.                                                     */
00706 ZMQ_EXPORT int zmq_join (void *s, const char *group);
00707 ZMQ_EXPORT int zmq_leave (void *s, const char *group);
00708 ZMQ_EXPORT uint32_t zmq_connect_peer (void *s_, const char *addr_);
00709
00710 /*  DRAFT Msg methods.                                                        */
00711 ZMQ_EXPORT int zmq_msg_set_routing_id (zmq_msg_t *msg, uint32_t routing_id);
```

```
00712 ZMQ_EXPORT uint32_t zmq_msg_routing_id (zmq_msg_t *msg);
00713 ZMQ_EXPORT int zmq_msg_set_group (zmq_msg_t *msg, const char *group);
00714 ZMQ_EXPORT const char *zmq_msg_group (zmq_msg_t *msg);
00715 ZMQ_EXPORT int
00716 zmq_msg_init_buffer (zmq_msg_t *msg_, const void *buf_, size_t size_);
00717
00718 /*  DRAFT Msg property names.                                                 */
00719 #define ZMQ_MSG_PROPERTY_ROUTING_ID "Routing-Id"
00720 #define ZMQ_MSG_PROPERTY_SOCKET_TYPE "Socket-Type"
00721 #define ZMQ_MSG_PROPERTY_USER_ID "User-Id"
00722 #define ZMQ_MSG_PROPERTY_PEER_ADDRESS "Peer-Address"
00723
00724 /*  Router notify options                                                     */
00725 #define ZMQ_NOTIFY_CONNECT 1
00726 #define ZMQ_NOTIFY_DISCONNECT 2
00727
00728 /******************************************************************************/
00729 /*  Poller polling on sockets,fd and thread-safe sockets                      */
00730 /******************************************************************************/
00731
00732 #define ZMQ_HAVE_POLLER
00733
00734 typedef struct zmq_poller_event_t
00735 {
00736     void *socket;
00737     zmq_fd_t fd;
00738     void *user_data;
00739     short events;
00740 } zmq_poller_event_t;
00741
00742 ZMQ_EXPORT void *zmq_poller_new (void);
00743 ZMQ_EXPORT int zmq_poller_destroy (void **poller_p);
00744 ZMQ_EXPORT int zmq_poller_size (void *poller);
00745 ZMQ_EXPORT int
00746 zmq_poller_add (void *poller, void *socket, void *user_data, short events);
00747 ZMQ_EXPORT int zmq_poller_modify (void *poller, void *socket, short events);
00748 ZMQ_EXPORT int zmq_poller_remove (void *poller, void *socket);
00749 ZMQ_EXPORT int
00750 zmq_poller_wait (void *poller, zmq_poller_event_t *event, long timeout);
00751 ZMQ_EXPORT int zmq_poller_wait_all (void *poller,
00752                                     zmq_poller_event_t *events,
00753                                     int n_events,
00754                                     long timeout);
00755 ZMQ_EXPORT int zmq_poller_fd (void *poller, zmq_fd_t *fd);
00756
00757 ZMQ_EXPORT int
00758 zmq_poller_add_fd (void *poller, zmq_fd_t fd, void *user_data, short events);
00759 ZMQ_EXPORT int zmq_poller_modify_fd (void *poller, zmq_fd_t fd, short events);
00760 ZMQ_EXPORT int zmq_poller_remove_fd (void *poller, zmq_fd_t fd);
00761
00762 ZMQ_EXPORT int zmq_socket_get_peer_state (void *socket,
00763                                           const void *routing_id,
00764                                           size_t routing_id_size);
00765
00766 /*  DRAFT Socket monitoring events                                            */
00767 #define ZMQ_EVENT_PIPES_STATS 0x10000
00768
00769 #define ZMQ_CURRENT_EVENT_VERSION 1
00770 #define ZMQ_CURRENT_EVENT_VERSION_DRAFT 2
00771
00772 #define ZMQ_EVENT_ALL_V1 ZMQ_EVENT_ALL
00773 #define ZMQ_EVENT_ALL_V2 ZMQ_EVENT_ALL_V1 | ZMQ_EVENT_PIPES_STATS
00774
00775 ZMQ_EXPORT int zmq_socket_monitor_versioned (
00776   void *s_, const char *addr_, uint64_t events_, int event_version_, int type_);
00777 ZMQ_EXPORT int zmq_socket_monitor_pipes_stats (void *s);
00778
00779 #endif // ZMQ_BUILD_DRAFT_API
00780
00781
00782 #undef ZMQ_EXPORT
00783
00784 #ifdef __cplusplus
00785 }
00786 #endif
00787
00788 #endif
```

5.14 zmq.hpp

```
00001 /*
00002     Copyright (c) 2016-2017 ZeroMQ community
00003     Copyright (c) 2009-2011 250bpm s.r.o.
00004     Copyright (c) 2011 Botond Ballo
00005     Copyright (c) 2007-2009 iMatix Corporation
```

```
00006
00007     Permission is hereby granted, free of charge, to any person obtaining a copy
00008     of this software and associated documentation files (the "Software"), to
00009     deal in the Software without restriction, including without limitation the
00010     rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
00011     sell copies of the Software, and to permit persons to whom the Software is
00012     furnished to do so, subject to the following conditions:
00013
00014     The above copyright notice and this permission notice shall be included in
00015     all copies or substantial portions of the Software.
00016
00017     THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00018     IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00019     FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00020     AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00021     LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
00022     FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
00023     IN THE SOFTWARE.
00024 */
00025
00026 #ifndef __ZMQ_HPP_INCLUDED__
00027 #define __ZMQ_HPP_INCLUDED__
00028
00029 #ifdef _WIN32
00030 #ifndef NOMINMAX
00031 #define NOMINMAX
00032 #endif
00033 #endif
00034
00035 // included here for _HAS_CXX* macros
00036 #include "zmq.h"
00037
00038 #if defined(_MSVC_LANG)
00039 #define CPPZMQ_LANG _MSVC_LANG
00040 #else
00041 #define CPPZMQ_LANG __cplusplus
00042 #endif
00043 // overwrite if specific language macros indicate higher version
00044 #if defined(_HAS_CXX14) && _HAS_CXX14 && CPPZMQ_LANG < 201402L
00045 #undef CPPZMQ_LANG
00046 #define CPPZMQ_LANG 201402L
00047 #endif
00048 #if defined(_HAS_CXX17) && _HAS_CXX17 && CPPZMQ_LANG < 201703L
00049 #undef CPPZMQ_LANG
00050 #define CPPZMQ_LANG 201703L
00051 #endif
00052
00053 // macros defined if has a specific standard or greater
00054 #if CPPZMQ_LANG >= 201103L || (defined(_MSC_VER) && _MSC_VER >= 1900)
00055 #define ZMQ_CPP11
00056 #endif
00057 #if CPPZMQ_LANG >= 201402L
00058 #define ZMQ_CPP14
00059 #endif
00060 #if CPPZMQ_LANG >= 201703L
00061 #define ZMQ_CPP17
00062 #endif
00063
00064 #if defined(ZMQ_CPP14) && !defined(_MSC_VER)
00065 #define ZMQ_DEPRECATED(msg) [[deprecated(msg)]]
00066 #elif defined(_MSC_VER)
00067 #define ZMQ_DEPRECATED(msg) __declspec(deprecated(msg))
00068 #elif defined(__GNUC__)
00069 #define ZMQ_DEPRECATED(msg) __attribute__((deprecated(msg)))
00070 #else
00071 #define ZMQ_DEPRECATED(msg)
00072 #endif
00073
00074 #if defined(ZMQ_CPP17)
00075 #define ZMQ_NODISCARD [[nodiscard]]
00076 #else
00077 #define ZMQ_NODISCARD
00078 #endif
00079
00080 #if defined(ZMQ_CPP11)
00081 #define ZMQ_NOTHROW noexcept
00082 #define ZMQ_EXPLICIT explicit
00083 #define ZMQ_OVERRIDE override
00084 #define ZMQ_NULLPTR nullptr
00085 #define ZMQ_CONSTEXPR_FN constexpr
00086 #define ZMQ_CONSTEXPR_VAR constexpr
00087 #define ZMQ_CPP11_DEPRECATED(msg) ZMQ_DEPRECATED(msg)
00088 #else
00089 #define ZMQ_NOTHROW throw()
00090 #define ZMQ_EXPLICIT
00091 #define ZMQ_OVERRIDE
00092 #define ZMQ_NULLPTR 0
```

```
00093 #define ZMQ_CONSTEXPR_FN
00094 #define ZMQ_CONSTEXPR_VAR const
00095 #define ZMQ_CPP11_DEPRECATED(msg)
00096 #endif
00097 #if defined(ZMQ_CPP14) && (!defined(_MSC_VER) || _MSC_VER > 1900) && (!defined(__GNUC__) || __GNUC__ >
      5 || (__GNUC__ == 5 && __GNUC_MINOR__ > 3))
00098 #define ZMQ_EXTENDED_CONSTEXPR
00099 #endif
00100 #if defined(ZMQ_CPP17)
00101 #define ZMQ_INLINE_VAR inline
00102 #define ZMQ_CONSTEXPR_IF constexpr
00103 #else
00104 #define ZMQ_INLINE_VAR
00105 #define ZMQ_CONSTEXPR_IF
00106 #endif
00107
00108 #include <cassert>
00109 #include <cstring>
00110
00111 #include <algorithm>
00112 #include <exception>
00113 #include <iomanip>
00114 #include <sstream>
00115 #include <string>
00116 #include <vector>
00117 #ifdef ZMQ_CPP11
00118 #include <array>
00119 #include <chrono>
00120 #include <tuple>
00121 #include <memory>
00122 #endif
00123
00124 #if defined(__has_include) && defined(ZMQ_CPP17)
00125 #define CPPZMQ_HAS_INCLUDE_CPP17(X) __has_include(X)
00126 #else
00127 #define CPPZMQ_HAS_INCLUDE_CPP17(X) 0
00128 #endif
00129
00130 #if CPPZMQ_HAS_INCLUDE_CPP17(<optional>) && !defined(CPPZMQ_HAS_OPTIONAL)
00131 #define CPPZMQ_HAS_OPTIONAL 1
00132 #endif
00133 #ifndef CPPZMQ_HAS_OPTIONAL
00134 #define CPPZMQ_HAS_OPTIONAL 0
00135 #elif CPPZMQ_HAS_OPTIONAL
00136 #include <optional>
00137 #endif
00138
00139 #if CPPZMQ_HAS_INCLUDE_CPP17(<string_view>) && !defined(CPPZMQ_HAS_STRING_VIEW)
00140 #define CPPZMQ_HAS_STRING_VIEW 1
00141 #endif
00142 #ifndef CPPZMQ_HAS_STRING_VIEW
00143 #define CPPZMQ_HAS_STRING_VIEW 0
00144 #elif CPPZMQ_HAS_STRING_VIEW
00145 #include <string_view>
00146 #endif
00147
00148 /*  Version macros for compile-time API version detection                     */
00149 #define CPPZMQ_VERSION_MAJOR 4
00150 #define CPPZMQ_VERSION_MINOR 10
00151 #define CPPZMQ_VERSION_PATCH 0
00152
00153 #define CPPZMQ_VERSION                                                              \
00154     ZMQ_MAKE_VERSION(CPPZMQ_VERSION_MAJOR, CPPZMQ_VERSION_MINOR,                   \
00155                      CPPZMQ_VERSION_PATCH)
00156
00157 //  Detect whether the compiler supports C++11 rvalue references.
00158 #if (defined(__GNUC__) && (__GNUC__ > 4 || (__GNUC__ == 4 && __GNUC_MINOR__ > 2))   \
00159      && defined(__GXX_EXPERIMENTAL_CXX0X__))
00160 #define ZMQ_HAS_RVALUE_REFS
00161 #define ZMQ_DELETED_FUNCTION = delete
00162 #elif defined(__clang__)
00163 #if __has_feature(cxx_rvalue_references)
00164 #define ZMQ_HAS_RVALUE_REFS
00165 #endif
00166
00167 #if __has_feature(cxx_deleted_functions)
00168 #define ZMQ_DELETED_FUNCTION = delete
00169 #else
00170 #define ZMQ_DELETED_FUNCTION
00171 #endif
00172 #elif defined(_MSC_VER) && (_MSC_VER >= 1900)
00173 #define ZMQ_HAS_RVALUE_REFS
00174 #define ZMQ_DELETED_FUNCTION = delete
00175 #elif defined(_MSC_VER) && (_MSC_VER >= 1600)
00176 #define ZMQ_HAS_RVALUE_REFS
00177 #define ZMQ_DELETED_FUNCTION
00178 #else
```

```
00179 #define ZMQ_DELETED_FUNCTION
00180 #endif
00181
00182 #if defined(ZMQ_CPP11) && !defined(__llvm__) && !defined(__INTEL_COMPILER)          \
00183   && defined(__GNUC__) && __GNUC__ < 5
00184 #define ZMQ_CPP11_PARTIAL
00185 #elif defined(__GLIBCXX__) && __GLIBCXX__ < 20160805
00186 //the date here is the last date of gcc 4.9.4, which
00187 // effectively means libstdc++ from gcc 5.5 and higher won't trigger this branch
00188 #define ZMQ_CPP11_PARTIAL
00189 #endif
00190
00191 #ifdef ZMQ_CPP11
00192 #ifdef ZMQ_CPP11_PARTIAL
00193 #define ZMQ_IS_TRIVIALLY_COPYABLE(T) __has_trivial_copy(T)
00194 #else
00195 #include <type_traits>
00196 #define ZMQ_IS_TRIVIALLY_COPYABLE(T) std::is_trivially_copyable<T>::value
00197 #endif
00198 #endif
00199
00200 #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(3, 3, 0)
00201 #define ZMQ_NEW_MONITOR_EVENT_LAYOUT
00202 #endif
00203
00204 #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 1, 0)
00205 #define ZMQ_HAS_PROXY_STEERABLE
00206 /*  Socket event data  */
00207 typedef struct
00208 {
00209     uint16_t event; // id of the event as bitfield
00210     int32_t value;  // value is either error code, fd or reconnect interval
00211 } zmq_event_t;
00212 #endif
00213
00214 // Avoid using deprecated message receive function when possible
00215 #if ZMQ_VERSION < ZMQ_MAKE_VERSION(3, 2, 0)
00216 #define zmq_msg_recv(msg, socket, flags) zmq_recvmsg(socket, msg, flags)
00217 #endif
00218
00219
00220 // In order to prevent unused variable warnings when building in non-debug
00221 // mode use this macro to make assertions.
00222 #ifndef NDEBUG
00223 #define ZMQ_ASSERT(expression) assert(expression)
00224 #else
00225 #define ZMQ_ASSERT(expression) (void) (expression)
00226 #endif
00227
00228 namespace zmq
00229 {
00230 #ifdef ZMQ_CPP11
00231 namespace detail
00232 {
00233 namespace ranges
00234 {
00235 using std::begin;
00236 using std::end;
00237 template<class T> auto begin(T &&r) -> decltype(begin(std::forward<T>(r)))
00238 {
00239     return begin(std::forward<T>(r));
00240 }
00241 template<class T> auto end(T &&r) -> decltype(end(std::forward<T>(r)))
00242 {
00243     return end(std::forward<T>(r));
00244 }
00245 } // namespace ranges
00246
00247 template<class T> using void_t = void;
00248
00249 template<class Iter>
00250 using iter_value_t = typename std::iterator_traits<Iter>::value_type;
00251
00252 template<class Range>
00253 using range_iter_t = decltype(
00254   ranges::begin(std::declval<typename std::remove_reference<Range>::type &>()));
00255
00256 template<class Range> using range_value_t = iter_value_t<range_iter_t<Range»;
00257
00258 template<class T, class = void> struct is_range : std::false_type
00259 {
00260 };
00261
00262 template<class T>
00263 struct is_range<
00264   T,
00265   void_t<decltype(
```

```
00266     ranges::begin(std::declval<typename std::remove_reference<T>::type &>())
00267     == ranges::end(std::declval<typename std::remove_reference<T>::type &>()))»
00268     : std::true_type
00269 {
00270 };
00271
00272 } // namespace detail
00273 #endif
00274
00275 typedef zmq_free_fn free_fn;
00276 typedef zmq_pollitem_t pollitem_t;
00277
00278 // duplicate definition from libzmq 4.3.3
00279 #if defined _WIN32
00280 #if defined _WIN64
00281 typedef unsigned __int64 fd_t;
00282 #else
00283 typedef unsigned int fd_t;
00284 #endif
00285 #else
00286 typedef int fd_t;
00287 #endif
00288
00289 class error_t : public std::exception
00290 {
00291   public:
00292     error_t() ZMQ_NOTHROW : errnum(zmq_errno()) {}
00293     explicit error_t(int err) ZMQ_NOTHROW : errnum(err) {}
00294     virtual const char *what() const ZMQ_NOTHROW ZMQ_OVERRIDE
00295     {
00296         return zmq_strerror(errnum);
00297     }
00298     int num() const ZMQ_NOTHROW { return errnum; }
00299
00300   private:
00301     int errnum;
00302 };
00303
00304 namespace detail {
00305 inline int poll(zmq_pollitem_t *items_, size_t nitems_, long timeout_)
00306 {
00307     int rc = zmq_poll(items_, static_cast<int>(nitems_), timeout_);
00308     if (rc < 0)
00309         throw error_t();
00310     return rc;
00311 }
00312 }
00313
00314 #ifdef ZMQ_CPP11
00315 ZMQ_DEPRECATED("from 4.8.0, use poll taking std::chrono::duration instead of long")
00316 inline int poll(zmq_pollitem_t *items_, size_t nitems_, long timeout_)
00317 #else
00318 inline int poll(zmq_pollitem_t *items_, size_t nitems_, long timeout_ = -1)
00319 #endif
00320 {
00321     return detail::poll(items_, nitems_, timeout_);
00322 }
00323
00324 ZMQ_DEPRECATED("from 4.3.1, use poll taking non-const items")
00325 inline int poll(zmq_pollitem_t const *items_, size_t nitems_, long timeout_ = -1)
00326 {
00327     return detail::poll(const_cast<zmq_pollitem_t *>(items_), nitems_, timeout_);
00328 }
00329
00330 #ifdef ZMQ_CPP11
00331 ZMQ_DEPRECATED("from 4.3.1, use poll taking non-const items")
00332 inline int
00333 poll(zmq_pollitem_t const *items, size_t nitems, std::chrono::milliseconds timeout)
00334 {
00335     return detail::poll(const_cast<zmq_pollitem_t *>(items), nitems,
00336                 static_cast<long>(timeout.count()));
00337 }
00338
00339 ZMQ_DEPRECATED("from 4.3.1, use poll taking non-const items")
00340 inline int poll(std::vector<zmq_pollitem_t> const &items,
00341                 std::chrono::milliseconds timeout)
00342 {
00343     return detail::poll(const_cast<zmq_pollitem_t *>(items.data()), items.size(),
00344                 static_cast<long>(timeout.count()));
00345 }
00346
00347 ZMQ_DEPRECATED("from 4.3.1, use poll taking non-const items")
00348 inline int poll(std::vector<zmq_pollitem_t> const &items, long timeout_ = -1)
00349 {
00350     return detail::poll(const_cast<zmq_pollitem_t *>(items.data()), items.size(), timeout_);
00351 }
00352
```

```
00353 inline int
00354 poll(zmq_pollitem_t *items, size_t nitems, std::chrono::milliseconds timeout =
      std::chrono::milliseconds{-1})
00355 {
00356     return detail::poll(items, nitems, static_cast<long>(timeout.count()));
00357 }
00358
00359 inline int poll(std::vector<zmq_pollitem_t> &items,
00360                 std::chrono::milliseconds timeout = std::chrono::milliseconds{-1})
00361 {
00362     return detail::poll(items.data(), items.size(), static_cast<long>(timeout.count()));
00363 }
00364
00365 ZMQ_DEPRECATED("from 4.3.1, use poll taking std::chrono::duration instead of long")
00366 inline int poll(std::vector<zmq_pollitem_t> &items, long timeout_)
00367 {
00368     return detail::poll(items.data(), items.size(), timeout_);
00369 }
00370
00371 template<std::size_t SIZE>
00372 inline int poll(std::array<zmq_pollitem_t, SIZE> &items,
00373                 std::chrono::milliseconds timeout = std::chrono::milliseconds{-1})
00374 {
00375     return detail::poll(items.data(), items.size(), static_cast<long>(timeout.count()));
00376 }
00377 #endif
00378
00379
00380 inline void version(int *major_, int *minor_, int *patch_)
00381 {
00382     zmq_version(major_, minor_, patch_);
00383 }
00384
00385 #ifdef ZMQ_CPP11
00386 inline std::tuple<int, int, int> version()
00387 {
00388     std::tuple<int, int, int> v;
00389     zmq_version(&std::get<0>(v), &std::get<1>(v), &std::get<2>(v));
00390     return v;
00391 }
00392
00393 #if !defined(ZMQ_CPP11_PARTIAL)
00394 namespace detail
00395 {
00396 template<class T> struct is_char_type
00397 {
00398     // true if character type for string literals in C++11
00399     static constexpr bool value =
00400       std::is_same<T, char>::value || std::is_same<T, wchar_t>::value
00401       || std::is_same<T, char16_t>::value || std::is_same<T, char32_t>::value;
00402 };
00403 }
00404 #endif
00405
00406 #endif
00407
00408 class message_t
00409 {
00410   public:
00411     message_t() ZMQ_NOTHROW
00412     {
00413         int rc = zmq_msg_init(&msg);
00414         ZMQ_ASSERT(rc == 0);
00415     }
00416
00417     explicit message_t(size_t size_)
00418     {
00419         int rc = zmq_msg_init_size(&msg, size_);
00420         if (rc != 0)
00421             throw error_t();
00422     }
00423
00424     template<class ForwardIter> message_t(ForwardIter first, ForwardIter last)
00425     {
00426         typedef typename std::iterator_traits<ForwardIter>::value_type value_t;
00427
00428         assert(std::distance(first, last) >= 0);
00429         size_t const size_ =
00430           static_cast<size_t>(std::distance(first, last)) * sizeof(value_t);
00431         int const rc = zmq_msg_init_size(&msg, size_);
00432         if (rc != 0)
00433             throw error_t();
00434         std::copy(first, last, data<value_t>());
00435     }
00436
00437     message_t(const void *data_, size_t size_)
00438     {
```

```
00439         int rc = zmq_msg_init_size(&msg, size_);
00440         if (rc != 0)
00441             throw error_t();
00442         if (size_) {
00443             // this constructor allows (nullptr, 0),
00444             // memcpy with a null pointer is UB
00445             memcpy(data(), data_, size_);
00446         }
00447     }
00448
00449     message_t(void *data_, size_t size_, free_fn *ffn_, void *hint_ = ZMQ_NULLPTR)
00450     {
00451         int rc = zmq_msg_init_data(&msg, data_, size_, ffn_, hint_);
00452         if (rc != 0)
00453             throw error_t();
00454     }
00455
00456     // overload set of string-like types and generic containers
00457 #if defined(ZMQ_CPP11) && !defined(ZMQ_CPP11_PARTIAL)
00458     // NOTE this constructor will include the null terminator
00459     // when called with a string literal.
00460     // An overload taking const char* can not be added because
00461     // it would be preferred over this function and break compatiblity.
00462     template<
00463       class Char,
00464       size_t N,
00465       typename = typename std::enable_if<detail::is_char_type<Char>::value>::type>
00466     ZMQ_DEPRECATED("from 4.7.0, use constructors taking iterators, (pointer, size) "
00467                    "or strings instead")
00468     explicit message_t(const Char (&data)[N]) :
00469         message_t(detail::ranges::begin(data), detail::ranges::end(data))
00470     {
00471     }
00472
00473     template<class Range,
00474              typename = typename std::enable_if<
00475                detail::is_range<Range>::value
00476                && ZMQ_IS_TRIVIALLY_COPYABLE(detail::range_value_t<Range>)
00477                && !detail::is_char_type<detail::range_value_t<Range>>::value
00478                && !std::is_same<Range, message_t>::value>::type>
00479     explicit message_t(const Range &rng) :
00480         message_t(detail::ranges::begin(rng), detail::ranges::end(rng))
00481     {
00482     }
00483
00484     explicit message_t(const std::string &str) : message_t(str.data(), str.size()) {}
00485
00486 #if CPPZMQ_HAS_STRING_VIEW
00487     explicit message_t(std::string_view str) : message_t(str.data(), str.size()) {}
00488 #endif
00489
00490 #endif
00491
00492 #ifdef ZMQ_HAS_RVALUE_REFS
00493     message_t(message_t &&rhs) ZMQ_NOTHROW : msg(rhs.msg)
00494     {
00495         int rc = zmq_msg_init(&rhs.msg);
00496         ZMQ_ASSERT(rc == 0);
00497     }
00498
00499     message_t &operator=(message_t &&rhs) ZMQ_NOTHROW
00500     {
00501         std::swap(msg, rhs.msg);
00502         return *this;
00503     }
00504 #endif
00505
00506     ~message_t() ZMQ_NOTHROW
00507     {
00508         int rc = zmq_msg_close(&msg);
00509         ZMQ_ASSERT(rc == 0);
00510     }
00511
00512     void rebuild()
00513     {
00514         int rc = zmq_msg_close(&msg);
00515         if (rc != 0)
00516             throw error_t();
00517         rc = zmq_msg_init(&msg);
00518         ZMQ_ASSERT(rc == 0);
00519     }
00520
00521     void rebuild(size_t size_)
00522     {
00523         int rc = zmq_msg_close(&msg);
00524         if (rc != 0)
00525             throw error_t();
```

```
00526         rc = zmq_msg_init_size(&msg, size_);
00527         if (rc != 0)
00528             throw error_t();
00529     }
00530
00531     void rebuild(const void *data_, size_t size_)
00532     {
00533         int rc = zmq_msg_close(&msg);
00534         if (rc != 0)
00535             throw error_t();
00536         rc = zmq_msg_init_size(&msg, size_);
00537         if (rc != 0)
00538             throw error_t();
00539         memcpy(data(), data_, size_);
00540     }
00541
00542     void rebuild(const std::string &str)
00543     {
00544         rebuild(str.data(), str.size());
00545     }
00546
00547     void rebuild(void *data_, size_t size_, free_fn *ffn_, void *hint_ = ZMQ_NULLPTR)
00548     {
00549         int rc = zmq_msg_close(&msg);
00550         if (rc != 0)
00551             throw error_t();
00552         rc = zmq_msg_init_data(&msg, data_, size_, ffn_, hint_);
00553         if (rc != 0)
00554             throw error_t();
00555     }
00556
00557     ZMQ_DEPRECATED("from 4.3.1, use move taking non-const reference instead")
00558     void move(message_t const *msg_)
00559     {
00560         int rc = zmq_msg_move(&msg, const_cast<zmq_msg_t *>(msg_->handle()));
00561         if (rc != 0)
00562             throw error_t();
00563     }
00564
00565     void move(message_t &msg_)
00566     {
00567         int rc = zmq_msg_move(&msg, msg_.handle());
00568         if (rc != 0)
00569             throw error_t();
00570     }
00571
00572     ZMQ_DEPRECATED("from 4.3.1, use copy taking non-const reference instead")
00573     void copy(message_t const *msg_)
00574     {
00575         int rc = zmq_msg_copy(&msg, const_cast<zmq_msg_t *>(msg_->handle()));
00576         if (rc != 0)
00577             throw error_t();
00578     }
00579
00580     void copy(message_t &msg_)
00581     {
00582         int rc = zmq_msg_copy(&msg, msg_.handle());
00583         if (rc != 0)
00584             throw error_t();
00585     }
00586
00587     bool more() const ZMQ_NOTHROW
00588     {
00589         int rc = zmq_msg_more(const_cast<zmq_msg_t *>(&msg));
00590         return rc != 0;
00591     }
00592
00593     void *data() ZMQ_NOTHROW { return zmq_msg_data(&msg); }
00594
00595     const void *data() const ZMQ_NOTHROW
00596     {
00597         return zmq_msg_data(const_cast<zmq_msg_t *>(&msg));
00598     }
00599
00600     size_t size() const ZMQ_NOTHROW
00601     {
00602         return zmq_msg_size(const_cast<zmq_msg_t *>(&msg));
00603     }
00604
00605     ZMQ_NODISCARD bool empty() const ZMQ_NOTHROW { return size() == 0u; }
00606
00607     template<typename T> T *data() ZMQ_NOTHROW { return static_cast<T *>(data()); }
00608
00609     template<typename T> T const *data() const ZMQ_NOTHROW
00610     {
00611         return static_cast<T const *>(data());
00612     }
```

```
00613
00614     ZMQ_DEPRECATED("from 4.3.0, use operator== instead")
00615     bool equal(const message_t *other) const ZMQ_NOTHROW { return *this == *other; }
00616
00617     bool operator==(const message_t &other) const ZMQ_NOTHROW
00618     {
00619         const size_t my_size = size();
00620         return my_size == other.size() && 0 == memcmp(data(), other.data(), my_size);
00621     }
00622
00623     bool operator!=(const message_t &other) const ZMQ_NOTHROW
00624     {
00625         return !(*this == other);
00626     }
00627
00628 #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(3, 2, 0)
00629     int get(int property_)
00630     {
00631         int value = zmq_msg_get(&msg, property_);
00632         if (value == -1)
00633             throw error_t();
00634         return value;
00635     }
00636 #endif
00637
00638 #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 1, 0)
00639     const char *gets(const char *property_)
00640     {
00641         const char *value = zmq_msg_gets(&msg, property_);
00642         if (value == ZMQ_NULLPTR)
00643             throw error_t();
00644         return value;
00645     }
00646 #endif
00647
00648 #if defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 0)
00649     uint32_t routing_id() const
00650     {
00651         return zmq_msg_routing_id(const_cast<zmq_msg_t *>(&msg));
00652     }
00653
00654     void set_routing_id(uint32_t routing_id)
00655     {
00656         int rc = zmq_msg_set_routing_id(&msg, routing_id);
00657         if (rc != 0)
00658             throw error_t();
00659     }
00660
00661     const char *group() const
00662     {
00663         return zmq_msg_group(const_cast<zmq_msg_t *>(&msg));
00664     }
00665
00666     void set_group(const char *group)
00667     {
00668         int rc = zmq_msg_set_group(&msg, group);
00669         if (rc != 0)
00670             throw error_t();
00671     }
00672 #endif
00673
00674     // interpret message content as a string
00675     std::string to_string() const
00676     {
00677         return std::string(static_cast<const char *>(data()), size());
00678     }
00679 #if CPPZMQ_HAS_STRING_VIEW
00680     // interpret message content as a string
00681     std::string_view to_string_view() const noexcept
00682     {
00683         return std::string_view(static_cast<const char *>(data()), size());
00684     }
00685 #endif
00686
00693     std::string str() const
00694     {
00695         // Partly mutuated from the same method in zmq::multipart_t
00696         std::stringstream os;
00697
00698         const unsigned char *msg_data = this->data<unsigned char>();
00699         unsigned char byte;
00700         size_t size = this->size();
00701         int is_ascii[2] = {0, 0};
00702
00703         os « "zmq::message_t [size " « std::dec « std::setw(3)
00704            « std::setfill('0') « size « "] (";
00705         // Totally arbitrary
```

```
00706         if (size >= 1000) {
00707             os « "... too big to print)";
00708         } else {
00709             while (size--) {
00710                 byte = *msg_data++;
00711
00712                 is_ascii[1] = (byte >= 32 && byte < 127);
00713                 if (is_ascii[1] != is_ascii[0])
00714                     os « " "; // Separate text/non text
00715
00716                 if (is_ascii[1]) {
00717                     os « byte;
00718                 } else {
00719                     os « std::hex « std::uppercase « std::setw(2)
00720                        « std::setfill('0') « static_cast<short>(byte);
00721                 }
00722                 is_ascii[0] = is_ascii[1];
00723             }
00724             os « ")";
00725         }
00726         return os.str();
00727     }
00728
00729     void swap(message_t &other) ZMQ_NOTHROW
00730     {
00731         // this assumes zmq::msg_t from libzmq is trivially relocatable
00732         std::swap(msg, other.msg);
00733     }
00734
00735     ZMQ_NODISCARD zmq_msg_t *handle() ZMQ_NOTHROW { return &msg; }
00736     ZMQ_NODISCARD const zmq_msg_t *handle() const ZMQ_NOTHROW { return &msg; }
00737
00738   private:
00739     //  The underlying message
00740     zmq_msg_t msg;
00741
00742     //  Disable implicit message copying, so that users won't use shared
00743     //  messages (less efficient) without being aware of the fact.
00744     message_t(const message_t &) ZMQ_DELETED_FUNCTION;
00745     void operator=(const message_t &) ZMQ_DELETED_FUNCTION;
00746 };
00747
00748 inline void swap(message_t &a, message_t &b) ZMQ_NOTHROW
00749 {
00750     a.swap(b);
00751 }
00752
00753 #ifdef ZMQ_CPP11
00754 enum class ctxopt
00755 {
00756 #ifdef ZMQ_BLOCKY
00757     blocky = ZMQ_BLOCKY,
00758 #endif
00759 #ifdef ZMQ_IO_THREADS
00760     io_threads = ZMQ_IO_THREADS,
00761 #endif
00762 #ifdef ZMQ_THREAD_SCHED_POLICY
00763     thread_sched_policy = ZMQ_THREAD_SCHED_POLICY,
00764 #endif
00765 #ifdef ZMQ_THREAD_PRIORITY
00766     thread_priority = ZMQ_THREAD_PRIORITY,
00767 #endif
00768 #ifdef ZMQ_THREAD_AFFINITY_CPU_ADD
00769     thread_affinity_cpu_add = ZMQ_THREAD_AFFINITY_CPU_ADD,
00770 #endif
00771 #ifdef ZMQ_THREAD_AFFINITY_CPU_REMOVE
00772     thread_affinity_cpu_remove = ZMQ_THREAD_AFFINITY_CPU_REMOVE,
00773 #endif
00774 #ifdef ZMQ_THREAD_NAME_PREFIX
00775     thread_name_prefix = ZMQ_THREAD_NAME_PREFIX,
00776 #endif
00777 #ifdef ZMQ_MAX_MSGSZ
00778     max_msgsz = ZMQ_MAX_MSGSZ,
00779 #endif
00780 #ifdef ZMQ_ZERO_COPY_RECV
00781     zero_copy_recv = ZMQ_ZERO_COPY_RECV,
00782 #endif
00783 #ifdef ZMQ_MAX_SOCKETS
00784     max_sockets = ZMQ_MAX_SOCKETS,
00785 #endif
00786 #ifdef ZMQ_SOCKET_LIMIT
00787     socket_limit = ZMQ_SOCKET_LIMIT,
00788 #endif
00789 #ifdef ZMQ_IPV6
00790     ipv6 = ZMQ_IPV6,
00791 #endif
00792 #ifdef ZMQ_MSG_T_SIZE
```

```
00793     msg_t_size = ZMQ_MSG_T_SIZE
00794 #endif
00795 };
00796 #endif
00797
00798 class context_t
00799 {
00800   public:
00801     context_t()
00802     {
00803         ptr = zmq_ctx_new();
00804         if (ptr == ZMQ_NULLPTR)
00805             throw error_t();
00806     }
00807
00808
00809     explicit context_t(int io_threads_, int max_sockets_ = ZMQ_MAX_SOCKETS_DFLT)
00810     {
00811         ptr = zmq_ctx_new();
00812         if (ptr == ZMQ_NULLPTR)
00813             throw error_t();
00814
00815         int rc = zmq_ctx_set(ptr, ZMQ_IO_THREADS, io_threads_);
00816         ZMQ_ASSERT(rc == 0);
00817
00818         rc = zmq_ctx_set(ptr, ZMQ_MAX_SOCKETS, max_sockets_);
00819         ZMQ_ASSERT(rc == 0);
00820     }
00821
00822 #ifdef ZMQ_HAS_RVALUE_REFS
00823     context_t(context_t &&rhs) ZMQ_NOTHROW : ptr(rhs.ptr) { rhs.ptr = ZMQ_NULLPTR; }
00824     context_t &operator=(context_t &&rhs) ZMQ_NOTHROW
00825     {
00826         close();
00827         std::swap(ptr, rhs.ptr);
00828         return *this;
00829     }
00830 #endif
00831
00832     ~context_t() ZMQ_NOTHROW { close(); }
00833
00834     ZMQ_CPP11_DEPRECATED("from 4.7.0, use set taking zmq::ctxopt instead")
00835     int setctxopt(int option_, int optval_)
00836     {
00837         int rc = zmq_ctx_set(ptr, option_, optval_);
00838         ZMQ_ASSERT(rc == 0);
00839         return rc;
00840     }
00841
00842     ZMQ_CPP11_DEPRECATED("from 4.7.0, use get taking zmq::ctxopt instead")
00843     int getctxopt(int option_) { return zmq_ctx_get(ptr, option_); }
00844
00845 #ifdef ZMQ_CPP11
00846     void set(ctxopt option, int optval)
00847     {
00848         int rc = zmq_ctx_set(ptr, static_cast<int>(option), optval);
00849         if (rc == -1)
00850             throw error_t();
00851     }
00852
00853     ZMQ_NODISCARD int get(ctxopt option)
00854     {
00855         int rc = zmq_ctx_get(ptr, static_cast<int>(option));
00856         // some options have a default value of -1
00857         // which is unfortunate, and may result in errors
00858         // that don't make sense
00859         if (rc == -1)
00860             throw error_t();
00861         return rc;
00862     }
00863 #endif
00864
00865     // Terminates context (see also shutdown()).
00866     void close() ZMQ_NOTHROW
00867     {
00868         if (ptr == ZMQ_NULLPTR)
00869             return;
00870
00871         int rc;
00872         do {
00873             rc = zmq_ctx_term(ptr);
00874         } while (rc == -1 && errno == EINTR);
00875
00876         ZMQ_ASSERT(rc == 0);
00877         ptr = ZMQ_NULLPTR;
00878     }
00879
```

```
00880     // Shutdown context in preparation for termination (close()).
00881     // Causes all blocking socket operations and any further
00882     // socket operations to return with ETERM.
00883     void shutdown() ZMQ_NOTHROW
00884     {
00885         if (ptr == ZMQ_NULLPTR)
00886             return;
00887         int rc = zmq_ctx_shutdown(ptr);
00888         ZMQ_ASSERT(rc == 0);
00889     }
00890
00891     //  Be careful with this, it's probably only useful for
00892     //  using the C api together with an existing C++ api.
00893     //  Normally you should never need to use this.
00894     ZMQ_EXPLICIT operator void *() ZMQ_NOTHROW { return ptr; }
00895
00896     ZMQ_EXPLICIT operator void const *() const ZMQ_NOTHROW { return ptr; }
00897
00898     ZMQ_NODISCARD void *handle() ZMQ_NOTHROW { return ptr; }
00899
00900     ZMQ_DEPRECATED("from 4.7.0, use handle() != nullptr instead")
00901     operator bool() const ZMQ_NOTHROW { return ptr != ZMQ_NULLPTR; }
00902
00903     void swap(context_t &other) ZMQ_NOTHROW { std::swap(ptr, other.ptr); }
00904
00905   private:
00906     void *ptr;
00907
00908     context_t(const context_t &) ZMQ_DELETED_FUNCTION;
00909     void operator=(const context_t &) ZMQ_DELETED_FUNCTION;
00910 };
00911
00912 inline void swap(context_t &a, context_t &b) ZMQ_NOTHROW
00913 {
00914     a.swap(b);
00915 }
00916
00917 #ifdef ZMQ_CPP11
00918
00919 struct recv_buffer_size
00920 {
00921     size_t size;             // number of bytes written to buffer
00922     size_t untruncated_size; // untruncated message size in bytes
00923
00924     ZMQ_NODISCARD bool truncated() const noexcept
00925     {
00926         return size != untruncated_size;
00927     }
00928 };
00929
00930 #if CPPZMQ_HAS_OPTIONAL
00931
00932 using send_result_t = std::optional<size_t>;
00933 using recv_result_t = std::optional<size_t>;
00934 using recv_buffer_result_t = std::optional<recv_buffer_size>;
00935
00936 #else
00937
00938 namespace detail
00939 {
00940 // A C++11 type emulating the most basic
00941 // operations of std::optional for trivial types
00942 template<class T> class trivial_optional
00943 {
00944   public:
00945     static_assert(std::is_trivial<T>::value, "T must be trivial");
00946     using value_type = T;
00947
00948     trivial_optional() = default;
00949     trivial_optional(T value) noexcept : _value(value), _has_value(true) {}
00950
00951     const T *operator->() const noexcept
00952     {
00953         assert(_has_value);
00954         return &_value;
00955     }
00956     T *operator->() noexcept
00957     {
00958         assert(_has_value);
00959         return &_value;
00960     }
00961
00962     const T &operator*() const noexcept
00963     {
00964         assert(_has_value);
00965         return _value;
00966     }
```

```
00967     T &operator*() noexcept
00968     {
00969         assert(_has_value);
00970         return _value;
00971     }
00972
00973     T &value()
00974     {
00975         if (!_has_value)
00976             throw std::exception();
00977         return _value;
00978     }
00979     const T &value() const
00980     {
00981         if (!_has_value)
00982             throw std::exception();
00983         return _value;
00984     }
00985
00986     explicit operator bool() const noexcept { return _has_value; }
00987     bool has_value() const noexcept { return _has_value; }
00988
00989   private:
00990     T _value{};
00991     bool _has_value{false};
00992 };
00993 } // namespace detail
00994
00995 using send_result_t = detail::trivial_optional<size_t>;
00996 using recv_result_t = detail::trivial_optional<size_t>;
00997 using recv_buffer_result_t = detail::trivial_optional<recv_buffer_size>;
00998
00999 #endif
01000
01001 namespace detail
01002 {
01003 template<class T> constexpr T enum_bit_or(T a, T b) noexcept
01004 {
01005     static_assert(std::is_enum<T>::value, "must be enum");
01006     using U = typename std::underlying_type<T>::type;
01007     return static_cast<T>(static_cast<U>(a) | static_cast<U>(b));
01008 }
01009 template<class T> constexpr T enum_bit_and(T a, T b) noexcept
01010 {
01011     static_assert(std::is_enum<T>::value, "must be enum");
01012     using U = typename std::underlying_type<T>::type;
01013     return static_cast<T>(static_cast<U>(a) & static_cast<U>(b));
01014 }
01015 template<class T> constexpr T enum_bit_xor(T a, T b) noexcept
01016 {
01017     static_assert(std::is_enum<T>::value, "must be enum");
01018     using U = typename std::underlying_type<T>::type;
01019     return static_cast<T>(static_cast<U>(a) ^ static_cast<U>(b));
01020 }
01021 template<class T> constexpr T enum_bit_not(T a) noexcept
01022 {
01023     static_assert(std::is_enum<T>::value, "must be enum");
01024     using U = typename std::underlying_type<T>::type;
01025     return static_cast<T>(~static_cast<U>(a));
01026 }
01027 } // namespace detail
01028
01029 // partially satisfies named requirement BitmaskType
01030 enum class send_flags : int
01031 {
01032     none = 0,
01033     dontwait = ZMQ_DONTWAIT,
01034     sndmore = ZMQ_SNDMORE
01035 };
01036
01037 constexpr send_flags operator|(send_flags a, send_flags b) noexcept
01038 {
01039     return detail::enum_bit_or(a, b);
01040 }
01041 constexpr send_flags operator&(send_flags a, send_flags b) noexcept
01042 {
01043     return detail::enum_bit_and(a, b);
01044 }
01045 constexpr send_flags operator^(send_flags a, send_flags b) noexcept
01046 {
01047     return detail::enum_bit_xor(a, b);
01048 }
01049 constexpr send_flags operator~(send_flags a) noexcept
01050 {
01051     return detail::enum_bit_not(a);
01052 }
01053
```

```
01054 // partially satisfies named requirement BitmaskType
01055 enum class recv_flags : int
01056 {
01057     none = 0,
01058     dontwait = ZMQ_DONTWAIT
01059 };
01060
01061 constexpr recv_flags operator|(recv_flags a, recv_flags b) noexcept
01062 {
01063     return detail::enum_bit_or(a, b);
01064 }
01065 constexpr recv_flags operator&(recv_flags a, recv_flags b) noexcept
01066 {
01067     return detail::enum_bit_and(a, b);
01068 }
01069 constexpr recv_flags operator^(recv_flags a, recv_flags b) noexcept
01070 {
01071     return detail::enum_bit_xor(a, b);
01072 }
01073 constexpr recv_flags operator~(recv_flags a) noexcept
01074 {
01075     return detail::enum_bit_not(a);
01076 }
01077
01078
01079 // mutable_buffer, const_buffer and buffer are based on
01080 // the Networking TS specification, draft:
01081 // http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4771.pdf
01082
01083 class mutable_buffer
01084 {
01085   public:
01086     constexpr mutable_buffer() noexcept : _data(nullptr), _size(0) {}
01087     constexpr mutable_buffer(void *p, size_t n) noexcept : _data(p), _size(n)
01088     {
01089 #ifdef ZMQ_EXTENDED_CONSTEXPR
01090         assert(p != nullptr || n == 0);
01091 #endif
01092     }
01093
01094     constexpr void *data() const noexcept { return _data; }
01095     constexpr size_t size() const noexcept { return _size; }
01096     mutable_buffer &operator+=(size_t n) noexcept
01097     {
01098         // (std::min) is a workaround for when a min macro is defined
01099         const auto shift = (std::min)(n, _size);
01100         _data = static_cast<char *>(_data) + shift;
01101         _size -= shift;
01102         return *this;
01103     }
01104
01105   private:
01106     void *_data;
01107     size_t _size;
01108 };
01109
01110 inline mutable_buffer operator+(const mutable_buffer &mb, size_t n) noexcept
01111 {
01112     return mutable_buffer(static_cast<char *>(mb.data()) + (std::min)(n, mb.size()),
01113                           mb.size() - (std::min)(n, mb.size()));
01114 }
01115 inline mutable_buffer operator+(size_t n, const mutable_buffer &mb) noexcept
01116 {
01117     return mb + n;
01118 }
01119
01120 class const_buffer
01121 {
01122   public:
01123     constexpr const_buffer() noexcept : _data(nullptr), _size(0) {}
01124     constexpr const_buffer(const void *p, size_t n) noexcept : _data(p), _size(n)
01125     {
01126 #ifdef ZMQ_EXTENDED_CONSTEXPR
01127         assert(p != nullptr || n == 0);
01128 #endif
01129     }
01130     constexpr const_buffer(const mutable_buffer &mb) noexcept :
01131         _data(mb.data()), _size(mb.size())
01132     {
01133     }
01134
01135     constexpr const void *data() const noexcept { return _data; }
01136     constexpr size_t size() const noexcept { return _size; }
01137     const_buffer &operator+=(size_t n) noexcept
01138     {
01139         const auto shift = (std::min)(n, _size);
01140         _data = static_cast<const char *>(_data) + shift;
```

```
01141         _size -= shift;
01142         return *this;
01143     }
01144
01145   private:
01146     const void *_data;
01147     size_t _size;
01148 };
01149
01150 inline const_buffer operator+(const const_buffer &cb, size_t n) noexcept
01151 {
01152     return const_buffer(static_cast<const char *>(cb.data())
01153                         + (std::min)(n, cb.size()),
01154                       cb.size() - (std::min)(n, cb.size()));
01155 }
01156 inline const_buffer operator+(size_t n, const const_buffer &cb) noexcept
01157 {
01158     return cb + n;
01159 }
01160
01161 // buffer creation
01162
01163 constexpr mutable_buffer buffer(void *p, size_t n) noexcept
01164 {
01165     return mutable_buffer(p, n);
01166 }
01167 constexpr const_buffer buffer(const void *p, size_t n) noexcept
01168 {
01169     return const_buffer(p, n);
01170 }
01171 constexpr mutable_buffer buffer(const mutable_buffer &mb) noexcept
01172 {
01173     return mb;
01174 }
01175 inline mutable_buffer buffer(const mutable_buffer &mb, size_t n) noexcept
01176 {
01177     return mutable_buffer(mb.data(), (std::min)(mb.size(), n));
01178 }
01179 constexpr const_buffer buffer(const const_buffer &cb) noexcept
01180 {
01181     return cb;
01182 }
01183 inline const_buffer buffer(const const_buffer &cb, size_t n) noexcept
01184 {
01185     return const_buffer(cb.data(), (std::min)(cb.size(), n));
01186 }
01187
01188 namespace detail
01189 {
01190 template<class T> struct is_buffer
01191 {
01192     static constexpr bool value =
01193       std::is_same<T, const_buffer>::value || std::is_same<T, mutable_buffer>::value;
01194 };
01195
01196 template<class T> struct is_pod_like
01197 {
01198     // NOTE: The networking draft N4771 section 16.11 requires
01199     // T in the buffer functions below to be
01200     // trivially copyable OR standard layout.
01201     // Here we decide to be conservative and require both.
01202     static constexpr bool value =
01203       ZMQ_IS_TRIVIALLY_COPYABLE(T) && std::is_standard_layout<T>::value;
01204 };
01205
01206 template<class C> constexpr auto seq_size(const C &c) noexcept -> decltype(c.size())
01207 {
01208     return c.size();
01209 }
01210 template<class T, size_t N>
01211 constexpr size_t seq_size(const T (&/*array*/)[N]) noexcept
01212 {
01213     return N;
01214 }
01215
01216 template<class Seq>
01217 auto buffer_contiguous_sequence(Seq &&seq) noexcept
01218   -> decltype(buffer(std::addressof(*std::begin(seq)), size_t{}))
01219 {
01220     using T = typename std::remove_cv<
01221       typename std::remove_reference<decltype(*std::begin(seq))>::type>::type;
01222     static_assert(detail::is_pod_like<T>::value, "T must be POD");
01223
01224     const auto size = seq_size(seq);
01225     return buffer(size != 0u ? std::addressof(*std::begin(seq)) : nullptr,
01226                   size * sizeof(T));
01227 }
```

```
01228 template<class Seq>
01229 auto buffer_contiguous_sequence(Seq &&seq, size_t n_bytes) noexcept
01230   -> decltype(buffer_contiguous_sequence(seq))
01231 {
01232     using T = typename std::remove_cv<
01233       typename std::remove_reference<decltype(*std::begin(seq))>::type>::type;
01234     static_assert(detail::is_pod_like<T>::value, "T must be POD");
01235
01236     const auto size = seq_size(seq);
01237     return buffer(size != 0u ? std::addressof(*std::begin(seq)) : nullptr,
01238                   (std::min)(size * sizeof(T), n_bytes));
01239 }
01240
01241 } // namespace detail
01242
01243 // C array
01244 template<class T, size_t N> mutable_buffer buffer(T (&data)[N]) noexcept
01245 {
01246     return detail::buffer_contiguous_sequence(data);
01247 }
01248 template<class T, size_t N>
01249 mutable_buffer buffer(T (&data)[N], size_t n_bytes) noexcept
01250 {
01251     return detail::buffer_contiguous_sequence(data, n_bytes);
01252 }
01253 template<class T, size_t N> const_buffer buffer(const T (&data)[N]) noexcept
01254 {
01255     return detail::buffer_contiguous_sequence(data);
01256 }
01257 template<class T, size_t N>
01258 const_buffer buffer(const T (&data)[N], size_t n_bytes) noexcept
01259 {
01260     return detail::buffer_contiguous_sequence(data, n_bytes);
01261 }
01262 // std::array
01263 template<class T, size_t N> mutable_buffer buffer(std::array<T, N> &data) noexcept
01264 {
01265     return detail::buffer_contiguous_sequence(data);
01266 }
01267 template<class T, size_t N>
01268 mutable_buffer buffer(std::array<T, N> &data, size_t n_bytes) noexcept
01269 {
01270     return detail::buffer_contiguous_sequence(data, n_bytes);
01271 }
01272 template<class T, size_t N>
01273 const_buffer buffer(std::array<const T, N> &data) noexcept
01274 {
01275     return detail::buffer_contiguous_sequence(data);
01276 }
01277 template<class T, size_t N>
01278 const_buffer buffer(std::array<const T, N> &data, size_t n_bytes) noexcept
01279 {
01280     return detail::buffer_contiguous_sequence(data, n_bytes);
01281 }
01282 template<class T, size_t N>
01283 const_buffer buffer(const std::array<T, N> &data) noexcept
01284 {
01285     return detail::buffer_contiguous_sequence(data);
01286 }
01287 template<class T, size_t N>
01288 const_buffer buffer(const std::array<T, N> &data, size_t n_bytes) noexcept
01289 {
01290     return detail::buffer_contiguous_sequence(data, n_bytes);
01291 }
01292 // std::vector
01293 template<class T, class Allocator>
01294 mutable_buffer buffer(std::vector<T, Allocator> &data) noexcept
01295 {
01296     return detail::buffer_contiguous_sequence(data);
01297 }
01298 template<class T, class Allocator>
01299 mutable_buffer buffer(std::vector<T, Allocator> &data, size_t n_bytes) noexcept
01300 {
01301     return detail::buffer_contiguous_sequence(data, n_bytes);
01302 }
01303 template<class T, class Allocator>
01304 const_buffer buffer(const std::vector<T, Allocator> &data) noexcept
01305 {
01306     return detail::buffer_contiguous_sequence(data);
01307 }
01308 template<class T, class Allocator>
01309 const_buffer buffer(const std::vector<T, Allocator> &data, size_t n_bytes) noexcept
01310 {
01311     return detail::buffer_contiguous_sequence(data, n_bytes);
01312 }
01313 // std::basic_string
01314 template<class T, class Traits, class Allocator>
```

```
01315 mutable_buffer buffer(std::basic_string<T, Traits, Allocator> &data) noexcept
01316 {
01317     return detail::buffer_contiguous_sequence(data);
01318 }
01319 template<class T, class Traits, class Allocator>
01320 mutable_buffer buffer(std::basic_string<T, Traits, Allocator> &data,
01321                       size_t n_bytes) noexcept
01322 {
01323     return detail::buffer_contiguous_sequence(data, n_bytes);
01324 }
01325 template<class T, class Traits, class Allocator>
01326 const_buffer buffer(const std::basic_string<T, Traits, Allocator> &data) noexcept
01327 {
01328     return detail::buffer_contiguous_sequence(data);
01329 }
01330 template<class T, class Traits, class Allocator>
01331 const_buffer buffer(const std::basic_string<T, Traits, Allocator> &data,
01332                     size_t n_bytes) noexcept
01333 {
01334     return detail::buffer_contiguous_sequence(data, n_bytes);
01335 }
01336
01337 #if CPPZMQ_HAS_STRING_VIEW
01338 // std::basic_string_view
01339 template<class T, class Traits>
01340 const_buffer buffer(std::basic_string_view<T, Traits> data) noexcept
01341 {
01342     return detail::buffer_contiguous_sequence(data);
01343 }
01344 template<class T, class Traits>
01345 const_buffer buffer(std::basic_string_view<T, Traits> data, size_t n_bytes) noexcept
01346 {
01347     return detail::buffer_contiguous_sequence(data, n_bytes);
01348 }
01349 #endif
01350
01351 // Buffer for a string literal (null terminated)
01352 // where the buffer size excludes the terminating character.
01353 // Equivalent to zmq::buffer(std::string_view("...")).
01354 template<class Char, size_t N>
01355 constexpr const_buffer str_buffer(const Char (&data)[N]) noexcept
01356 {
01357     static_assert(detail::is_pod_like<Char>::value, "Char must be POD");
01358 #ifdef ZMQ_EXTENDED_CONSTEXPR
01359     assert(data[N - 1] == Char{0});
01360 #endif
01361     return const_buffer(static_cast<const Char *>(data), (N - 1) * sizeof(Char));
01362 }
01363
01364 namespace literals
01365 {
01366 constexpr const_buffer operator"" _zbuf(const char *str, size_t len) noexcept
01367 {
01368     return const_buffer(str, len * sizeof(char));
01369 }
01370 constexpr const_buffer operator"" _zbuf(const wchar_t *str, size_t len) noexcept
01371 {
01372     return const_buffer(str, len * sizeof(wchar_t));
01373 }
01374 constexpr const_buffer operator"" _zbuf(const char16_t *str, size_t len) noexcept
01375 {
01376     return const_buffer(str, len * sizeof(char16_t));
01377 }
01378 constexpr const_buffer operator"" _zbuf(const char32_t *str, size_t len) noexcept
01379 {
01380     return const_buffer(str, len * sizeof(char32_t));
01381 }
01382 }
01383
01384 #ifdef ZMQ_CPP11
01385 enum class socket_type : int
01386 {
01387     req = ZMQ_REQ,
01388     rep = ZMQ_REP,
01389     dealer = ZMQ_DEALER,
01390     router = ZMQ_ROUTER,
01391     pub = ZMQ_PUB,
01392     sub = ZMQ_SUB,
01393     xpub = ZMQ_XPUB,
01394     xsub = ZMQ_XSUB,
01395     push = ZMQ_PUSH,
01396     pull = ZMQ_PULL,
01397 #if defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 0)
01398     server = ZMQ_SERVER,
01399     client = ZMQ_CLIENT,
01400     radio = ZMQ_RADIO,
01401     dish = ZMQ_DISH,
```

```
01402     gather = ZMQ_GATHER,
01403     scatter = ZMQ_SCATTER,
01404     dgram = ZMQ_DGRAM,
01405 #endif
01406 #if defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 3, 3)
01407     peer = ZMQ_PEER,
01408     channel = ZMQ_CHANNEL,
01409 #endif
01410 #if ZMQ_VERSION_MAJOR >= 4
01411     stream = ZMQ_STREAM,
01412 #endif
01413     pair = ZMQ_PAIR
01414 };
01415 #endif
01416
01417 namespace sockopt
01418 {
01419 // There are two types of options,
01420 // integral type with known compiler time size (int, bool, int64_t, uint64_t)
01421 // and arrays with dynamic size (strings, binary data).
01422
01423 // BoolUnit: if true accepts values of type bool (but passed as T into libzmq)
01424 template<int Opt, class T, bool BoolUnit = false> struct integral_option
01425 {
01426 };
01427
01428 // NullTerm:
01429 // 0: binary data
01430 // 1: null-terminated string (`getsockopt` size includes null)
01431 // 2: binary (size 32) or Z85 encoder string of size 41 (null included)
01432 template<int Opt, int NullTerm = 1> struct array_option
01433 {
01434 };
01435
01436 #define ZMQ_DEFINE_INTEGRAL_OPT(OPT, NAME, TYPE)                                \
01437     using NAME##_t = integral_option<OPT, TYPE, false>;                         \
01438     ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {}
01439 #define ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(OPT, NAME, TYPE)                      \
01440     using NAME##_t = integral_option<OPT, TYPE, true>;                          \
01441     ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {}
01442 #define ZMQ_DEFINE_ARRAY_OPT(OPT, NAME)                                         \
01443     using NAME##_t = array_option<OPT>;                                         \
01444     ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {}
01445 #define ZMQ_DEFINE_ARRAY_OPT_BINARY(OPT, NAME)                                  \
01446     using NAME##_t = array_option<OPT, 0>;                                      \
01447     ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {}
01448 #define ZMQ_DEFINE_ARRAY_OPT_BIN_OR_Z85(OPT, NAME)                              \
01449     using NAME##_t = array_option<OPT, 2>;                                      \
01450     ZMQ_INLINE_VAR ZMQ_CONSTEXPR_VAR NAME##_t NAME {}
01451
01452 // deprecated, use zmq::fd_t
01453 using cppzmq_fd_t = ::zmq::fd_t;
01454
01455 #ifdef ZMQ_AFFINITY
01456 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_AFFINITY, affinity, uint64_t);
01457 #endif
01458 #ifdef ZMQ_BACKLOG
01459 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_BACKLOG, backlog, int);
01460 #endif
01461 #ifdef ZMQ_BINDTODEVICE
01462 ZMQ_DEFINE_ARRAY_OPT_BINARY(ZMQ_BINDTODEVICE, bindtodevice);
01463 #endif
01464 #ifdef ZMQ_CONFLATE
01465 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_CONFLATE, conflate, int);
01466 #endif
01467 #ifdef ZMQ_CONNECT_ROUTING_ID
01468 ZMQ_DEFINE_ARRAY_OPT(ZMQ_CONNECT_ROUTING_ID, connect_routing_id);
01469 #endif
01470 #ifdef ZMQ_CONNECT_TIMEOUT
01471 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_CONNECT_TIMEOUT, connect_timeout, int);
01472 #endif
01473 #ifdef ZMQ_CURVE_PUBLICKEY
01474 ZMQ_DEFINE_ARRAY_OPT_BIN_OR_Z85(ZMQ_CURVE_PUBLICKEY, curve_publickey);
01475 #endif
01476 #ifdef ZMQ_CURVE_SECRETKEY
01477 ZMQ_DEFINE_ARRAY_OPT_BIN_OR_Z85(ZMQ_CURVE_SECRETKEY, curve_secretkey);
01478 #endif
01479 #ifdef ZMQ_CURVE_SERVER
01480 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_CURVE_SERVER, curve_server, int);
01481 #endif
01482 #ifdef ZMQ_CURVE_SERVERKEY
01483 ZMQ_DEFINE_ARRAY_OPT_BIN_OR_Z85(ZMQ_CURVE_SERVERKEY, curve_serverkey);
01484 #endif
01485 #ifdef ZMQ_DISCONNECT_MSG
01486 ZMQ_DEFINE_ARRAY_OPT_BINARY(ZMQ_DISCONNECT_MSG, disconnect_msg);
01487 #endif
01488 #ifdef ZMQ_EVENTS
```

```
01489 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_EVENTS, events, int);
01490 #endif
01491 #ifdef ZMQ_FD
01492 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_FD, fd, ::zmq::fd_t);
01493 #endif
01494 #ifdef ZMQ_GSSAPI_PLAINTEXT
01495 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_GSSAPI_PLAINTEXT, gssapi_plaintext, int);
01496 #endif
01497 #ifdef ZMQ_GSSAPI_SERVER
01498 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_GSSAPI_SERVER, gssapi_server, int);
01499 #endif
01500 #ifdef ZMQ_GSSAPI_SERVICE_PRINCIPAL
01501 ZMQ_DEFINE_ARRAY_OPT(ZMQ_GSSAPI_SERVICE_PRINCIPAL, gssapi_service_principal);
01502 #endif
01503 #ifdef ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAMETYPE
01504 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_GSSAPI_SERVICE_PRINCIPAL_NAMETYPE,
01505                         gssapi_service_principal_nametype,
01506                         int);
01507 #endif
01508 #ifdef ZMQ_GSSAPI_PRINCIPAL
01509 ZMQ_DEFINE_ARRAY_OPT(ZMQ_GSSAPI_PRINCIPAL, gssapi_principal);
01510 #endif
01511 #ifdef ZMQ_GSSAPI_PRINCIPAL_NAMETYPE
01512 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_GSSAPI_PRINCIPAL_NAMETYPE,
01513                         gssapi_principal_nametype,
01514                         int);
01515 #endif
01516 #ifdef ZMQ_HANDSHAKE_IVL
01517 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_HANDSHAKE_IVL, handshake_ivl, int);
01518 #endif
01519 #ifdef ZMQ_HEARTBEAT_IVL
01520 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_HEARTBEAT_IVL, heartbeat_ivl, int);
01521 #endif
01522 #ifdef ZMQ_HEARTBEAT_TIMEOUT
01523 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_HEARTBEAT_TIMEOUT, heartbeat_timeout, int);
01524 #endif
01525 #ifdef ZMQ_HEARTBEAT_TTL
01526 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_HEARTBEAT_TTL, heartbeat_ttl, int);
01527 #endif
01528 #ifdef ZMQ_HELLO_MSG
01529 ZMQ_DEFINE_ARRAY_OPT_BINARY(ZMQ_HELLO_MSG, hello_msg);
01530 #endif
01531 #ifdef ZMQ_IMMEDIATE
01532 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_IMMEDIATE, immediate, int);
01533 #endif
01534 #ifdef ZMQ_INVERT_MATCHING
01535 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_INVERT_MATCHING, invert_matching, int);
01536 #endif
01537 #ifdef ZMQ_IPV6
01538 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_IPV6, ipv6, int);
01539 #endif
01540 #ifdef ZMQ_LAST_ENDPOINT
01541 ZMQ_DEFINE_ARRAY_OPT(ZMQ_LAST_ENDPOINT, last_endpoint);
01542 #endif
01543 #ifdef ZMQ_LINGER
01544 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_LINGER, linger, int);
01545 #endif
01546 #ifdef ZMQ_MAXMSGSIZE
01547 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_MAXMSGSIZE, maxmsgsize, int64_t);
01548 #endif
01549 #ifdef ZMQ_MECHANISM
01550 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_MECHANISM, mechanism, int);
01551 #endif
01552 #ifdef ZMQ_METADATA
01553 ZMQ_DEFINE_ARRAY_OPT(ZMQ_METADATA, metadata);
01554 #endif
01555 #ifdef ZMQ_MULTICAST_HOPS
01556 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_MULTICAST_HOPS, multicast_hops, int);
01557 #endif
01558 #ifdef ZMQ_MULTICAST_LOOP
01559 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_MULTICAST_LOOP, multicast_loop, int);
01560 #endif
01561 #ifdef ZMQ_MULTICAST_MAXTPDU
01562 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_MULTICAST_MAXTPDU, multicast_maxtpdu, int);
01563 #endif
01564 #ifdef ZMQ_ONLY_FIRST_SUBSCRIBE
01565 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_ONLY_FIRST_SUBSCRIBE, only_first_subscribe, int);
01566 #endif
01567 #ifdef ZMQ_PLAIN_SERVER
01568 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_PLAIN_SERVER, plain_server, int);
01569 #endif
01570 #ifdef ZMQ_PLAIN_PASSWORD
01571 ZMQ_DEFINE_ARRAY_OPT(ZMQ_PLAIN_PASSWORD, plain_password);
01572 #endif
01573 #ifdef ZMQ_PLAIN_USERNAME
01574 ZMQ_DEFINE_ARRAY_OPT(ZMQ_PLAIN_USERNAME, plain_username);
01575 #endif
```

```
01576 #ifdef ZMQ_PRIORITY
01577 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_PRIORITY, priority, int);
01578 #endif
01579 #ifdef ZMQ_USE_FD
01580 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_USE_FD, use_fd, int);
01581 #endif
01582 #ifdef ZMQ_PROBE_ROUTER
01583 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_PROBE_ROUTER, probe_router, int);
01584 #endif
01585 #ifdef ZMQ_RATE
01586 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RATE, rate, int);
01587 #endif
01588 #ifdef ZMQ_RCVBUF
01589 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RCVBUF, rcvbuf, int);
01590 #endif
01591 #ifdef ZMQ_RCVHWM
01592 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RCVHWM, rcvhwm, int);
01593 #endif
01594 #ifdef ZMQ_RCVMORE
01595 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_RCVMORE, rcvmore, int);
01596 #endif
01597 #ifdef ZMQ_RCVTIMEO
01598 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RCVTIMEO, rcvtimeo, int);
01599 #endif
01600 #ifdef ZMQ_RECONNECT_IVL
01601 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RECONNECT_IVL, reconnect_ivl, int);
01602 #endif
01603 #ifdef ZMQ_RECONNECT_IVL_MAX
01604 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RECONNECT_IVL_MAX, reconnect_ivl_max, int);
01605 #endif
01606 #ifdef ZMQ_RECONNECT_STOP
01607 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RECONNECT_STOP, reconnect_stop, int);
01608 #endif
01609 #ifdef ZMQ_RECOVERY_IVL
01610 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_RECOVERY_IVL, recovery_ivl, int);
01611 #endif
01612 #ifdef ZMQ_REQ_CORRELATE
01613 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_REQ_CORRELATE, req_correlate, int);
01614 #endif
01615 #ifdef ZMQ_REQ_RELAXED
01616 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_REQ_RELAXED, req_relaxed, int);
01617 #endif
01618 #ifdef ZMQ_ROUTER_HANDOVER
01619 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_ROUTER_HANDOVER, router_handover, int);
01620 #endif
01621 #ifdef ZMQ_ROUTER_MANDATORY
01622 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_ROUTER_MANDATORY, router_mandatory, int);
01623 #endif
01624 #ifdef ZMQ_ROUTER_NOTIFY
01625 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_ROUTER_NOTIFY, router_notify, int);
01626 #endif
01627 #ifdef ZMQ_ROUTING_ID
01628 ZMQ_DEFINE_ARRAY_OPT_BINARY(ZMQ_ROUTING_ID, routing_id);
01629 #endif
01630 #ifdef ZMQ_SNDBUF
01631 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_SNDBUF, sndbuf, int);
01632 #endif
01633 #ifdef ZMQ_SNDHWM
01634 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_SNDHWM, sndhwm, int);
01635 #endif
01636 #ifdef ZMQ_SNDTIMEO
01637 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_SNDTIMEO, sndtimeo, int);
01638 #endif
01639 #ifdef ZMQ_SOCKS_PASSWORD
01640 ZMQ_DEFINE_ARRAY_OPT(ZMQ_SOCKS_PASSWORD, socks_password);
01641 #endif
01642 #ifdef ZMQ_SOCKS_PROXY
01643 ZMQ_DEFINE_ARRAY_OPT(ZMQ_SOCKS_PROXY, socks_proxy);
01644 #endif
01645 #ifdef ZMQ_SOCKS_USERNAME
01646 ZMQ_DEFINE_ARRAY_OPT(ZMQ_SOCKS_USERNAME, socks_username);
01647 #endif
01648 #ifdef ZMQ_STREAM_NOTIFY
01649 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_STREAM_NOTIFY, stream_notify, int);
01650 #endif
01651 #ifdef ZMQ_SUBSCRIBE
01652 ZMQ_DEFINE_ARRAY_OPT(ZMQ_SUBSCRIBE, subscribe);
01653 #endif
01654 #ifdef ZMQ_TCP_KEEPALIVE
01655 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_KEEPALIVE, tcp_keepalive, int);
01656 #endif
01657 #ifdef ZMQ_TCP_KEEPALIVE_CNT
01658 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_KEEPALIVE_CNT, tcp_keepalive_cnt, int);
01659 #endif
01660 #ifdef ZMQ_TCP_KEEPALIVE_IDLE
01661 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_KEEPALIVE_IDLE, tcp_keepalive_idle, int);
01662 #endif
```

```
01663 #ifdef ZMQ_TCP_KEEPALIVE_INTVL
01664 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_KEEPALIVE_INTVL, tcp_keepalive_intvl, int);
01665 #endif
01666 #ifdef ZMQ_TCP_MAXRT
01667 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TCP_MAXRT, tcp_maxrt, int);
01668 #endif
01669 #ifdef ZMQ_THREAD_SAFE
01670 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_THREAD_SAFE, thread_safe, int);
01671 #endif
01672 #ifdef ZMQ_TOS
01673 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TOS, tos, int);
01674 #endif
01675 #ifdef ZMQ_TYPE
01676 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TYPE, type, int);
01677 #ifdef ZMQ_CPP11
01678 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_TYPE, socket_type, socket_type);
01679 #endif // ZMQ_CPP11
01680 #endif // ZMQ_TYPE
01681 #ifdef ZMQ_UNSUBSCRIBE
01682 ZMQ_DEFINE_ARRAY_OPT(ZMQ_UNSUBSCRIBE, unsubscribe);
01683 #endif
01684 #ifdef ZMQ_VMCI_BUFFER_SIZE
01685 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_VMCI_BUFFER_SIZE, vmci_buffer_size, uint64_t);
01686 #endif
01687 #ifdef ZMQ_VMCI_BUFFER_MIN_SIZE
01688 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_VMCI_BUFFER_MIN_SIZE, vmci_buffer_min_size, uint64_t);
01689 #endif
01690 #ifdef ZMQ_VMCI_BUFFER_MAX_SIZE
01691 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_VMCI_BUFFER_MAX_SIZE, vmci_buffer_max_size, uint64_t);
01692 #endif
01693 #ifdef ZMQ_VMCI_CONNECT_TIMEOUT
01694 ZMQ_DEFINE_INTEGRAL_OPT(ZMQ_VMCI_CONNECT_TIMEOUT, vmci_connect_timeout, int);
01695 #endif
01696 #ifdef ZMQ_XPUB_VERBOSE
01697 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_VERBOSE, xpub_verbose, int);
01698 #endif
01699 #ifdef ZMQ_XPUB_VERBOSER
01700 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_VERBOSER, xpub_verboser, int);
01701 #endif
01702 #ifdef ZMQ_XPUB_MANUAL
01703 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_MANUAL, xpub_manual, int);
01704 #endif
01705 #ifdef ZMQ_XPUB_MANUAL_LAST_VALUE
01706 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_MANUAL_LAST_VALUE, xpub_manual_last_value, int);
01707 #endif
01708 #ifdef ZMQ_XPUB_NODROP
01709 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_XPUB_NODROP, xpub_nodrop, int);
01710 #endif
01711 #ifdef ZMQ_XPUB_WELCOME_MSG
01712 ZMQ_DEFINE_ARRAY_OPT(ZMQ_XPUB_WELCOME_MSG, xpub_welcome_msg);
01713 #endif
01714 #ifdef ZMQ_ZAP_ENFORCE_DOMAIN
01715 ZMQ_DEFINE_INTEGRAL_BOOL_UNIT_OPT(ZMQ_ZAP_ENFORCE_DOMAIN, zap_enforce_domain, int);
01716 #endif
01717 #ifdef ZMQ_ZAP_DOMAIN
01718 ZMQ_DEFINE_ARRAY_OPT(ZMQ_ZAP_DOMAIN, zap_domain);
01719 #endif
01720
01721 } // namespace sockopt
01722 #endif // ZMQ_CPP11
01723
01724
01725 namespace detail
01726 {
01727 class socket_base
01728 {
01729   public:
01730     socket_base() ZMQ_NOTHROW : _handle(ZMQ_NULLPTR) {}
01731     ZMQ_EXPLICIT socket_base(void *handle) ZMQ_NOTHROW : _handle(handle) {}
01732
01733     template<typename T>
01734     ZMQ_CPP11_DEPRECATED("from 4.7.0, use `set` taking option from zmq::sockopt")
01735     void setsockopt(int option_, T const &optval)
01736     {
01737         setsockopt(option_, &optval, sizeof(T));
01738     }
01739
01740     ZMQ_CPP11_DEPRECATED("from 4.7.0, use `set` taking option from zmq::sockopt")
01741     void setsockopt(int option_, const void *optval_, size_t optvallen_)
01742     {
01743         int rc = zmq_setsockopt(_handle, option_, optval_, optvallen_);
01744         if (rc != 0)
01745             throw error_t();
01746     }
01747
01748     ZMQ_CPP11_DEPRECATED("from 4.7.0, use `get` taking option from zmq::sockopt")
01749     void getsockopt(int option_, void *optval_, size_t *optvallen_) const
```

```
01750     {
01751         int rc = zmq_getsockopt(_handle, option_, optval_, optvallen_);
01752         if (rc != 0)
01753             throw error_t();
01754     }
01755
01756     template<typename T>
01757     ZMQ_CPP11_DEPRECATED("from 4.7.0, use `get` taking option from zmq::sockopt")
01758     T getsockopt(int option_) const
01759     {
01760         T optval;
01761         size_t optlen = sizeof(T);
01762         getsockopt(option_, &optval, &optlen);
01763         return optval;
01764     }
01765
01766 #ifdef ZMQ_CPP11
01767     // Set integral socket option, e.g.
01768     // `socket.set(zmq::sockopt::linger, 0)`
01769     template<int Opt, class T, bool BoolUnit>
01770     void set(sockopt::integral_option<Opt, T, BoolUnit>, const T &val)
01771     {
01772         static_assert(std::is_integral<T>::value, "T must be integral");
01773         set_option(Opt, &val, sizeof val);
01774     }
01775
01776     // Set integral socket option from boolean, e.g.
01777     // `socket.set(zmq::sockopt::immediate, false)`
01778     template<int Opt, class T>
01779     void set(sockopt::integral_option<Opt, T, true>, bool val)
01780     {
01781         static_assert(std::is_integral<T>::value, "T must be integral");
01782         T rep_val = val;
01783         set_option(Opt, &rep_val, sizeof rep_val);
01784     }
01785
01786     // Set array socket option, e.g.
01787     // `socket.set(zmq::sockopt::plain_username, "foo123")`
01788     template<int Opt, int NullTerm>
01789     void set(sockopt::array_option<Opt, NullTerm>, const char *buf)
01790     {
01791         set_option(Opt, buf, std::strlen(buf));
01792     }
01793
01794     // Set array socket option, e.g.
01795     // `socket.set(zmq::sockopt::routing_id, zmq::buffer(id))`
01796     template<int Opt, int NullTerm>
01797     void set(sockopt::array_option<Opt, NullTerm>, const_buffer buf)
01798     {
01799         set_option(Opt, buf.data(), buf.size());
01800     }
01801
01802     // Set array socket option, e.g.
01803     // `socket.set(zmq::sockopt::routing_id, id_str)`
01804     template<int Opt, int NullTerm>
01805     void set(sockopt::array_option<Opt, NullTerm>, const std::string &buf)
01806     {
01807         set_option(Opt, buf.data(), buf.size());
01808     }
01809
01810 #if CPPZMQ_HAS_STRING_VIEW
01811     // Set array socket option, e.g.
01812     // `socket.set(zmq::sockopt::routing_id, id_str)`
01813     template<int Opt, int NullTerm>
01814     void set(sockopt::array_option<Opt, NullTerm>, std::string_view buf)
01815     {
01816         set_option(Opt, buf.data(), buf.size());
01817     }
01818 #endif
01819
01820     // Get scalar socket option, e.g.
01821     // `auto opt = socket.get(zmq::sockopt::linger)`
01822     template<int Opt, class T, bool BoolUnit>
01823     ZMQ_NODISCARD T get(sockopt::integral_option<Opt, T, BoolUnit>) const
01824     {
01825         static_assert(std::is_scalar<T>::value, "T must be scalar");
01826         T val;
01827         size_t size = sizeof val;
01828         get_option(Opt, &val, &size);
01829         assert(size == sizeof val);
01830         return val;
01831     }
01832
01833     // Get array socket option, writes to buf, returns option size in bytes, e.g.
01834     // `size_t optsize = socket.get(zmq::sockopt::routing_id, zmq::buffer(id))`
01835     template<int Opt, int NullTerm>
01836     ZMQ_NODISCARD size_t get(sockopt::array_option<Opt, NullTerm>,
```

```
01837                               mutable_buffer buf) const
01838     {
01839         size_t size = buf.size();
01840         get_option(Opt, buf.data(), &size);
01841         return size;
01842     }
01843
01844     // Get array socket option as string (initializes the string buffer size to init_size) e.g.
01845     // `auto s = socket.get(zmq::sockopt::routing_id)`
01846     // Note: removes the null character from null-terminated string options,
01847     // i.e. the string size excludes the null character.
01848     template<int Opt, int NullTerm>
01849     ZMQ_NODISCARD std::string get(sockopt::array_option<Opt, NullTerm>,
01850                                   size_t init_size = 1024) const
01851     {
01852         if ZMQ_CONSTEXPR_IF (NullTerm == 2) {
01853             if (init_size == 1024) {
01854                 init_size = 41; // get as Z85 string
01855             }
01856         }
01857         std::string str(init_size, '\0');
01858         size_t size = get(sockopt::array_option<Opt>{}, buffer(str));
01859         if ZMQ_CONSTEXPR_IF (NullTerm == 1) {
01860             if (size > 0) {
01861                 assert(str[size - 1] == '\0');
01862                 --size;
01863             }
01864         } else if ZMQ_CONSTEXPR_IF (NullTerm == 2) {
01865             assert(size == 32 || size == 41);
01866             if (size == 41) {
01867                 assert(str[size - 1] == '\0');
01868                 --size;
01869             }
01870         }
01871         str.resize(size);
01872         return str;
01873     }
01874 #endif
01875
01876     void bind(std::string const &addr) { bind(addr.c_str()); }
01877
01878     void bind(const char *addr_)
01879     {
01880         int rc = zmq_bind(_handle, addr_);
01881         if (rc != 0)
01882             throw error_t();
01883     }
01884
01885     void unbind(std::string const &addr) { unbind(addr.c_str()); }
01886
01887     void unbind(const char *addr_)
01888     {
01889         int rc = zmq_unbind(_handle, addr_);
01890         if (rc != 0)
01891             throw error_t();
01892     }
01893
01894     void connect(std::string const &addr) { connect(addr.c_str()); }
01895
01896     void connect(const char *addr_)
01897     {
01898         int rc = zmq_connect(_handle, addr_);
01899         if (rc != 0)
01900             throw error_t();
01901     }
01902
01903     void disconnect(std::string const &addr) { disconnect(addr.c_str()); }
01904
01905     void disconnect(const char *addr_)
01906     {
01907         int rc = zmq_disconnect(_handle, addr_);
01908         if (rc != 0)
01909             throw error_t();
01910     }
01911
01912     ZMQ_DEPRECATED("from 4.7.1, use handle() != nullptr or operator bool")
01913     bool connected() const ZMQ_NOTHROW { return (_handle != ZMQ_NULLPTR); }
01914
01915     ZMQ_CPP11_DEPRECATED("from 4.3.1, use send taking a const_buffer and send_flags")
01916     size_t send(const void *buf_, size_t len_, int flags_ = 0)
01917     {
01918         int nbytes = zmq_send(_handle, buf_, len_, flags_);
01919         if (nbytes >= 0)
01920             return static_cast<size_t>(nbytes);
01921         if (zmq_errno() == EAGAIN)
01922             return 0;
01923         throw error_t();
```

```
01924     }
01925
01926     ZMQ_CPP11_DEPRECATED("from 4.3.1, use send taking message_t and send_flags")
01927     bool send(message_t &msg_,
01928               int flags_ = 0) // default until removed
01929     {
01930         int nbytes = zmq_msg_send(msg_.handle(), _handle, flags_);
01931         if (nbytes >= 0)
01932             return true;
01933         if (zmq_errno() == EAGAIN)
01934             return false;
01935         throw error_t();
01936     }
01937
01938     template<typename T>
01939     ZMQ_CPP11_DEPRECATED(
01940       "from 4.4.1, use send taking message_t or buffer (for contiguous "
01941       "ranges), and send_flags")
01942     bool send(T first, T last, int flags_ = 0)
01943     {
01944         zmq::message_t msg(first, last);
01945         int nbytes = zmq_msg_send(msg.handle(), _handle, flags_);
01946         if (nbytes >= 0)
01947             return true;
01948         if (zmq_errno() == EAGAIN)
01949             return false;
01950         throw error_t();
01951     }
01952
01953 #ifdef ZMQ_HAS_RVALUE_REFS
01954     ZMQ_CPP11_DEPRECATED("from 4.3.1, use send taking message_t and send_flags")
01955     bool send(message_t &&msg_,
01956               int flags_ = 0) // default until removed
01957     {
01958 #ifdef ZMQ_CPP11
01959         return send(msg_, static_cast<send_flags>(flags_)).has_value();
01960 #else
01961         return send(msg_, flags_);
01962 #endif
01963     }
01964 #endif
01965
01966 #ifdef ZMQ_CPP11
01967     send_result_t send(const_buffer buf, send_flags flags = send_flags::none)
01968     {
01969         const int nbytes =
01970           zmq_send(_handle, buf.data(), buf.size(), static_cast<int>(flags));
01971         if (nbytes >= 0)
01972             return static_cast<size_t>(nbytes);
01973         if (zmq_errno() == EAGAIN)
01974             return {};
01975         throw error_t();
01976     }
01977
01978     send_result_t send(message_t &msg, send_flags flags)
01979     {
01980         int nbytes = zmq_msg_send(msg.handle(), _handle, static_cast<int>(flags));
01981         if (nbytes >= 0)
01982             return static_cast<size_t>(nbytes);
01983         if (zmq_errno() == EAGAIN)
01984             return {};
01985         throw error_t();
01986     }
01987
01988     send_result_t send(message_t &&msg, send_flags flags)
01989     {
01990         return send(msg, flags);
01991     }
01992 #endif
01993
01994     ZMQ_CPP11_DEPRECATED(
01995       "from 4.3.1, use recv taking a mutable_buffer and recv_flags")
01996     size_t recv(void *buf_, size_t len_, int flags_ = 0)
01997     {
01998         int nbytes = zmq_recv(_handle, buf_, len_, flags_);
01999         if (nbytes >= 0)
02000             return static_cast<size_t>(nbytes);
02001         if (zmq_errno() == EAGAIN)
02002             return 0;
02003         throw error_t();
02004     }
02005
02006     ZMQ_CPP11_DEPRECATED(
02007       "from 4.3.1, use recv taking a reference to message_t and recv_flags")
02008     bool recv(message_t *msg_, int flags_ = 0)
02009     {
02010         int nbytes = zmq_msg_recv(msg_->handle(), _handle, flags_);
```

```
02011         if (nbytes >= 0)
02012             return true;
02013         if (zmq_errno() == EAGAIN)
02014             return false;
02015         throw error_t();
02016     }
02017
02018 #ifdef ZMQ_CPP11
02019     ZMQ_NODISCARD
02020     recv_buffer_result_t recv(mutable_buffer buf,
02021                               recv_flags flags = recv_flags::none)
02022     {
02023         const int nbytes =
02024           zmq_recv(_handle, buf.data(), buf.size(), static_cast<int>(flags));
02025         if (nbytes >= 0) {
02026             return recv_buffer_size{
02027               (std::min)(static_cast<size_t>(nbytes), buf.size()),
02028               static_cast<size_t>(nbytes)};
02029         }
02030         if (zmq_errno() == EAGAIN)
02031             return {};
02032         throw error_t();
02033     }
02034
02035     ZMQ_NODISCARD
02036     recv_result_t recv(message_t &msg, recv_flags flags = recv_flags::none)
02037     {
02038         const int nbytes =
02039           zmq_msg_recv(msg.handle(), _handle, static_cast<int>(flags));
02040         if (nbytes >= 0) {
02041             assert(msg.size() == static_cast<size_t>(nbytes));
02042             return static_cast<size_t>(nbytes);
02043         }
02044         if (zmq_errno() == EAGAIN)
02045             return {};
02046         throw error_t();
02047     }
02048 #endif
02049
02050 #if defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 0)
02051     void join(const char *group)
02052     {
02053         int rc = zmq_join(_handle, group);
02054         if (rc != 0)
02055             throw error_t();
02056     }
02057
02058     void leave(const char *group)
02059     {
02060         int rc = zmq_leave(_handle, group);
02061         if (rc != 0)
02062             throw error_t();
02063     }
02064 #endif
02065
02066     ZMQ_NODISCARD void *handle() ZMQ_NOTHROW { return _handle; }
02067     ZMQ_NODISCARD const void *handle() const ZMQ_NOTHROW { return _handle; }
02068
02069     ZMQ_EXPLICIT operator bool() const ZMQ_NOTHROW { return _handle != ZMQ_NULLPTR; }
02070     // note: non-const operator bool can be removed once
02071     // operator void* is removed from socket_t
02072     ZMQ_EXPLICIT operator bool() ZMQ_NOTHROW { return _handle != ZMQ_NULLPTR; }
02073
02074   protected:
02075     void *_handle;
02076
02077   private:
02078     void set_option(int option_, const void *optval_, size_t optvallen_)
02079     {
02080         int rc = zmq_setsockopt(_handle, option_, optval_, optvallen_);
02081         if (rc != 0)
02082             throw error_t();
02083     }
02084
02085     void get_option(int option_, void *optval_, size_t *optvallen_) const
02086     {
02087         int rc = zmq_getsockopt(_handle, option_, optval_, optvallen_);
02088         if (rc != 0)
02089             throw error_t();
02090     }
02091 };
02092 } // namespace detail
02093
02094 struct from_handle_t
02095 {
02096     struct _private
02097     {
```

```
02098     }; // disabling use other than with from_handle
02099     ZMQ_CONSTEXPR_FN ZMQ_EXPLICIT from_handle_t(_private /*p*/) ZMQ_NOTHROW {}
02100 };
02101
02102 ZMQ_CONSTEXPR_VAR from_handle_t from_handle =
02103   from_handle_t(from_handle_t::_private());
02104
02105 // A non-owning nullable reference to a socket.
02106 // The reference is invalidated on socket close or destruction.
02107 class socket_ref : public detail::socket_base
02108 {
02109   public:
02110     socket_ref() ZMQ_NOTHROW : detail::socket_base() {}
02111 #ifdef ZMQ_CPP11
02112     socket_ref(std::nullptr_t) ZMQ_NOTHROW : detail::socket_base() {}
02113 #endif
02114     socket_ref(from_handle_t /*fh*/, void *handle) ZMQ_NOTHROW
02115         : detail::socket_base(handle)
02116     {
02117     }
02118 };
02119
02120 #ifdef ZMQ_CPP11
02121 inline bool operator==(socket_ref sr, std::nullptr_t /*p*/) ZMQ_NOTHROW
02122 {
02123     return sr.handle() == nullptr;
02124 }
02125 inline bool operator==(std::nullptr_t /*p*/, socket_ref sr) ZMQ_NOTHROW
02126 {
02127     return sr.handle() == nullptr;
02128 }
02129 inline bool operator!=(socket_ref sr, std::nullptr_t /*p*/) ZMQ_NOTHROW
02130 {
02131     return !(sr == nullptr);
02132 }
02133 inline bool operator!=(std::nullptr_t /*p*/, socket_ref sr) ZMQ_NOTHROW
02134 {
02135     return !(sr == nullptr);
02136 }
02137 #endif
02138
02139 inline bool operator==(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
02140 {
02141     return std::equal_to<const void *>()(a.handle(), b.handle());
02142 }
02143 inline bool operator!=(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
02144 {
02145     return !(a == b);
02146 }
02147 inline bool operator<(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
02148 {
02149     return std::less<const void *>()(a.handle(), b.handle());
02150 }
02151 inline bool operator>(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
02152 {
02153     return b < a;
02154 }
02155 inline bool operator<=(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
02156 {
02157     return !(a > b);
02158 }
02159 inline bool operator>=(const detail::socket_base& a, const detail::socket_base& b) ZMQ_NOTHROW
02160 {
02161     return !(a < b);
02162 }
02163
02164 } // namespace zmq
02165
02166 #ifdef ZMQ_CPP11
02167 namespace std
02168 {
02169 template<> struct hash<zmq::socket_ref>
02170 {
02171     size_t operator()(zmq::socket_ref sr) const ZMQ_NOTHROW
02172     {
02173         return hash<void *>()(sr.handle());
02174     }
02175 };
02176 } // namespace std
02177 #endif
02178
02179 namespace zmq
02180 {
02181 class socket_t : public detail::socket_base
02182 {
02183     friend class monitor_t;
02184
```

```
02185   public:
02186     socket_t() ZMQ_NOTHROW : detail::socket_base(ZMQ_NULLPTR), ctxptr(ZMQ_NULLPTR) {}
02187
02188     socket_t(context_t &context_, int type_) :
02189         detail::socket_base(zmq_socket(context_.handle(), type_)),
02190         ctxptr(context_.handle())
02191     {
02192         if (_handle == ZMQ_NULLPTR)
02193             throw error_t();
02194     }
02195
02196 #ifdef ZMQ_CPP11
02197     socket_t(context_t &context_, socket_type type_) :
02198         socket_t(context_, static_cast<int>(type_))
02199     {
02200     }
02201 #endif
02202
02203 #ifdef ZMQ_HAS_RVALUE_REFS
02204     socket_t(socket_t &&rhs) ZMQ_NOTHROW : detail::socket_base(rhs._handle),
02205                                            ctxptr(rhs.ctxptr)
02206     {
02207         rhs._handle = ZMQ_NULLPTR;
02208         rhs.ctxptr = ZMQ_NULLPTR;
02209     }
02210     socket_t &operator=(socket_t &&rhs) ZMQ_NOTHROW
02211     {
02212         close();
02213         std::swap(_handle, rhs._handle);
02214         std::swap(ctxptr, rhs.ctxptr);
02215         return *this;
02216     }
02217 #endif
02218
02219     ~socket_t() ZMQ_NOTHROW { close(); }
02220
02221     operator void *() ZMQ_NOTHROW { return _handle; }
02222
02223     operator void const *() const ZMQ_NOTHROW { return _handle; }
02224
02225     void close() ZMQ_NOTHROW
02226     {
02227         if (_handle == ZMQ_NULLPTR)
02228             // already closed
02229             return;
02230         int rc = zmq_close(_handle);
02231         ZMQ_ASSERT(rc == 0);
02232         _handle = ZMQ_NULLPTR;
02233         ctxptr = ZMQ_NULLPTR;
02234     }
02235
02236     void swap(socket_t &other) ZMQ_NOTHROW
02237     {
02238         std::swap(_handle, other._handle);
02239         std::swap(ctxptr, other.ctxptr);
02240     }
02241
02242     operator socket_ref() ZMQ_NOTHROW { return socket_ref(from_handle, _handle); }
02243
02244   private:
02245     void *ctxptr;
02246
02247     socket_t(const socket_t &) ZMQ_DELETED_FUNCTION;
02248     void operator=(const socket_t &) ZMQ_DELETED_FUNCTION;
02249
02250     // used by monitor_t
02251     socket_t(void *context_, int type_) :
02252         detail::socket_base(zmq_socket(context_, type_)), ctxptr(context_)
02253     {
02254         if (_handle == ZMQ_NULLPTR)
02255             throw error_t();
02256         if (ctxptr == ZMQ_NULLPTR)
02257             throw error_t();
02258     }
02259 };
02260
02261 inline void swap(socket_t &a, socket_t &b) ZMQ_NOTHROW
02262 {
02263     a.swap(b);
02264 }
02265
02266 ZMQ_DEPRECATED("from 4.3.1, use proxy taking socket_t objects")
02267 inline void proxy(void *frontend, void *backend, void *capture)
02268 {
02269     int rc = zmq_proxy(frontend, backend, capture);
02270     if (rc != 0)
02271         throw error_t();
```

```
02272 }
02273
02274 inline void
02275 proxy(socket_ref frontend, socket_ref backend, socket_ref capture = socket_ref())
02276 {
02277     int rc = zmq_proxy(frontend.handle(), backend.handle(), capture.handle());
02278     if (rc != 0)
02279         throw error_t();
02280 }
02281
02282 #ifdef ZMQ_HAS_PROXY_STEERABLE
02283 ZMQ_DEPRECATED("from 4.3.1, use proxy_steerable taking socket_t objects")
02284 inline void
02285 proxy_steerable(void *frontend, void *backend, void *capture, void *control)
02286 {
02287     int rc = zmq_proxy_steerable(frontend, backend, capture, control);
02288     if (rc != 0)
02289         throw error_t();
02290 }
02291
02292 inline void proxy_steerable(socket_ref frontend,
02293                             socket_ref backend,
02294                             socket_ref capture,
02295                             socket_ref control)
02296 {
02297     int rc = zmq_proxy_steerable(frontend.handle(), backend.handle(),
02298                                  capture.handle(), control.handle());
02299     if (rc != 0)
02300         throw error_t();
02301 }
02302 #endif
02303
02304 class monitor_t
02305 {
02306   public:
02307     monitor_t() : _socket(), _monitor_socket() {}
02308
02309     virtual ~monitor_t() { close(); }
02310
02311 #ifdef ZMQ_HAS_RVALUE_REFS
02312     monitor_t(monitor_t &&rhs) ZMQ_NOTHROW : _socket(), _monitor_socket()
02313     {
02314         std::swap(_socket, rhs._socket);
02315         std::swap(_monitor_socket, rhs._monitor_socket);
02316     }
02317
02318     monitor_t &operator=(monitor_t &&rhs) ZMQ_NOTHROW
02319     {
02320         close();
02321         _socket = socket_ref();
02322         std::swap(_socket, rhs._socket);
02323         std::swap(_monitor_socket, rhs._monitor_socket);
02324         return *this;
02325     }
02326 #endif
02327
02328
02329     void
02330     monitor(socket_t &socket, std::string const &addr, int events = ZMQ_EVENT_ALL)
02331     {
02332         monitor(socket, addr.c_str(), events);
02333     }
02334
02335     void monitor(socket_t &socket, const char *addr_, int events = ZMQ_EVENT_ALL)
02336     {
02337         init(socket, addr_, events);
02338         while (true) {
02339             check_event(-1);
02340         }
02341     }
02342
02343     void init(socket_t &socket, std::string const &addr, int events = ZMQ_EVENT_ALL)
02344     {
02345         init(socket, addr.c_str(), events);
02346     }
02347
02348     void init(socket_t &socket, const char *addr_, int events = ZMQ_EVENT_ALL)
02349     {
02350         int rc = zmq_socket_monitor(socket.handle(), addr_, events);
02351         if (rc != 0)
02352             throw error_t();
02353
02354         _socket = socket;
02355         _monitor_socket = socket_t(socket.ctxptr, ZMQ_PAIR);
02356         _monitor_socket.connect(addr_);
02357
02358         on_monitor_started();
```

```
02359     }
02360
02361     bool check_event(int timeout = 0)
02362     {
02363         assert(_monitor_socket);
02364
02365         zmq::message_t eventMsg;
02366
02367         zmq::pollitem_t items[] = {
02368           {_monitor_socket.handle(), 0, ZMQ_POLLIN, 0},
02369         };
02370
02371         #ifdef ZMQ_CPP11
02372         zmq::poll(&items[0], 1, std::chrono::milliseconds(timeout));
02373         #else
02374         zmq::poll(&items[0], 1, timeout);
02375         #endif
02376
02377         if (items[0].revents & ZMQ_POLLIN) {
02378             int rc = zmq_msg_recv(eventMsg.handle(), _monitor_socket.handle(), 0);
02379             if (rc == -1 && zmq_errno() == ETERM)
02380                 return false;
02381             assert(rc != -1);
02382
02383         } else {
02384             return false;
02385         }
02386
02387 #if ZMQ_VERSION_MAJOR >= 4
02388         const char *data = static_cast<const char *>(eventMsg.data());
02389         zmq_event_t msgEvent;
02390         memcpy(&msgEvent.event, data, sizeof(uint16_t));
02391         data += sizeof(uint16_t);
02392         memcpy(&msgEvent.value, data, sizeof(int32_t));
02393         zmq_event_t *event = &msgEvent;
02394 #else
02395         zmq_event_t *event = static_cast<zmq_event_t *>(eventMsg.data());
02396 #endif
02397
02398 #ifdef ZMQ_NEW_MONITOR_EVENT_LAYOUT
02399         zmq::message_t addrMsg;
02400         int rc = zmq_msg_recv(addrMsg.handle(), _monitor_socket.handle(), 0);
02401         if (rc == -1 && zmq_errno() == ETERM) {
02402             return false;
02403         }
02404
02405         assert(rc != -1);
02406         std::string address = addrMsg.to_string();
02407 #else
02408         // 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.
02409         std::string address = event->data.connected.addr;
02410 #endif
02411
02412 #ifdef ZMQ_EVENT_MONITOR_STOPPED
02413         if (event->event == ZMQ_EVENT_MONITOR_STOPPED) {
02414             return false;
02415         }
02416
02417 #endif
02418
02419         switch (event->event) {
02420             case ZMQ_EVENT_CONNECTED:
02421                 on_event_connected(*event, address.c_str());
02422                 break;
02423             case ZMQ_EVENT_CONNECT_DELAYED:
02424                 on_event_connect_delayed(*event, address.c_str());
02425                 break;
02426             case ZMQ_EVENT_CONNECT_RETRIED:
02427                 on_event_connect_retried(*event, address.c_str());
02428                 break;
02429             case ZMQ_EVENT_LISTENING:
02430                 on_event_listening(*event, address.c_str());
02431                 break;
02432             case ZMQ_EVENT_BIND_FAILED:
02433                 on_event_bind_failed(*event, address.c_str());
02434                 break;
02435             case ZMQ_EVENT_ACCEPTED:
02436                 on_event_accepted(*event, address.c_str());
02437                 break;
02438             case ZMQ_EVENT_ACCEPT_FAILED:
02439                 on_event_accept_failed(*event, address.c_str());
02440                 break;
02441             case ZMQ_EVENT_CLOSED:
02442                 on_event_closed(*event, address.c_str());
02443                 break;
02444             case ZMQ_EVENT_CLOSE_FAILED:
```

```
02445                 on_event_close_failed(*event, address.c_str());
02446                 break;
02447             case ZMQ_EVENT_DISCONNECTED:
02448                 on_event_disconnected(*event, address.c_str());
02449                 break;
02450 #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 3, 0) || (defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >=
    ZMQ_MAKE_VERSION(4, 2, 3))
02451             case ZMQ_EVENT_HANDSHAKE_FAILED_NO_DETAIL:
02452                 on_event_handshake_failed_no_detail(*event, address.c_str());
02453                 break;
02454             case ZMQ_EVENT_HANDSHAKE_FAILED_PROTOCOL:
02455                 on_event_handshake_failed_protocol(*event, address.c_str());
02456                 break;
02457             case ZMQ_EVENT_HANDSHAKE_FAILED_AUTH:
02458                 on_event_handshake_failed_auth(*event, address.c_str());
02459                 break;
02460             case ZMQ_EVENT_HANDSHAKE_SUCCEEDED:
02461                 on_event_handshake_succeeded(*event, address.c_str());
02462                 break;
02463 #elif defined(ZMQ_BUILD_DRAFT_API) && ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 1)
02464             case ZMQ_EVENT_HANDSHAKE_FAILED:
02465                 on_event_handshake_failed(*event, address.c_str());
02466                 break;
02467             case ZMQ_EVENT_HANDSHAKE_SUCCEED:
02468                 on_event_handshake_succeed(*event, address.c_str());
02469                 break;
02470 #endif
02471             default:
02472                 on_event_unknown(*event, address.c_str());
02473                 break;
02474         }
02475
02476         return true;
02477     }
02478
02479 #ifdef ZMQ_EVENT_MONITOR_STOPPED
02480     void abort()
02481     {
02482         if (_socket)
02483             zmq_socket_monitor(_socket.handle(), ZMQ_NULLPTR, 0);
02484
02485         _socket = socket_ref();
02486     }
02487 #endif
02488     virtual void on_monitor_started() {}
02489     virtual void on_event_connected(const zmq_event_t &event_, const char *addr_)
02490     {
02491         (void) event_;
02492         (void) addr_;
02493     }
02494     virtual void on_event_connect_delayed(const zmq_event_t &event_,
02495                                           const char *addr_)
02496     {
02497         (void) event_;
02498         (void) addr_;
02499     }
02500     virtual void on_event_connect_retried(const zmq_event_t &event_,
02501                                           const char *addr_)
02502     {
02503         (void) event_;
02504         (void) addr_;
02505     }
02506     virtual void on_event_listening(const zmq_event_t &event_, const char *addr_)
02507     {
02508         (void) event_;
02509         (void) addr_;
02510     }
02511     virtual void on_event_bind_failed(const zmq_event_t &event_, const char *addr_)
02512     {
02513         (void) event_;
02514         (void) addr_;
02515     }
02516     virtual void on_event_accepted(const zmq_event_t &event_, const char *addr_)
02517     {
02518         (void) event_;
02519         (void) addr_;
02520     }
02521     virtual void on_event_accept_failed(const zmq_event_t &event_, const char *addr_)
02522     {
02523         (void) event_;
02524         (void) addr_;
02525     }
02526     virtual void on_event_closed(const zmq_event_t &event_, const char *addr_)
02527     {
02528         (void) event_;
02529         (void) addr_;
02530     }
```

```
02531     virtual void on_event_close_failed(const zmq_event_t &event_, const char *addr_)
02532     {
02533         (void) event_;
02534         (void) addr_;
02535     }
02536     virtual void on_event_disconnected(const zmq_event_t &event_, const char *addr_)
02537     {
02538         (void) event_;
02539         (void) addr_;
02540     }
02541 #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 3)
02542     virtual void on_event_handshake_failed_no_detail(const zmq_event_t &event_,
02543                                                      const char *addr_)
02544     {
02545         (void) event_;
02546         (void) addr_;
02547     }
02548     virtual void on_event_handshake_failed_protocol(const zmq_event_t &event_,
02549                                                     const char *addr_)
02550     {
02551         (void) event_;
02552         (void) addr_;
02553     }
02554     virtual void on_event_handshake_failed_auth(const zmq_event_t &event_,
02555                                                 const char *addr_)
02556     {
02557         (void) event_;
02558         (void) addr_;
02559     }
02560     virtual void on_event_handshake_succeeded(const zmq_event_t &event_,
02561                                               const char *addr_)
02562     {
02563         (void) event_;
02564         (void) addr_;
02565     }
02566 #elif ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 1)
02567     virtual void on_event_handshake_failed(const zmq_event_t &event_,
02568                                            const char *addr_)
02569     {
02570         (void) event_;
02571         (void) addr_;
02572     }
02573     virtual void on_event_handshake_succeed(const zmq_event_t &event_,
02574                                             const char *addr_)
02575     {
02576         (void) event_;
02577         (void) addr_;
02578     }
02579 #endif
02580     virtual void on_event_unknown(const zmq_event_t &event_, const char *addr_)
02581     {
02582         (void) event_;
02583         (void) addr_;
02584     }
02585
02586   private:
02587     monitor_t(const monitor_t &) ZMQ_DELETED_FUNCTION;
02588     void operator=(const monitor_t &) ZMQ_DELETED_FUNCTION;
02589
02590     socket_ref _socket;
02591     socket_t _monitor_socket;
02592
02593     void close() ZMQ_NOTHROW
02594     {
02595         if (_socket)
02596             zmq_socket_monitor(_socket.handle(), ZMQ_NULLPTR, 0);
02597         _monitor_socket.close();
02598     }
02599 };
02600
02601 #if defined(ZMQ_BUILD_DRAFT_API) && defined(ZMQ_CPP11) && defined(ZMQ_HAVE_POLLER)
02602
02603 // polling events
02604 enum class event_flags : short
02605 {
02606     none = 0,
02607     pollin = ZMQ_POLLIN,
02608     pollout = ZMQ_POLLOUT,
02609     pollerr = ZMQ_POLLERR,
02610     pollpri = ZMQ_POLLPRI
02611 };
02612
02613 constexpr event_flags operator|(event_flags a, event_flags b) noexcept
02614 {
02615     return detail::enum_bit_or(a, b);
02616 }
02617 constexpr event_flags operator&(event_flags a, event_flags b) noexcept
```

```
02618 {
02619     return detail::enum_bit_and(a, b);
02620 }
02621 constexpr event_flags operator^(event_flags a, event_flags b) noexcept
02622 {
02623     return detail::enum_bit_xor(a, b);
02624 }
02625 constexpr event_flags operator~(event_flags a) noexcept
02626 {
02627     return detail::enum_bit_not(a);
02628 }
02629
02630 struct no_user_data;
02631
02632 // layout compatible with zmq_poller_event_t
02633 template<class T = no_user_data> struct poller_event
02634 {
02635     socket_ref socket;
02636     ::zmq::fd_t fd;
02637     T *user_data;
02638     event_flags events;
02639 };
02640
02641 template<typename T = no_user_data> class poller_t
02642 {
02643   public:
02644     using event_type = poller_event<T>;
02645
02646     poller_t() : poller_ptr(zmq_poller_new())
02647     {
02648         if (!poller_ptr)
02649             throw error_t();
02650     }
02651
02652     template<
02653       typename Dummy = void,
02654       typename =
02655         typename std::enable_if<!std::is_same<T, no_user_data>::value, Dummy>::type>
02656     void add(zmq::socket_ref socket, event_flags events, T *user_data)
02657     {
02658         add_impl(socket, events, user_data);
02659     }
02660
02661     void add(zmq::socket_ref socket, event_flags events)
02662     {
02663         add_impl(socket, events, nullptr);
02664     }
02665
02666     template<
02667       typename Dummy = void,
02668       typename =
02669         typename std::enable_if<!std::is_same<T, no_user_data>::value, Dummy>::type>
02670     void add(fd_t fd, event_flags events, T *user_data)
02671     {
02672         add_impl(fd, events, user_data);
02673     }
02674
02675     void add(fd_t fd, event_flags events) { add_impl(fd, events, nullptr); }
02676
02677     void remove(zmq::socket_ref socket)
02678     {
02679         if (0 != zmq_poller_remove(poller_ptr.get(), socket.handle())) {
02680             throw error_t();
02681         }
02682     }
02683
02684     void modify(zmq::socket_ref socket, event_flags events)
02685     {
02686         if (0
02687             != zmq_poller_modify(poller_ptr.get(), socket.handle(),
02688                                  static_cast<short>(events))) {
02689             throw error_t();
02690         }
02691     }
02692
02693     size_t wait_all(std::vector<event_type> &poller_events,
02694                     const std::chrono::milliseconds timeout)
02695     {
02696         int rc = zmq_poller_wait_all(
02697           poller_ptr.get(),
02698           reinterpret_cast<zmq_poller_event_t *>(poller_events.data()),
02699           static_cast<int>(poller_events.size()),
02700           static_cast<long>(timeout.count()));
02701         if (rc > 0)
02702             return static_cast<size_t>(rc);
02703
02704 #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 2, 3)
```

```
02705         if (zmq_errno() == EAGAIN)
02706 #else
02707         if (zmq_errno() == ETIMEDOUT)
02708 #endif
02709             return 0;
02710
02711         throw error_t();
02712     }
02713
02714 #if ZMQ_VERSION >= ZMQ_MAKE_VERSION(4, 3, 3)
02715     size_t size() const noexcept
02716     {
02717         int rc = zmq_poller_size(const_cast<void *>(poller_ptr.get()));
02718         ZMQ_ASSERT(rc >= 0);
02719         return static_cast<size_t>(std::max(rc, 0));
02720     }
02721 #endif
02722
02723   private:
02724     struct destroy_poller_t
02725     {
02726         void operator()(void *ptr) noexcept
02727         {
02728             int rc = zmq_poller_destroy(&ptr);
02729             ZMQ_ASSERT(rc == 0);
02730         }
02731     };
02732
02733     std::unique_ptr<void, destroy_poller_t> poller_ptr;
02734
02735     void add_impl(zmq::socket_ref socket, event_flags events, T *user_data)
02736     {
02737         if (0
02738             != zmq_poller_add(poller_ptr.get(), socket.handle(), user_data,
02739                               static_cast<short>(events))) {
02740             throw error_t();
02741         }
02742     }
02743
02744     void add_impl(fd_t fd, event_flags events, T *user_data)
02745     {
02746         if (0
02747             != zmq_poller_add_fd(poller_ptr.get(), fd, user_data,
02748                                  static_cast<short>(events))) {
02749             throw error_t();
02750         }
02751     }
02752 };
02753 #endif //  defined(ZMQ_BUILD_DRAFT_API) && defined(ZMQ_CPP11) && defined(ZMQ_HAVE_POLLER)
02754
02755 inline std::ostream &operator<<(std::ostream &os, const message_t &msg)
02756 {
02757     return os << msg.str();
02758 }
02759
02760 } // namespace zmq
02761
02762 #endif // __ZMQ_HPP_INCLUDED__
```

5.15 zmq_addon.hpp

```
00001 /*
00002     Copyright (c) 2016-2017 ZeroMQ community
00003     Copyright (c) 2016 VOCA AS / Harald Nøkland
00004
00005     Permission is hereby granted, free of charge, to any person obtaining a copy
00006     of this software and associated documentation files (the "Software"), to
00007     deal in the Software without restriction, including without limitation the
00008     rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
00009     sell copies of the Software, and to permit persons to whom the Software is
00010     furnished to do so, subject to the following conditions:
00011
00012     The above copyright notice and this permission notice shall be included in
00013     all copies or substantial portions of the Software.
00014
00015     THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00016     IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00017     FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00018     AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00019     LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
00020     FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
00021     IN THE SOFTWARE.
00022 */
00023
00024 #ifndef __ZMQ_ADDON_HPP_INCLUDED__
```

```
00025 #define __ZMQ_ADDON_HPP_INCLUDED__
00026
00027 #include "zmq.hpp"
00028
00029 #include <deque>
00030 #include <iomanip>
00031 #include <sstream>
00032 #include <stdexcept>
00033 #ifdef ZMQ_CPP11
00034 #include <limits>
00035 #include <functional>
00036 #include <unordered_map>
00037 #endif
00038
00039 namespace zmq
00040 {
00041 #ifdef ZMQ_CPP11
00042
00043 namespace detail
00044 {
00045 template<bool CheckN, class OutputIt>
00046 recv_result_t
00047 recv_multipart_n(socket_ref s, OutputIt out, size_t n, recv_flags flags)
00048 {
00049     size_t msg_count = 0;
00050     message_t msg;
00051     while (true) {
00052         if ZMQ_CONSTEXPR_IF (CheckN) {
00053             if (msg_count >= n)
00054                 throw std::runtime_error(
00055                   "Too many message parts in recv_multipart_n");
00056         }
00057         if (!s.recv(msg, flags)) {
00058             // zmq ensures atomic delivery of messages
00059             assert(msg_count == 0);
00060             return {};
00061         }
00062         ++msg_count;
00063         const bool more = msg.more();
00064         *out++ = std::move(msg);
00065         if (!more)
00066             break;
00067     }
00068     return msg_count;
00069 }
00070
00071 inline bool is_little_endian()
00072 {
00073     const uint16_t i = 0x01;
00074     return *reinterpret_cast<const uint8_t *>(&i) == 0x01;
00075 }
00076
00077 inline void write_network_order(unsigned char *buf, const uint32_t value)
00078 {
00079     if (is_little_endian()) {
00080         ZMQ_CONSTEXPR_VAR uint32_t mask = (std::numeric_limits<std::uint8_t>::max)();
00081         *buf++ = static_cast<unsigned char>((value » 24) & mask);
00082         *buf++ = static_cast<unsigned char>((value » 16) & mask);
00083         *buf++ = static_cast<unsigned char>((value » 8) & mask);
00084         *buf++ = static_cast<unsigned char>(value & mask);
00085     } else {
00086         std::memcpy(buf, &value, sizeof(value));
00087     }
00088 }
00089
00090 inline uint32_t read_u32_network_order(const unsigned char *buf)
00091 {
00092     if (is_little_endian()) {
00093         return (static_cast<uint32_t>(buf[0]) « 24)
00094                + (static_cast<uint32_t>(buf[1]) « 16)
00095                + (static_cast<uint32_t>(buf[2]) « 8)
00096                + static_cast<uint32_t>(buf[3]);
00097     } else {
00098         uint32_t value;
00099         std::memcpy(&value, buf, sizeof(value));
00100         return value;
00101     }
00102 }
00103 } // namespace detail
00104
00105 /*  Receive a multipart message.
00106
00107     Writes the zmq::message_t objects to OutputIterator out.
00108     The out iterator must handle an unspecified number of writes,
00109     e.g. by using std::back_inserter.
00110
00111     Returns: the number of messages received or nullopt (on EAGAIN).
```

```
00112     Throws: if recv throws. Any exceptions thrown
00113     by the out iterator will be propagated and the message
00114     may have been only partially received with pending
00115     message parts. It is adviced to close this socket in that event.
00116 */
00117 template<class OutputIt>
00118 ZMQ_NODISCARD recv_result_t recv_multipart(socket_ref s,
00119                                            OutputIt out,
00120                                            recv_flags flags = recv_flags::none)
00121 {
00122     return detail::recv_multipart_n<false>(s, std::move(out), 0, flags);
00123 }
00124
00125 /*  Receive a multipart message.
00126
00127     Writes at most n zmq::message_t objects to OutputIterator out.
00128     If the number of message parts of the incoming message exceeds n
00129     then an exception will be thrown.
00130
00131     Returns: the number of messages received or nullopt (on EAGAIN).
00132     Throws: if recv throws. Throws std::runtime_error if the number
00133     of message parts exceeds n (exactly n messages will have been written
00134     to out). Any exceptions thrown
00135     by the out iterator will be propagated and the message
00136     may have been only partially received with pending
00137     message parts. It is adviced to close this socket in that event.
00138 */
00139 template<class OutputIt>
00140 ZMQ_NODISCARD recv_result_t recv_multipart_n(socket_ref s,
00141                                              OutputIt out,
00142                                              size_t n,
00143                                              recv_flags flags = recv_flags::none)
00144 {
00145     return detail::recv_multipart_n<true>(s, std::move(out), n, flags);
00146 }
00147
00148 /*  Send a multipart message.
00149
00150     The range must be a ForwardRange of zmq::message_t,
00151     zmq::const_buffer or zmq::mutable_buffer.
00152     The flags may be zmq::send_flags::sndmore if there are
00153     more message parts to be sent after the call to this function.
00154
00155     Returns: the number of messages sent (exactly msgs.size()) or nullopt (on EAGAIN).
00156     Throws: if send throws. Any exceptions thrown
00157     by the msgs range will be propagated and the message
00158     may have been only partially sent. It is adviced to close this socket in that event.
00159 */
00160 template<class Range
00161 #ifndef ZMQ_CPP11_PARTIAL
00162          ,
00163          typename = typename std::enable_if<
00164            detail::is_range<Range>::value
00165            && (std::is_same<detail::range_value_t<Range>, message_t>::value
00166                || detail::is_buffer<detail::range_value_t<Range»::value)>::type
00167 #endif
00168          >
00169 send_result_t
00170 send_multipart(socket_ref s, Range &&msgs, send_flags flags = send_flags::none)
00171 {
00172     using std::begin;
00173     using std::end;
00174     auto it = begin(msgs);
00175     const auto end_it = end(msgs);
00176     size_t msg_count = 0;
00177     while (it != end_it) {
00178         const auto next = std::next(it);
00179         const auto msg_flags =
00180           flags | (next == end_it ? send_flags::none : send_flags::sndmore);
00181         if (!s.send(*it, msg_flags)) {
00182             // zmq ensures atomic delivery of messages
00183             assert(it == begin(msgs));
00184             return {};
00185         }
00186         ++msg_count;
00187         it = next;
00188     }
00189     return msg_count;
00190 }
00191
00192 /* Encode a multipart message.
00193
00194    The range must be a ForwardRange of zmq::message_t.  A
00195    zmq::multipart_t or STL container may be passed for encoding.
00196
00197    Returns: a zmq::message_t holding the encoded multipart data.
00198
```

```
00199    Throws: std::range_error is thrown if the size of any single part
00200    can not fit in an unsigned 32 bit integer.
00201
00202    The encoding is compatible with that used by the CZMQ function
00203    zmsg_encode(), see https://rfc.zeromq.org/spec/50/.
00204    Each part consists of a size followed by the data.
00205    These are placed contiguously into the output message.  A part of
00206    size less than 255 bytes will have a single byte size value.
00207    Larger parts will have a five byte size value with the first byte
00208    set to 0xFF and the remaining four bytes holding the size of the
00209    part's data.
00210 */
00211 template<class Range
00212 #ifndef ZMQ_CPP11_PARTIAL
00213          ,
00214          typename = typename std::enable_if<
00215            detail::is_range<Range>::value
00216            && (std::is_same<detail::range_value_t<Range>, message_t>::value
00217                || detail::is_buffer<detail::range_value_t<Range»::value)>::type
00218 #endif
00219          >
00220 message_t encode(const Range &parts)
00221 {
00222     size_t mmsg_size = 0;
00223
00224     // First pass check sizes
00225     for (const auto &part : parts) {
00226         const size_t part_size = part.size();
00227         if (part_size > (std::numeric_limits<std::uint32_t>::max)()) {
00228             // Size value must fit into uint32_t.
00229             throw std::range_error("Invalid size, message part too large");
00230         }
00231         const size_t count_size =
00232           part_size < (std::numeric_limits<std::uint8_t>::max)() ? 1 : 5;
00233         mmsg_size += part_size + count_size;
00234     }
00235
00236     message_t encoded(mmsg_size);
00237     unsigned char *buf = encoded.data<unsigned char>();
00238     for (const auto &part : parts) {
00239         const uint32_t part_size = static_cast<uint32_t>(part.size());
00240         const unsigned char *part_data =
00241           static_cast<const unsigned char *>(part.data());
00242
00243         if (part_size < (std::numeric_limits<std::uint8_t>::max)()) {
00244             // small part
00245             *buf++ = (unsigned char) part_size;
00246         } else {
00247             // big part
00248             *buf++ = (std::numeric_limits<uint8_t>::max)();
00249             detail::write_network_order(buf, part_size);
00250             buf += sizeof(part_size);
00251         }
00252         std::memcpy(buf, part_data, part_size);
00253         buf += part_size;
00254     }
00255
00256     assert(static_cast<size_t>(buf - encoded.data<unsigned char>()) == mmsg_size);
00257     return encoded;
00258 }
00259
00260 /*  Decode an encoded message to multiple parts.
00261
00262     The given output iterator must be a ForwardIterator to a container
00263     holding zmq::message_t such as a zmq::multipart_t or various STL
00264     containers.
00265
00266     Returns the ForwardIterator advanced once past the last decoded
00267     part.
00268
00269     Throws: a std::out_of_range is thrown if the encoded part sizes
00270     lead to exceeding the message data bounds.
00271
00272     The decoding assumes the message is encoded in the manner
00273     performed by zmq::encode(), see https://rfc.zeromq.org/spec/50/.
00274  */
00275 template<class OutputIt> OutputIt decode(const message_t &encoded, OutputIt out)
00276 {
00277     const unsigned char *source = encoded.data<unsigned char>();
00278     const unsigned char *const limit = source + encoded.size();
00279
00280     while (source < limit) {
00281         size_t part_size = *source++;
00282         if (part_size == (std::numeric_limits<std::uint8_t>::max)()) {
00283             if (static_cast<size_t>(limit - source) < sizeof(uint32_t)) {
00284                 throw std::out_of_range(
00285                   "Malformed encoding, overflow in reading size");
```

```
00286             }
00287             part_size = detail::read_u32_network_order(source);
00288             // the part size is allowed to be less than 0xFF
00289             source += sizeof(uint32_t);
00290         }
00291
00292         if (static_cast<size_t>(limit - source) < part_size) {
00293             throw std::out_of_range("Malformed encoding, overflow in reading part");
00294         }
00295         *out = message_t(source, part_size);
00296         ++out;
00297         source += part_size;
00298     }
00299
00300     assert(source == limit);
00301     return out;
00302 }
00303
00304 #endif
00305
00306
00307 #ifdef ZMQ_HAS_RVALUE_REFS
00308
00309 /*
00310     This class handles multipart messaging. It is the C++ equivalent of zmsg.h,
00311     which is part of CZMQ (the high-level C binding). Furthermore, it is a major
00312     improvement compared to zmsg.hpp, which is part of the examples in the ØMQ
00313     Guide. Unnecessary copying is avoided by using move semantics to efficiently
00314     add/remove parts.
00315 */
00316 class multipart_t
00317 {
00318   private:
00319     std::deque<message_t> m_parts;
00320
00321   public:
00322     typedef std::deque<message_t>::value_type value_type;
00323
00324     typedef std::deque<message_t>::iterator iterator;
00325     typedef std::deque<message_t>::const_iterator const_iterator;
00326
00327     typedef std::deque<message_t>::reverse_iterator reverse_iterator;
00328     typedef std::deque<message_t>::const_reverse_iterator const_reverse_iterator;
00329
00330     // Default constructor
00331     multipart_t() {}
00332
00333     // Construct from socket receive
00334     multipart_t(socket_ref socket) { recv(socket); }
00335
00336     // Construct from memory block
00337     multipart_t(const void *src, size_t size) { addmem(src, size); }
00338
00339     // Construct from string
00340     multipart_t(const std::string &string) { addstr(string); }
00341
00342     // Construct from message part
00343     multipart_t(message_t &&message) { add(std::move(message)); }
00344
00345     // Move constructor
00346     multipart_t(multipart_t &&other) ZMQ_NOTHROW { m_parts = std::move(other.m_parts); }
00347
00348     // Move assignment operator
00349     multipart_t &operator=(multipart_t &&other) ZMQ_NOTHROW
00350     {
00351         m_parts = std::move(other.m_parts);
00352         return *this;
00353     }
00354
00355     // Destructor
00356     virtual ~multipart_t() { clear(); }
00357
00358     message_t &operator[](size_t n) { return m_parts[n]; }
00359
00360     const message_t &operator[](size_t n) const { return m_parts[n]; }
00361
00362     message_t &at(size_t n) { return m_parts.at(n); }
00363
00364     const message_t &at(size_t n) const { return m_parts.at(n); }
00365
00366     iterator begin() { return m_parts.begin(); }
00367
00368     const_iterator begin() const { return m_parts.begin(); }
00369
00370     const_iterator cbegin() const { return m_parts.cbegin(); }
00371
00372     reverse_iterator rbegin() { return m_parts.rbegin(); }
```

```
00373
00374     const_reverse_iterator rbegin() const { return m_parts.rbegin(); }
00375
00376     iterator end() { return m_parts.end(); }
00377
00378     const_iterator end() const { return m_parts.end(); }
00379
00380     const_iterator cend() const { return m_parts.cend(); }
00381
00382     reverse_iterator rend() { return m_parts.rend(); }
00383
00384     const_reverse_iterator rend() const { return m_parts.rend(); }
00385
00386     // Delete all parts
00387     void clear() { m_parts.clear(); }
00388
00389     // Get number of parts
00390     size_t size() const { return m_parts.size(); }
00391
00392     // Check if number of parts is zero
00393     bool empty() const { return m_parts.empty(); }
00394
00395     // Receive multipart message from socket
00396     bool recv(socket_ref socket, int flags = 0)
00397     {
00398         clear();
00399         bool more = true;
00400         while (more) {
00401             message_t message;
00402 #ifdef ZMQ_CPP11
00403             if (!socket.recv(message, static_cast<recv_flags>(flags)))
00404                 return false;
00405 #else
00406             if (!socket.recv(&message, flags))
00407                 return false;
00408 #endif
00409             more = message.more();
00410             add(std::move(message));
00411         }
00412         return true;
00413     }
00414
00415     // Send multipart message to socket
00416     bool send(socket_ref socket, int flags = 0)
00417     {
00418         flags &= ~(ZMQ_SNDMORE);
00419         bool more = size() > 0;
00420         while (more) {
00421             message_t message = pop();
00422             more = size() > 0;
00423 #ifdef ZMQ_CPP11
00424             if (!socket.send(message, static_cast<send_flags>(
00425                                       (more ? ZMQ_SNDMORE : 0) | flags)))
00426                 return false;
00427 #else
00428             if (!socket.send(message, (more ? ZMQ_SNDMORE : 0) | flags))
00429                 return false;
00430 #endif
00431         }
00432         clear();
00433         return true;
00434     }
00435
00436     // Concatenate other multipart to front
00437     void prepend(multipart_t &&other)
00438     {
00439         while (!other.empty())
00440             push(other.remove());
00441     }
00442
00443     // Concatenate other multipart to back
00444     void append(multipart_t &&other)
00445     {
00446         while (!other.empty())
00447             add(other.pop());
00448     }
00449
00450     // Push memory block to front
00451     void pushmem(const void *src, size_t size)
00452     {
00453         m_parts.push_front(message_t(src, size));
00454     }
00455
00456     // Push memory block to back
00457     void addmem(const void *src, size_t size)
00458     {
00459         m_parts.push_back(message_t(src, size));
```

```
00460     }
00461
00462     // Push string to front
00463     void pushstr(const std::string &string)
00464     {
00465         m_parts.push_front(message_t(string.data(), string.size()));
00466     }
00467
00468     // Push string to back
00469     void addstr(const std::string &string)
00470     {
00471         m_parts.push_back(message_t(string.data(), string.size()));
00472     }
00473
00474     // Push type (fixed-size) to front
00475     template<typename T> void pushtyp(const T &type)
00476     {
00477         static_assert(!std::is_same<T, std::string>::value,
00478                       "Use pushstr() instead of pushtyp<std::string>()");
00479         m_parts.push_front(message_t(&type, sizeof(type)));
00480     }
00481
00482     // Push type (fixed-size) to back
00483     template<typename T> void addtyp(const T &type)
00484     {
00485         static_assert(!std::is_same<T, std::string>::value,
00486                       "Use addstr() instead of addtyp<std::string>()");
00487         m_parts.push_back(message_t(&type, sizeof(type)));
00488     }
00489
00490     // Push message part to front
00491     void push(message_t &&message) { m_parts.push_front(std::move(message)); }
00492
00493     // Push message part to back
00494     void add(message_t &&message) { m_parts.push_back(std::move(message)); }
00495
00496     // Alias to allow std::back_inserter()
00497     void push_back(message_t &&message) { m_parts.push_back(std::move(message)); }
00498
00499     // Pop string from front
00500     std::string popstr()
00501     {
00502         std::string string(m_parts.front().data<char>(), m_parts.front().size());
00503         m_parts.pop_front();
00504         return string;
00505     }
00506
00507     // Pop type (fixed-size) from front
00508     template<typename T> T poptyp()
00509     {
00510         static_assert(!std::is_same<T, std::string>::value,
00511                       "Use popstr() instead of poptyp<std::string>()");
00512         if (sizeof(T) != m_parts.front().size())
00513             throw std::runtime_error(
00514               "Invalid type, size does not match the message size");
00515         T type = *m_parts.front().data<T>();
00516         m_parts.pop_front();
00517         return type;
00518     }
00519
00520     // Pop message part from front
00521     message_t pop()
00522     {
00523         message_t message = std::move(m_parts.front());
00524         m_parts.pop_front();
00525         return message;
00526     }
00527
00528     // Pop message part from back
00529     message_t remove()
00530     {
00531         message_t message = std::move(m_parts.back());
00532         m_parts.pop_back();
00533         return message;
00534     }
00535
00536     // get message part from front
00537     const message_t &front() { return m_parts.front(); }
00538
00539     // get message part from back
00540     const message_t &back() { return m_parts.back(); }
00541
00542     // Get pointer to a specific message part
00543     const message_t *peek(size_t index) const { return &m_parts[index]; }
00544
00545     // Get a string copy of a specific message part
00546     std::string peekstr(size_t index) const
```

```
00547     {
00548         std::string string(m_parts[index].data<char>(), m_parts[index].size());
00549         return string;
00550     }
00551
00552     // Peek type (fixed-size) from front
00553     template<typename T> T peektyp(size_t index) const
00554     {
00555         static_assert(!std::is_same<T, std::string>::value,
00556                       "Use peekstr() instead of peektyp<std::string>()");
00557         if (sizeof(T) != m_parts[index].size())
00558             throw std::runtime_error(
00559               "Invalid type, size does not match the message size");
00560         T type = *m_parts[index].data<T>();
00561         return type;
00562     }
00563
00564     // Create multipart from type (fixed-size)
00565     template<typename T> static multipart_t create(const T &type)
00566     {
00567         multipart_t multipart;
00568         multipart.addtyp(type);
00569         return multipart;
00570     }
00571
00572     // Copy multipart
00573     multipart_t clone() const
00574     {
00575         multipart_t multipart;
00576         for (size_t i = 0; i < size(); i++)
00577             multipart.addmem(m_parts[i].data(), m_parts[i].size());
00578         return multipart;
00579     }
00580
00581     // Dump content to string
00582     std::string str() const
00583     {
00584         std::stringstream ss;
00585         for (size_t i = 0; i < m_parts.size(); i++) {
00586             const unsigned char *data = m_parts[i].data<unsigned char>();
00587             size_t size = m_parts[i].size();
00588
00589             // Dump the message as text or binary
00590             bool isText = true;
00591             for (size_t j = 0; j < size; j++) {
00592                 if (data[j] < 32 || data[j] > 127) {
00593                     isText = false;
00594                     break;
00595                 }
00596             }
00597             ss « "\n[" « std::dec « std::setw(3) « std::setfill('0') « size
00598                « "] ";
00599             if (size >= 1000) {
00600                 ss « "... (too big to print)";
00601                 continue;
00602             }
00603             for (size_t j = 0; j < size; j++) {
00604                 if (isText)
00605                     ss « static_cast<char>(data[j]);
00606                 else
00607                     ss « std::hex « std::setw(2) « std::setfill('0')
00608                        « static_cast<short>(data[j]);
00609             }
00610         }
00611         return ss.str();
00612     }
00613
00614     // Check if equal to other multipart
00615     bool equal(const multipart_t *other) const ZMQ_NOTHROW
00616     {
00617         return *this == *other;
00618     }
00619
00620     bool operator==(const multipart_t &other) const ZMQ_NOTHROW
00621     {
00622         if (size() != other.size())
00623             return false;
00624         for (size_t i = 0; i < size(); i++)
00625             if (at(i) != other.at(i))
00626                 return false;
00627         return true;
00628     }
00629
00630     bool operator!=(const multipart_t &other) const ZMQ_NOTHROW
00631     {
00632         return !(*this == other);
00633     }
```

```
00634
00635 #ifdef ZMQ_CPP11
00636
00637     // Return single part message_t encoded from this multipart_t.
00638     message_t encode() const { return zmq::encode(*this); }
00639
00640     // Decode encoded message into multiple parts and append to self.
00641     void decode_append(const message_t &encoded)
00642     {
00643         zmq::decode(encoded, std::back_inserter(*this));
00644     }
00645
00646     // Return a new multipart_t containing the decoded message_t.
00647     static multipart_t decode(const message_t &encoded)
00648     {
00649         multipart_t tmp;
00650         zmq::decode(encoded, std::back_inserter(tmp));
00651         return tmp;
00652     }
00653
00654 #endif
00655
00656   private:
00657     // Disable implicit copying (moving is more efficient)
00658     multipart_t(const multipart_t &other) ZMQ_DELETED_FUNCTION;
00659     void operator=(const multipart_t &other) ZMQ_DELETED_FUNCTION;
00660 }; // class multipart_t
00661
00662 inline std::ostream &operator<<(std::ostream &os, const multipart_t &msg)
00663 {
00664     return os << msg.str();
00665 }
00666
00667 #endif // ZMQ_HAS_RVALUE_REFS
00668
00669 #if defined(ZMQ_BUILD_DRAFT_API) && defined(ZMQ_CPP11) && defined(ZMQ_HAVE_POLLER)
00670 class active_poller_t
00671 {
00672   public:
00673     active_poller_t() = default;
00674     ~active_poller_t() = default;
00675
00676     active_poller_t(const active_poller_t &) = delete;
00677     active_poller_t &operator=(const active_poller_t &) = delete;
00678
00679     active_poller_t(active_poller_t &&src) = default;
00680     active_poller_t &operator=(active_poller_t &&src) = default;
00681
00682     using handler_type = std::function<void(event_flags)>;
00683
00684     void add(zmq::socket_ref socket, event_flags events, handler_type handler)
00685     {
00686         if (!handler)
00687             throw std::invalid_argument("null handler in active_poller_t::add");
00688         auto ret = handlers.emplace(
00689           socket, std::make_shared<handler_type>(std::move(handler)));
00690         if (!ret.second)
00691             throw error_t(EINVAL); // already added
00692         try {
00693             base_poller.add(socket, events, ret.first->second.get());
00694             need_rebuild = true;
00695         }
00696         catch (...) {
00697             // rollback
00698             handlers.erase(socket);
00699             throw;
00700         }
00701     }
00702
00703     void remove(zmq::socket_ref socket)
00704     {
00705         base_poller.remove(socket);
00706         handlers.erase(socket);
00707         need_rebuild = true;
00708     }
00709
00710     void modify(zmq::socket_ref socket, event_flags events)
00711     {
00712         base_poller.modify(socket, events);
00713     }
00714
00715     size_t wait(std::chrono::milliseconds timeout)
00716     {
00717         if (need_rebuild) {
00718             poller_events.resize(handlers.size());
00719             poller_handlers.clear();
00720             poller_handlers.reserve(handlers.size());
```

```
00721             for (const auto &handler : handlers) {
00722                 poller_handlers.push_back(handler.second);
00723             }
00724             need_rebuild = false;
00725         }
00726         const auto count = base_poller.wait_all(poller_events, timeout);
00727         std::for_each(poller_events.begin(),
00728                       poller_events.begin() + static_cast<ptrdiff_t>(count),
00729                       [](decltype(base_poller)::event_type &event) {
00730                           assert(event.user_data != nullptr);
00731                           (*event.user_data)(event.events);
00732                       });
00733         return count;
00734     }
00735
00736     ZMQ_NODISCARD bool empty() const noexcept { return handlers.empty(); }
00737
00738     size_t size() const noexcept { return handlers.size(); }
00739
00740   private:
00741     bool need_rebuild{false};
00742
00743     poller_t<handler_type> base_poller{};
00744     std::unordered_map<socket_ref, std::shared_ptr<handler_type>> handlers{};
00745     std::vector<decltype(base_poller)::event_type> poller_events{};
00746     std::vector<std::shared_ptr<handler_type>> poller_handlers{};
00747 };     // class active_poller_t
00748 #endif //  defined(ZMQ_BUILD_DRAFT_API) && defined(ZMQ_CPP11) && defined(ZMQ_HAVE_POLLER)
00749
00750
00751 } // namespace zmq
00752
00753 #endif // __ZMQ_ADDON_HPP_INCLUDED__
```

5.16 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

- struct zmqutils::CommandData
- class zmqutils::CommandClientBase

5.16.1 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.

5.17 command_client.h

Go to the documentation of this file.

```
00001
/*****************************************************************************************************************************
00002  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
*
00003  *
*
00004  *   Copyright (C) 2023 Degoras Project Team
*
00005  *                      < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
*
00006  *                      < Jesús Relinque Madroñal >
*
00007  *
*
00008  *   This file is part of LibZMQUtils.
*
00009  *
*
00010  *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
the EUPL license   *
00011  *   as soon they will be approved by the European Commission (IDABC).
*
00012  *
*
00013  *   This project is free software: you can redistribute it and/or modify it under the terms of the
EUPL license as    *
00014  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
*
00015  *
*
00016  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
or agreed to in *
00017  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
without even the  *
00018  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
check specific   *
00019  *   language governing permissions and limitations and more details.
*
00020  *
*
00021  *   You should use this project in compliance with the EUPL license. You should have received a copy
of the license   *
00022  *   along with this project. If not, see the license at < https://eupl.eu/ >.
*
00023
*****************************************************************************************************************************/
00024
00033 //
==========================================================================================================================
00034 #pragma once
00035 //
==========================================================================================================================
00036
00037 // C++ INCLUDES
00038 //
==========================================================================================================================
00039 #include <future>
00040 #include <map>
00041 //
==========================================================================================================================
00042
00043 // ZMQUTILS INCLUDES
00044 //
==========================================================================================================================
00045 #include "LibZMQUtils/libzmqutils_global.h"
00046 #include "LibZMQUtils/CommandServerClient/common.h"
00047 //
==========================================================================================================================
00048
00049 // ZMQ DECLARATIONS
00050 //
==========================================================================================================================
00051 namespace zmq
00052 {
00053     class context_t;
00054     class socket_t;
00055 }
00056 //
==========================================================================================================================
00057
00058 // ZMQUTILS NAMESPACES
00059 //
==========================================================================================================================
```

```
00060 namespace zmqutils{
00061 //
      ================================================================================================================
00062
00063 using common::CmdRequestId;
00064 using common::BaseServerCommand;
00065 using common::CommandType;
00066
00067 struct LIBZMQUTILS_EXPORT CommandData
00068 {
00069     CommandData(CommandType id) :
00070         command_id(id),
00071         params(nullptr),
00072         params_size(0){}
00073
00074     CommandType command_id;
00075     std::unique_ptr<std::uint8_t> params;
00076     size_t params_size;
00077 };
00078
00079 class LIBZMQUTILS_EXPORT CommandClientBase
00080 {
00081
00082 public:
00083
00084
00085     // TODO: maybe this should be configurable
00086     static const int kClientAliveTimeoutMsec;
00087     static const int kClientSendAlivePeriodMsec;
00088
00089     enum class CommandError : std::uint32_t
00090     {
00091         NOT_ERROR,
00092         NO_COMMAND,
00093         NOT_CONNECTED,
00094         ALREADY_DISCONNECTED,
00095         ALREADY_CONNECTED,
00096         BAD_PARAMETERS,
00097         COMMAND_FAILED,
00098         NOT_IMPLEMENTED
00099     };
00100
00101
00102     CommandClientBase(const std::string &server_endpoint);
00103
00104     virtual ~CommandClientBase();
00105
00106     bool startClient(const std::string& interface_name);
00107     void stopClient();
00108     void resetClient();
00109
00110     void startAutoAlive();
00111     void stopAutoAlive();
00112
00113     void setClientHostIP(const std::string& interf);
00114
00115     void setClientId(const std::string &id);
00116
00117     virtual int sendCommand(const CommandData& msg, void* &data_out, size_t &out_bytes);
00118
00119     // Remove. Only for testing.
00120     int sendBadCommand1(void* &data_out, size_t &out_bytes);
00121
00122 private:
00123
00124     int recvFromSocket(zmq::socket_t *socket, void *&data, size_t &data_size_bytes) const;
00125     void sendAliveCallback();
00126     zmq::multipart_t prepareMessage(const CommandData &msg);
00127
00128     // Internal client identification.
00129     common::HostClient client_info_;
00130
00131     // Server endpoint.
00132     std::string server_endpoint_;
00133
00134     // ZMQ context and socket.
00135     zmq::context_t *context_;
00136     zmq::socket_t *socket_;
00137
00138     // Mutex.
00139     std::mutex mtx_;
00140
00141     std::future<void> auto_alive_future_;
00142     std::condition_variable auto_alive_cv_;
00143     std::atomic_bool auto_alive_working_;
00144
00145 };
```

```
00146
00147 } // END NAMESPACES.
00148 //
      ========================================================================================================================
```

5.18 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

5.18.1 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.

5.19 command_server.h

Go to the documentation of this file.

```
00001
      /*************************************************************************************************************
00002  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
      *
00003  *
      *
00004  *   Copyright (C) 2023 Degoras Project Team
      *
00005  *                      < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
      *
00006  *                      < Jesús Relinque Madroñal >
      *
00007  *
      *
00008  *   This file is part of LibZMQUtils.
      *
00009  *
      *
00010  *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
      the EUPL license   *
00011  *   as soon they will be approved by the European Commission (IDABC).
      *
00012  *
      *
```

```
00013  *   This project is free software: you can redistribute it and/or modify it under the terms of the
       EUPL license as    *
00014  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
       *
00015  *
       *
00016  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
       or agreed to in *
00017  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
       without even the  *
00018  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
       check specific   *
00019  *   language governing permissions and limitations and more details.
       *
00020  *
       *
00021  *   You should use this project in compliance with the EUPL license. You should have received a copy
       of the license   *
00022  *   along with this project. If not, see the license at < https://eupl.eu/ >.
       *
00023
       ********************************************************************************************************************************/
00024
00033 //
       ============================================================================================================================
00034 #pragma once
00035 //
       ============================================================================================================================
00036
00037 // C++ INCLUDES
00038 //
       ============================================================================================================================
00039 #include <future>
00040 #include <map>
00041 #include <zmq/zmq.hpp>
00042 #include <zmq/zmq_addon.hpp>
00043 //
       ============================================================================================================================
00044
00045 // ZMQUTILS INCLUDES
00046 //
       ============================================================================================================================
00047 #include "LibZMQUtils/libzmqutils_global.h"
00048 #include "LibZMQUtils/CommandServerClient/common.h"
00049 #include "LibZMQUtils/utils.h"
00050 //
       ============================================================================================================================
00051
00052 // ZMQ DECLARATIONS
00053 //
       ============================================================================================================================
00054 namespace zmq
00055 {
00056     class context_t;
00057     class socket_t;
00058 }
00059 //
       ============================================================================================================================
00060
00061 // ZMQUTILS NAMESPACES
00062 //
       ============================================================================================================================
00063 namespace zmqutils{
00064 //
       ============================================================================================================================
00065
00066 //
       ============================================================================================================================
00067 using common::BaseServerResultStr;
00068 using common::CommandReply;
00069 using common::CommandRequest;
00070 using common::CmdRequestId;
00071 using common::CmdReplyRes;
00072 using common::BaseServerCommand;
00073 using common::BaseServerResult;
00074 using common::HostClient;
00075 using utils::NetworkAdapterInfo;
00076 //
       ============================================================================================================================
00077
00078
00079 class LIBZMQUTILS_EXPORT CommandServerBase
00080 {
00081
00082 public:
00083
00105     CommandServerBase(unsigned port, const std::string &local_addr = "*");
```

```
00106
00111     const unsigned& getServerPort() const;
00112
00121     const std::vector<NetworkAdapterInfo> &getServerAddresses() const;
00122
00131     const std::string& getServerEndpoint() const;
00132
00142     const std::future<void>& getServerWorkerFuture() const;
00143
00153     const std::map<std::string, HostClient>& getConnectedClients() const;
00154
00164     bool isWorking() const{return this->server_working_;}
00165
00178     void setClientStatusCheck(bool);
00179
00186     void startServer();
00187
00194     void stopServer();
00195
00201     virtual ~CommandServerBase();
00202
00203 protected:
00204
00214     virtual void onServerStop() = 0;
00215
00225     virtual void onServerStart() = 0;
00226
00240     virtual void onWaitingCommand() = 0;
00241
00253     virtual void onConnected(const HostClient&) = 0;
00254
00266     virtual void onDisconnected(const HostClient&) = 0;
00267
00279     virtual void onDeadClient(const HostClient&) = 0;
00280
00292     virtual void onInvalidMsgReceived(const CommandRequest&) = 0;
00293
00308     virtual void onCommandReceived(const CommandRequest&) = 0;
00309
00326     virtual void onCustomCommandReceived(const CommandRequest&, CommandReply&);
00327
00347     virtual void onServerError(const zmq::error_t &error, const std::string& ext_info = "") = 0;
00348
00360     virtual void onSendingResponse(const CommandReply&) = 0;
00361
00362 private:
00363
00364     // Helper for prepare the result message.
00365     static void prepareCommandResult(BaseServerResult, std::unique_ptr<uint8_t>& data_out);
00366
00367     // Helper for check if the base command is valid.
00368     static bool validateCommand(int raw_command);
00369
00370     // Server worker. Will be execute asynchronously.
00371     void serverWorker();
00372
00373     // Process command class.
00374     void processCommand(const CommandRequest&, CommandReply&);
00375
00376     // Client status checker.
00377     void checkClientsAliveStatus();
00378
00379     // Update client last connection.
00380     void updateClientLastConnection(const std::string& id);
00381
00382     // Update the server timeout.
00383     void updateServerTimeout();
00384
00385     // Internal connect execution process.
00386     BaseServerResult execReqConnect(const CommandRequest&);
00387
00388     // Internal disconnect execution process.
00389     BaseServerResult execReqDisconnect(const CommandRequest&);
00390
00391     // Function for receive data from the client.
00392     BaseServerResult recvFromSocket(CommandRequest&);
00393
00394     // Function for reset the socket.
00395     void resetSocket();
00396
00397     // ZMQ socket and context.
00398     zmq::context_t *context_;
00399     zmq::socket_t* main_socket_;
00400
00401     // Endpoint data.
00402     std::string server_endpoint_;
00403     std::vector<utils::NetworkAdapterInfo> server_listen_adapters_;
```

```
00404     unsigned server_port_;
00405
00406     // Mutex.
00407     std::mutex mtx_;
00408
00409     // Future for the server worker.
00410     std::future<void> server_worker_future_;
00411
00412     // Clients container.
00413     std::map<std::string, HostClient> connected_clients_;
00414
00415     // Usefull flags.
00416     std::atomic_bool server_working_;
00417     std::atomic_bool check_clients_alive_;
00418 };
00419
00420 } // END NAMESPACES.
00421 //
      ================================================================================================================================
```

5.20 libzmqutils_global.h

```
00001
      /*****************************************************************************************************************************
00002  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
      *
00003  *
      *
00004  *   Copyright (C) 2023 Degoras Project Team
      *
00005  *                      < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
      *
00006  *                      < Jesús Relinque Madroñal >
      *
00007  *
      *
00008  *   This file is part of LibZMQUtils.
      *
00009  *
      *
00010  *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
      the EUPL license   *
00011  *   as soon they will be approved by the European Commission (IDABC).
      *
00012  *
      *
00013  *   This project is free software: you can redistribute it and/or modify it under the terms of the
      EUPL license as    *
00014  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
      *
00015  *
      *
00016  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
      or agreed to in *
00017  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
      without even the  *
00018  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
      check specific   *
00019  *   language governing permissions and limitations and more details.
      *
00020  *
      *
00021  *   You should use this project in compliance with the EUPL license. You should have received a copy
      of the license   *
00022  *   along with this project. If not, see the license at < https://eupl.eu/ >.
      *
00023
      *****************************************************************************************************************************/
00024
00025 //
      ================================================================================================================================
00026 #pragma once
00027 //
      ================================================================================================================================
00028
00029 //
      ================================================================================================================================
00030 #if ((defined __WIN32__) || (defined _WIN32)) && (!defined LIBZMQUTILS_STATIC)
00031 #ifdef LIBZMQUTILS_LIBRARY
00032 #define LIBZMQUTILS_EXPORT  __declspec(dllexport)
00033 #else
00034 #define LIBZMQUTILS_EXPORT  __declspec(dllimport)
00035 #endif
00036 #else
00037 /* Static libraries or non-Windows needs no special declaration. */
```

```
00038 # define LIBZMQUTILS_EXPORT
00039 #endif
00040 //
      ================================================================================================================
```

5.21 command_client.cpp

```
00001
00002 #include <zmq/zmq.hpp>
00003 #include <zmq/zmq_addon.hpp>
00004
00005 #include <iostream>
00006
00007 #include <iostream>
00008 #include <string>
00009 #include <cstring>
00010 #include <cstdlib>
00011 #include <algorithm>
00012
00013 #include "LibZMQUtils/CommandServerClient/command_client.h"
00014 #include "LibZMQUtils/utils.h"
00015
00016 // ZMQUTILS NAMESPACES
00017 //
      ================================================================================================================
00018 namespace zmqutils{
00019 //
      ================================================================================================================
00020
00021 const int CommandClientBase::kClientAliveTimeoutMsec = 5000;
00022 const int CommandClientBase::kClientSendAlivePeriodMsec = 3000;
00023
00024
00025 CommandClientBase::CommandClientBase(const std::string &server_endpoint) :
00026     server_endpoint_(server_endpoint),
00027     context_(nullptr),
00028     socket_(nullptr),
00029     auto_alive_working_(false)
00030 {
00031
00032 }
00033
00034 CommandClientBase::~CommandClientBase()
00035 {
00036     if (this->auto_alive_working_)
00037         this->stopAutoAlive();
00038     this->stopClient();
00039 }
00040
00041 bool CommandClientBase::startClient(const std::string& interface_name)
00042 {
00043     // Auxiliar variables.
00044     std::string ip, name, pid;
00045
00046     // Get the client ip.
00047     std::vector<utils::NetworkAdapterInfo> interfcs = utils::getHostIPsWithInterfaces();
00048     auto it = std::find_if(interfcs.begin(), interfcs.end(), [&interface_name](const
      utils::NetworkAdapterInfo& info)
00049                            {return info.name == interface_name;});
00050     if (it == interfcs.end())
00051         return false;
00052     ip = it->ip;
00053
00054     // Get the host name.
00055     name = utils::getHostname();
00056
00057     // Get the current pid.
00058     pid = std::to_string(utils::getCurrentPID());
00059
00060     // Store the info.
00061     this->client_info_ = common::HostClient(ip, name, pid);
00062
00063     std::cout<<client_info_.id<<std::endl;
00064
00065     // If server is already started, do nothing
00066     if (this->socket_)
00067         return false;
00068
00069     // Create the ZMQ context.
00070     if (!this->context_)
00071         this->context_ = new zmq::context_t(1);
00072
00073     try
00074     {
00075         this->socket_ = new zmq::socket_t(*this->context_, zmq::socket_type::req);
```

```
00076         this->socket_->connect(this->server_endpoint_);
00077         // Set timeout so socket will not wait for answer more than client alive timeout.
00078         this->socket_->set(zmq::sockopt::rcvtimeo, CommandClientBase::kClientAliveTimeoutMsec);
00079         this->socket_->set(zmq::sockopt::linger, 0);
00080     }
00081     catch (const zmq::error_t &error)
00082     {
00083         delete this->socket_;
00084         this->socket_ = nullptr;
00085 
00086         std::cerr << "Error at socket creation: " << error.num();
00087         // TODO: handle error
00088         return false;
00089     }
00090 
00091     // All ok.
00092     return true;
00093 }
00094 
00095 void CommandClientBase::stopClient()
00096 {
00097     // If server is already stopped, do nothing.
00098     if (!this->socket_)
00099         return;
00100 
00101 
00102     // Destroy the  socket.
00103     delete this->socket_;
00104     this->socket_ = nullptr;
00105 
00106     std::this_thread::sleep_for(std::chrono::milliseconds(1050));
00107 
00108     // Delete context
00109 
00110     if (this->context_)
00111     {
00112         delete this->context_;
00113         this->context_ = nullptr;
00114     }
00115 }
00116 
00117 void CommandClientBase::resetClient()
00118 {
00119     if (this->socket_)
00120     {
00121         // Destroy the socket and create again to flush.
00122         delete this->socket_;
00123 
00124         std::this_thread::sleep_for(std::chrono::milliseconds(1050));
00125 
00126         try
00127         {
00128             this->socket_ = new zmq::socket_t(*this->context_, zmq::socket_type::req);
00129             this->socket_->connect(this->server_endpoint_);
00130             // Set timeout so socket will not wait for answer more than client alive timeout.
00131             this->socket_->set(zmq::sockopt::rcvtimeo, CommandClientBase::kClientAliveTimeoutMsec);
00132             this->socket_->set(zmq::sockopt::linger, 0);
00133         }
00134         catch (const zmq::error_t &error)
00135         {
00136             delete this->socket_;
00137             this->socket_ = nullptr;
00138 
00139             std::cerr << "Error at socket creation: " << error.num();
00140             // TODO: handle error
00141         }
00142     }
00143 }
00144 
00145 void CommandClientBase::startAutoAlive()
00146 {
00147     this->auto_alive_working_ = true;
00148     this->auto_alive_future_ = std::async(std::launch::async, [this]{this->sendAliveCallback();});
00149 }
00150 
00151 void CommandClientBase::stopAutoAlive()
00152 {
00153     if (this->auto_alive_working_)
00154     {
00155         this->auto_alive_working_ = false;
00156         this->auto_alive_cv_.notify_all();
00157         this->auto_alive_future_.wait();
00158     }
00159 }
00160 
00161 void CommandClientBase::setClientHostIP(const std::string&){}
00162 
```

```
00163 void CommandClientBase::setClientId(const std::string &){}
00164
00165 int CommandClientBase::sendCommand(const CommandData& msg, void* &data_out, size_t &out_bytes)
00166 {
00167     if (!this->socket_)
00168         return -1;
00169     try
00170     {
00171
00172         zmq::multipart_t multipart_msg(this->prepareMessage(msg));
00173
00174         // Send the multiple messages.
00175         multipart_msg.send(*this->socket_);
00176
00177         std::cout<<"Data sent"<<std::endl;
00178
00179     }  catch (const zmq::error_t &error)
00180     {
00181         // TODO: handle error
00182         return error.num();
00183     }
00184
00185     std::cout<<"Waiting response"<<std::endl;
00186
00187     int res = this->recvFromSocket(this->socket_, data_out, out_bytes);
00188
00189     if (this->auto_alive_working_)
00190         this->auto_alive_cv_.notify_one();
00191
00192     return res;
00193
00194 }
00195
00196 int CommandClientBase::sendBadCommand1(void* &, size_t &)
00197 {
00198     return 0;
00199 }
00200
00201 int CommandClientBase::recvFromSocket(zmq::socket_t *socket, void *&data, size_t &data_size_bytes)
    const
00202 {
00203     // Reset output variables
00204     data = nullptr;
00205     data_size_bytes = 0;
00206
00207     // Try to receive data. If an execption is thrown, receiving fails and an error code is generated.
00208     int result = 0;
00209     zmq::recv_result_t recv_result;
00210     zmq::message_t message_recv;
00211
00212     try
00213     {
00214         recv_result = socket->recv(message_recv);
00215     }
00216     catch(zmq::error_t& error)
00217     {
00218         result = error.num();
00219     }
00220
00221     // Return error code if receiving fails or if no data was received
00222     if (!recv_result.has_value())
00223     {
00224         result = EAGAIN;
00225     }
00226     else if (0 == result)
00227     {
00228         // If data was received, copy received data to out parameters and return not error code
00229         data_size_bytes = message_recv.size();
00230         if (data_size_bytes > 0)
00231         {
00232             uint8_t *msg_data = new uint8_t[data_size_bytes];
00233             uint8_t *message_recv_pointer = static_cast<uint8_t*>(message_recv.data());
00234             std::copy(message_recv_pointer, message_recv_pointer + data_size_bytes, msg_data);
00235             data = msg_data;
00236         }
00237     }
00238
00239     return result;
00240 }
00241
00242 void CommandClientBase::sendAliveCallback()
00243 {
00244     std::mutex m;
00245     std::unique_lock<std::mutex> lk(m);
00246     bool send_success = true;
00247     bool recv_success = true;
00248     void *data_out;
```

```
00249     size_t out_size;
00250     zmq::multipart_t msg;
00251     zmq::socket_t *alive_socket = new zmq::socket_t(*this->context_, zmq::socket_type::req);
00252     alive_socket->connect(this->server_endpoint_);
00253     // Set timeout so socket will not wait for answer more than client alive timeout.
00254     alive_socket->set(zmq::sockopt::rcvtimeo, CommandClientBase::kClientAliveTimeoutMsec);
00255     alive_socket->set(zmq::sockopt::linger, 0);
00256 
00257     while(this->auto_alive_working_)
00258     {
00259         auto res =
00260             this->auto_alive_cv_.wait_for(lk,
    std::chrono::milliseconds(CommandClientBase::kClientSendAlivePeriodMsec));
00261 
00262         if (std::cv_status::timeout == res)
00263         {
00264             msg = this->prepareMessage(
00265 
    CommandData(static_cast<common::CommandType>(BaseServerCommand::REQ_ALIVE)));
00266             try
00267             {
00268                 msg.send(*alive_socket);
00269             }  catch (const zmq::error_t &error)
00270             {
00271                 // TODO: handle error
00272                 std::cerr « "Failed to automatically send alive command with error: " « error.num() «
    std::endl;
00273                 send_success = false;
00274             }
00275 
00276             if (send_success)
00277             {
00278                 auto recv_result = this->recvFromSocket(alive_socket, data_out, out_size);
00279                 auto *data_bytes = static_cast<std::uint8_t*>(data_out);
00280 
00281                 if (0 == recv_result && out_size == sizeof(CommandClientBase::CommandError))
00282                 {
00283                     CommandClientBase::CommandError error;
00284 
00285 
00286                     zmqutils::utils::binarySerializeDeserialize(
00287                                 data_bytes, sizeof(CommandClientBase::CommandError), &error);
00288 
00289                     recv_success = error == CommandClientBase::CommandError::NOT_ERROR;
00290 
00291                 }
00292                 else
00293                 {
00294                     std::cerr « "Auto alive message answer receive failed" « std::endl;
00295                     recv_success = false;
00296                 }
00297 
00298                 delete[] data_bytes;
00299             }
00300 
00301             if (!send_success || !recv_success)
00302             {
00303                 std::cerr « "Failed auto sending alive message. Process will be stopped." « std::endl;
00304                 this->auto_alive_working_ = false;
00305             }
00306 
00307         }
00308 
00309     }
00310 
00311     delete alive_socket;
00312 }
00313 
00314 zmq::multipart_t CommandClientBase::prepareMessage(const CommandData &msg)
00315 {
00316     // Prepare the ip data.
00317     zmq::message_t message_ip(this->client_info_.ip.begin(), this->client_info_.ip.end());
00318     // Prepare the hostname data.
00319     zmq::message_t message_host(this->client_info_.hostname.begin(),
    this->client_info_.hostname.end());
00320     // Prepare the pid data.
00321     zmq::message_t message_pid(this->client_info_.pid.begin(), this->client_info_.pid.end());
00322     // Prepare the command data.
00323     std::uint8_t command_buffer[sizeof(common::CmdRequestId)];
00324     zmqutils::utils::binarySerializeDeserialize(&msg.command_id, sizeof(common::CmdRequestId),
    command_buffer);
00325     zmq::message_t message_command(&command_buffer, sizeof(common::CmdRequestId));
00326 
00327 
00328     // Prepare the multipart msg.
00329     zmq::multipart_t multipart_msg;
00330     multipart_msg.add(std::move(message_ip));
```

```
00331     multipart_msg.add(std::move(message_host));
00332     multipart_msg.add(std::move(message_pid));
00333     multipart_msg.add(std::move(message_command));
00334 
00335     // Add command parameters if they exist
00336     if (msg.params_size > 0)
00337     {
00338         // Prepare the command parameters
00339         zmq::message_t message_params(msg.params.get(), msg.params_size);
00340         multipart_msg.add(std::move(message_params));
00341     }
00342 
00343     return multipart_msg;
00344 }
00345 
00346 } // END NAMESPACES.
00347 //
      ==========================================================================================================================
```

5.22 sources/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¨
```

5.22.1 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.

5.23 command_server.cpp

Go to the documentation of this file.

```
00001 
      /*************************************************************************************************************
00002  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
      *
00003  *
      *
00004  *   Copyright (C) 2023 Degoras Project Team
      *
00005  *                      < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
      *
00006  *                      < Jesús Relinque Madroñal >
      *
00007  *
      *
00008  *   This file is part of LibZMQUtils.
      *
00009  *
      *
00010  *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
      the EUPL license   *
```

```
00011  *   as soon they will be approved by the European Commission (IDABC).
       *
00012  *
       *
00013  *   This project is free software: you can redistribute it and/or modify it under the terms of the
       EUPL license as    *
00014  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
       *
00015  *
       *
00016  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
       or agreed to in *
00017  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
       without even the  *
00018  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
       check specific   *
00019  *   language governing permissions and limitations and more details.
       *
00020  *
       *
00021  *   You should use this project in compliance with the EUPL license. You should have received a copy
       of the license   *
00022  *   along with this project. If not, see the license at < https://eupl.eu/ >.
       *
00023
       ********************************************************************************************************************/
00024
00033 // C++ INCLUDES
00034 //
       ============================================================================================================================
00035 #include <iostream>
00036 #include <stdio.h>
00037 #include <zmq/zmq_addon.hpp>
00038 #include <zmq/zmq.h>
00039 //
       ============================================================================================================================
00040
00041 // ZMQUTILS INCLUDES
00042 //
       ============================================================================================================================
00043 #include "LibZMQUtils/CommandServerClient/command_server.h"
00044 #include "LibZMQUtils/utils.h"
00045 //
       ============================================================================================================================
00046
00047 // ZMQUTILS NAMESPACES
00048 //
       ============================================================================================================================
00049 namespace zmqutils{
00050 //
       ============================================================================================================================
00051
00052 CommandServerBase::CommandServerBase(unsigned int port, const std::string& local_addr) :
00053     context_(nullptr),
00054     main_socket_(nullptr),
00055     server_endpoint_("tcp://" + local_addr + ":" + std::to_string(port)),
00056     server_port_(port),
00057     server_working_(false),
00058     check_clients_alive_(true)
00059 {
00060     // Get the adapters.
00061     std::vector<utils::NetworkAdapterInfo> interfcs = utils::getHostIPsWithInterfaces();
00062     // Store the adapters.
00063     if(local_addr == "*")
00064         this->server_listen_adapters_ = interfcs;
00065     else
00066     {
00067         for(const auto& intrfc : interfcs)
00068         {
00069             if(intrfc.ip == local_addr)
00070                 this->server_listen_adapters_.push_back(intrfc);
00071         }
00072     }
00073 }
00074
00075 const std::future<void> &CommandServerBase::getServerWorkerFuture() const {return
       this->server_worker_future_;}
00076
00077 const std::map<std::string, HostClient> &CommandServerBase::getConnectedClients() const
00078 {return this->connected_clients_;}
00079
00080 void CommandServerBase::setClientStatusCheck(bool)
00081 {
00082     // Safe mutex lock
00083     std::unique_lock<std::mutex> lock(this->mtx_);
00084     // Disable the client alive checking.
00085     this->check_clients_alive_ = false;
```

```
00086     if(this->main_socket_)
00087         this->main_socket_->set(zmq::sockopt::rcvtimeo, -1);
00088 }
00089
00090 const unsigned& CommandServerBase::getServerPort() const {return this->server_port_;}
00091
00092 const std::vector<utils::NetworkAdapterInfo>& CommandServerBase::getServerAddresses() const
00093 {return this->server_listen_adapters_;}
00094
00095 const std::string& CommandServerBase::getServerEndpoint() const {return this->server_endpoint_;}
00096
00097 void CommandServerBase::startServer()
00098 {
00099     // Safe mutex lock
00100     std::unique_lock<std::mutex> lock(this->mtx_);
00101
00102     // If server is already started, do nothing
00103     if (this->server_working_)
00104         return;
00105
00106     // Create the ZMQ context.
00107     if (!this->context_)
00108         this->context_ = new zmq::context_t(1);
00109
00110     // Launch server worker in other thread.
00111     this->server_worker_future_ = std::async(std::launch::async, &CommandServerBase::serverWorker,
      this);
00112 }
00113
00114 void CommandServerBase::stopServer()
00115 {
00116     // Safe mutex lock
00117     std::unique_lock<std::mutex> lock(this->mtx_);
00118
00119     // If server is already stopped, do nothing.
00120     if (!this->server_working_)
00121         return;
00122
00123     // Set the shared working flag to false (is atomic).
00124     this->server_working_ = false;
00125
00126     // Delete the context.
00127     if (this->context_)
00128     {
00129         delete this->context_;
00130         context_ = nullptr;
00131     }
00132
00133     // Clean the clients.
00134     this->connected_clients_.clear();
00135 }
00136
00137 CommandServerBase::~CommandServerBase()
00138 {
00139     // Stop the server (this function also deletes the pointers).
00140     this->stopServer();
00141 }
00142
00143 BaseServerResult CommandServerBase::execReqConnect(const CommandRequest& cmd_req)
00144 {
00145     // Safe mutex lock.
00146     std::unique_lock<std::mutex> lock(this->mtx_);
00147
00148     // Check if the client is already connected.
00149     auto it = this->connected_clients_.find(cmd_req.client.id);
00150     if(it != this->connected_clients_.end())
00151         return BaseServerResult::ALREADY_CONNECTED;
00152
00153     // Add the new client.
00154     this->connected_clients_[cmd_req.client.id] = cmd_req.client;
00155
00156     // Update the timeout of the main socket.
00157     if(this->check_clients_alive_)
00158         this->updateServerTimeout();
00159
00160     // Call to the internal callback.
00161     this->onConnected(cmd_req.client);
00162
00163     // All ok.
00164     return BaseServerResult::COMMAND_OK;
00165 }
00166
00167 BaseServerResult CommandServerBase::execReqDisconnect(const CommandRequest& cmd_req)
00168 {
00169     // Safe mutex lock.
00170     std::unique_lock<std::mutex> lock(this->mtx_);
00171
```

```
00172     // Get the client.
00173     auto it = this->connected_clients_.find(cmd_req.client.id);
00174 
00175     // Remove the client from the map of connected clients.
00176     this->connected_clients_.erase(it);
00177 
00178     // Call to the internal callback.
00179     this->onDisconnected(cmd_req.client);
00180 
00181     // Update the timeout of the main socket.
00182     if(this->check_clients_alive_)
00183         this->updateServerTimeout();
00184 
00185     // All ok.
00186     return BaseServerResult::COMMAND_OK;
00187 }
00188 
00189 void CommandServerBase::serverWorker()
00190 {
00191     // Auxiliar variables.
00192     BaseServerResult result;
00193 
00194     // Set the working flag to true.
00195     this->server_working_ = true;
00196 
00197     // Start server socket
00198     this->resetSocket();
00199 
00200     // Server worker loop.
00201     // If there is no client connected wait for a client to connect or for an exit message. If there
00202     // is a client connected set timeout, so if no command comes in time, check the last time
    connection
00203     // for each client. The loop can be stopped (in a safe way) if using the stopServer() function.
00204     while(this->main_socket_ && this->server_working_)
00205     {
00206         // Message container.
00207         CommandRequest cmd_request;
00208 
00209         // Result container.
00210         CommandReply cmd_reply;
00211 
00212         // Receive the data.
00213         result = this->recvFromSocket(cmd_request);
00214 
00215         // Check all the clients status.
00216         if(this->check_clients_alive_)
00217             this->checkClientsAliveStatus();
00218 
00219         // Process the data.
00220         if(result == BaseServerResult::COMMAND_OK && !this->server_working_)
00221         {
00222             // In this case, we will close the server. Call to the internal callback.
00223             this->onServerStop();
00224         }
00225         else if(result == BaseServerResult::TIMEOUT_REACHED)
00226         {
00227             // DO NOTHING.
00228         }
00229         else if (result != BaseServerResult::COMMAND_OK)
00230         {
00231             // Internal callback.
00232             this->onInvalidMsgReceived(cmd_request);
00233 
00234             // Prepare the message.
00235             std::uint8_t res_buff[sizeof(BaseServerResult)];
00236             utils::binarySerializeDeserialize(&result, sizeof(BaseServerResult), res_buff);
00237             zmq::message_t message_res(res_buff, sizeof(BaseServerResult));
00238 
00239             // Send response callback.
00240             cmd_reply.result = result;
00241             this->onSendingResponse(cmd_reply);
00242 
00243             // Send the response.
00244             try
00245             {
00246                 this->main_socket_->send(message_res, zmq::send_flags::none);
00247             }
00248             catch (const zmq::error_t &error)
00249             {
00250                 // Check if we want to close the server.
00251                 // The error code is for ZMQ EFSM error.
00252                 if(!(error.num() == common::kZmqEFSMError && !this->server_working_))
00253                     this->onServerError(error, "Error while sending a response.");
00254             }
00255         }
00256         else if (result == BaseServerResult::COMMAND_OK)
00257         {
```

```
00258             // Reply id buffer.
00259             std::unique_ptr<std::uint8_t> rep_id_buff;
00260 
00261             // Execute the command.
00262             this->processCommand(cmd_request, cmd_reply);
00263 
00264             // Prepare the command result.
00265             CommandServerBase::prepareCommandResult(cmd_reply.result, rep_id_buff);
00266             zmq::message_t message_rep_id(rep_id_buff.get(), sizeof(common::CmdReplyRes));
00267 
00268             // Prepare the multipart msg.
00269             zmq::multipart_t multipart_msg;
00270             multipart_msg.add(std::move(message_rep_id));
00271 
00272             // Specific data.
00273             if(cmd_reply.result == BaseServerResult::COMMAND_OK && cmd_reply.params_size != 0)
00274             {
00275                 // Prepare the custom response.
00276                 zmq::message_t message_rep_custom(cmd_reply.params.get(), cmd_reply.params_size);
00277                 multipart_msg.add(std::move(message_rep_custom));
00278             }
00279 
00280             // Sending callback.
00281             this->onSendingResponse(cmd_reply);
00282 
00283             // Send the message.
00284             try
00285             {
00286                 multipart_msg.send(*this->main_socket_);
00287             }
00288             catch (const zmq::error_t &error)
00289             {
00290                 // Check if we want to close the server.
00291                 // The error code is for ZMQ EFSM error.
00292                 if(!(error.num() == common::kZmqEFSMError && !this->server_working_))
00293                     this->onServerError(error, "Error while sending a response.");
00294             }
00295         }
00296     }
00297 
00298     // Delete pointers for clean finish the worker.
00299     if (this->main_socket_)
00300     {
00301         delete this->main_socket_;
00302         this->main_socket_ = nullptr;
00303     }
00304 }
00305 
00306 BaseServerResult CommandServerBase::recvFromSocket(CommandRequest& request)
00307 {
00308     // Result variable.
00309     BaseServerResult result = BaseServerResult::COMMAND_OK;
00310 
00311     // Containers.
00312     bool recv_result;
00313     zmq::multipart_t multipart_msg;
00314 
00315     // Try to receive data. If an execption is thrown, receiving fails and an error code is generated.
00316     try
00317     {
00318         // Call to the internal waiting command callback.
00319         this->onWaitingCommand();
00320 
00321         // Wait the command.
00322         recv_result = multipart_msg.recv(*(this->main_socket_));
00323 
00324         // Store the raw data.
00325         request.raw_msg = multipart_msg.clone();
00326     }
00327     catch(zmq::error_t& error)
00328     {
00329         // Check if we want to close the server.
00330         // The error code is for ZMQ EFSM error.
00331         if(error.num() == common::kZmqEFSMError && !this->server_working_)
00332             return BaseServerResult::COMMAND_OK;
00333 
00334         // Else, call to error callback.
00335         this->onServerError(error, "Error while receiving a request.");
00336         return BaseServerResult::INTERNAL_ZMQ_ERROR;
00337     }
00338 
00339     // Check for empty msg or timeout reached.
00340     if (multipart_msg.empty() && !recv_result)
00341         return BaseServerResult::TIMEOUT_REACHED;
00342     else if (multipart_msg.empty())
00343         return BaseServerResult::EMPTY_MSG;
00344 
```

```
00345     // Check the multipart msg size.
00346     if (multipart_msg.size() == 4 || multipart_msg.size() == 5)
00347     {
00348         // Auxiliar containers.
00349         std::string ip;
00350         std::string hostname;
00351         std::string pid;
00352 
00353         // Get the multipart data.
00354         zmq::message_t message_ip = multipart_msg.pop();
00355         zmq::message_t message_hostname = multipart_msg.pop();
00356         zmq::message_t message_pid = multipart_msg.pop();
00357         zmq::message_t message_command = multipart_msg.pop();
00358 
00359         // Get the sizes.
00360         size_t ip_size_bytes = message_ip.size();
00361         size_t host_size_bytes = message_hostname.size();
00362         size_t pid_size_bytes = message_pid.size();
00363         size_t command_size_bytes = message_command.size();
00364 
00365         // First get the ip data.
00366         if (ip_size_bytes > 0)
00367             ip = std::string(static_cast<char*>(message_ip.data()), ip_size_bytes);
00368         else
00369             return BaseServerResult::EMPTY_CLIENT_IP;
00370 
00371         // Get the hostname data.
00372         if (host_size_bytes > 0)
00373             hostname = std::string(static_cast<char*>(message_hostname.data()), host_size_bytes);
00374         else
00375             return BaseServerResult::EMPTY_CLIENT_NAME;
00376 
00377         // Get the pid data.
00378         if (host_size_bytes > 0)
00379             pid = std::string(static_cast<char*>(message_pid.data()), pid_size_bytes);
00380         else
00381             return BaseServerResult::EMPTY_CLIENT_PID;
00382 
00383         // Update the client info.
00384         request.client = HostClient(ip, hostname, pid);
00385         request.client.last_connection = std::chrono::steady_clock::now();
00386 
00387         // Update the last connection if the client is connected.
00388         this->updateClientLastConnection(request.client.id);
00389 
00390         // Get the command.
00391         if (command_size_bytes == sizeof(CmdRequestId))
00392         {
00393             int raw_command;
00394             utils::binarySerializeDeserialize(message_command.data(), sizeof(BaseServerCommand),
    &raw_command);
00395             // Validate the command.
00396             if(CommandServerBase::validateCommand(raw_command))
00397                 request.command = static_cast<BaseServerCommand>(raw_command);
00398             else
00399             {
00400                 request.command = BaseServerCommand::INVALID_COMMAND;
00401                 return BaseServerResult::INVALID_MSG;
00402             }
00403         }
00404         else
00405         {
00406             request.command = BaseServerCommand::INVALID_COMMAND;
00407             return BaseServerResult::INVALID_MSG;
00408         }
00409 
00410         // If there is still one more part, they are the parameters.
00411         if (multipart_msg.size() == 1)
00412         {
00413             // Get the message and the size.
00414             zmq::message_t message_params = multipart_msg.pop();
00415             size_t params_size_bytes = message_params.size();
00416 
00417             std::cout<<multipart_msg.str()<<std::endl;
00418             std::cout<<params_size_bytes<<std::endl;
00419 
00420             // Check the parameters.
00421             if(params_size_bytes > 0)
00422             {
00423                 // Get and store the parameters data.
00424                 std::unique_ptr<std::uint8_t> params =
00425                     std::unique_ptr<std::uint8_t>(new std::uint8_t[params_size_bytes]);
00426                 auto *params_pointer = static_cast<std::uint8_t*>(message_params.data());
00427                 std::copy(params_pointer, params_pointer + params_size_bytes, params.get());
00428                 request.params = std::move(params);
00429                 request.params_size = params_size_bytes;
00430             }
```

```
00431             else
00432                 return BaseServerResult::EMPTY_PARAMS;
00433         }
00434     }
00435     else
00436         return BaseServerResult::INVALID_PARTS;
00437 
00438     // Return the result.
00439     return result;
00440 }
00441 
00442 void CommandServerBase::prepareCommandResult(BaseServerResult result, std::unique_ptr<std::uint8_t>&
    data_out)
00443 {
00444     data_out = std::unique_ptr<std::uint8_t>(new std::uint8_t[sizeof(BaseServerResult)]);
00445     utils::binarySerializeDeserialize(&result, sizeof(CmdReplyRes), data_out.get());
00446 }
00447 
00448 bool CommandServerBase::validateCommand(int raw_command)
00449 {
00450     // Auxiliar variables.
00451     bool result = false;
00452     int reserved_cmd = static_cast<int>(common::BaseServerCommand::RESERVED_COMMANDS);
00453     int end_base_cmd = static_cast<int>(common::BaseServerCommand::END_BASE_COMMANDS);
00454     // Check if the command is valid.
00455     if (raw_command >= common::kMinBaseCmdId && raw_command < reserved_cmd)
00456         result = true;
00457     else if(raw_command > end_base_cmd)
00458         result = true;
00459     return result;
00460 }
00461 
00462 void CommandServerBase::processCommand(const CommandRequest& request, CommandReply& reply)
00463 {
00464     // First of all, call to the internal callback.
00465     this->onCommandReceived(request);
00466 
00467     // Store the command in the reply.
00468     reply.request_cmd = request.command;
00469 
00470     // Process the different commands.
00471     // 1 - Process is the connect request.
00472     // 2 - If the command is other, check if the client is connected to the server.
00473     // 3 - If it is, check if the command is valid.
00474     // 4 - If valid, process the rest of the base commands or the custom command.
00475     if (BaseServerCommand::REQ_CONNECT == request.command)
00476     {
00477         reply.result = this->execReqConnect(request);
00478     }
00479     else if(this->connected_clients_.find(request.client.id) == this->connected_clients_.end())
00480     {
00481         reply.result = BaseServerResult::CLIENT_NOT_CONNECTED;
00482     }
00483     else if (BaseServerCommand::REQ_DISCONNECT == request.command)
00484     {
00485         reply.result = this->execReqDisconnect(request);
00486     }
00487     else if (BaseServerCommand::REQ_ALIVE == request.command)
00488     {
00489         reply.result = BaseServerResult::COMMAND_OK;
00490     }
00491     else
00492     {
00493         // Custom command, so call the internal callback.
00494         this->onCustomCommandReceived(request, reply);
00495 
00496         // Chek for an invalid msg.
00497         if(reply.result == BaseServerResult::INVALID_MSG)
00498             this->onInvalidMsgReceived(request);
00499     }
00500 }
00501 
00502 void CommandServerBase::checkClientsAliveStatus()
00503 {
00504     // Safe mutex lock
00505     std::unique_lock<std::mutex> lock(this->mtx_);
00506 
00507     // Auxiliar containers.
00508     std::vector<std::string> dead_clients;
00509     std::chrono::milliseconds timeout(common::kDefaultClientAliveTimeoutMsec);
00510     std::chrono::milliseconds min_remaining_time = timeout;
00511 
00512     // Get the current time.
00513     utils::SCTimePointStd now = std::chrono::steady_clock::now();
00514 
00515     // Check each connection.
00516     for(auto& client : this->connected_clients_)
```

```
00517     {
00518         // Get the last connection time.
00519         const auto& last_conn = client.second.last_connection;
00520         // Check if the client reaches the timeout checking the last connection time.
00521         auto since_last_conn = std::chrono::duration_cast<std::chrono::milliseconds>(now - last_conn);
00522         if(since_last_conn >= timeout)
00523         {
00524             // If dead, call the onDead callback and quit the client from the map.
00525             this->onDeadClient(client.second);
00526             dead_clients.push_back(client.first);
00527         }
00528         else
00529         {
00530             // If the client is not dead, check the minor timeout of the client to set
00531             // with the remain time to reach the timeout.
00532             min_remaining_time = std::min(min_remaining_time, timeout - since_last_conn);
00533         }
00534     }
00535
00536     // Remove dead clients from the map.
00537     for(auto& client : dead_clients)
00538     {
00539         this->connected_clients_.erase(client);
00540     }
00541
00542     // Disable the timeout if no clients remains or set the socket timeout to the
00543     // minimum remaining time to the timeout among all clients.
00544     if(this->connected_clients_.empty())
00545     {
00546         this->main_socket_->set(zmq::sockopt::rcvtimeo, -1);
00547     }
00548     else
00549     {
00550         this->main_socket_->set(zmq::sockopt::rcvtimeo, static_cast<int>(min_remaining_time.count()));
00551     }
00552 }
00553
00554 void CommandServerBase::updateClientLastConnection(const std::string &id)
00555 {
00556     // Safe mutex lock.
00557     std::unique_lock<std::mutex> lock(this->mtx_);
00558     // Update the client last connection.
00559     auto client_itr = this->connected_clients_.find(id);
00560     if(client_itr != this->connected_clients_.end())
00561         client_itr->second.last_connection = std::chrono::steady_clock::now();
00562 }
00563
00564 void CommandServerBase::updateServerTimeout()
00565 {
00566     // Calculate the minor timeout to set it into the socket.
00567     auto min_timeout = std::min_element(this->connected_clients_.begin(),
    this->connected_clients_.end(),
00568         [](const auto& a, const auto& b)
00569         {
00570         auto diff_a = std::chrono::duration_cast<std::chrono::milliseconds>(
00571             std::chrono::steady_clock::now() - a.second.last_connection);
00572         auto diff_b = std::chrono::duration_cast<std::chrono::milliseconds>(
00573             std::chrono::steady_clock::now() - b.second.last_connection);
00574         return diff_a.count() < diff_b.count();
00575         });
00576
00577     if (min_timeout != this->connected_clients_.end())
00578     {
00579         auto remain_time = common::kDefaultClientAliveTimeoutMsec -
    std::chrono::duration_cast<std::chrono::milliseconds>(
00580                                     std::chrono::steady_clock::now() -
    min_timeout->second.last_connection).count();
00581         this->main_socket_->set(zmq::sockopt::rcvtimeo, std::max(0, static_cast<int>(remain_time)));
00582     }
00583     else
00584     {
00585         this->main_socket_->set(zmq::sockopt::rcvtimeo, -1);
00586     }
00587 }
00588
00589 void CommandServerBase::resetSocket()
00590 {
00591     // Auxiliar variables.
00592     int res = 0;
00593     const zmq::error_t* last_error;
00594     unsigned reconnect_count = common::kServerReconnTimes;
00595
00596     // Delete the previous socket.
00597     if (this->main_socket_)
00598     {
00599         delete this->main_socket_;
00600         this->main_socket_ = nullptr;
```

```
00601     }
00602     // Try creating a new socket.
00603     do
00604     {
00605         try
00606         {
00607             // Create the ZMQ rep socket.
00608             std::this_thread::sleep_for(std::chrono::microseconds(500));
00609             this->main_socket_ = new zmq::socket_t(*this->context_, zmq::socket_type::rep);
00610             this->main_socket_->bind(this->server_endpoint_);
00611             this->main_socket_->set(zmq::sockopt::linger, 0);
00612         }
00613         catch (const zmq::error_t& error)
00614         {
00615             // Delete the socket and store the last error.
00616             delete this->main_socket_;
00617             this->main_socket_ = nullptr;
00618             last_error = &error;
00619         }
00620         reconnect_count--;
00621     } while (res == EADDRINUSE && reconnect_count > 0);
00622
00623     if (!this->main_socket_ )
00624     {
00625         // Update the working flag and calls to the callback.
00626         this->server_working_ = false;
00627         this->onServerError(*last_error, "Error during socket creation.");
00628     }
00629     else
00630     {
00631         // Call to the internal callback.
00632         this->onServerStart();
00633     }
00634 }
00635
00636 void CommandServerBase::onCustomCommandReceived(const CommandRequest&, CommandReply& rep)
00637 {
00638     rep.result = BaseServerResult::NOT_IMPLEMENTED;
00639 }
00640
00641 } // END NAMESPACES.
00642 //
    ========================================================================================================================
```

5.24 common.cpp

```
00001
    /*****************************************************************************************************************
00002  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
    *
00003  *
    *
00004  *   Copyright (C) 2023 Degoras Project Team
    *
00005  *                      < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
    *
00006  *                      < Jesús Relinque Madroñal >
    *
00007  *
    *
00008  *   This file is part of LibZMQUtils.
    *
00009  *
    *
00010  *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
    the EUPL license   *
00011  *   as soon they will be approved by the European Commission (IDABC).
    *
00012  *
    *
00013  *   This project is free software: you can redistribute it and/or modify it under the terms of the
    EUPL license as    *
00014  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
    *
00015  *
    *
00016  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
    or agreed to in *
00017  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
    without even the  *
00018  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
    check specific   *
00019  *   language governing permissions and limitations and more details.
    *
```

```
00020  *
       *
00021  *   You should use this project in compliance with the EUPL license. You should have received a copy
       of the license   *
00022  *   along with this project. If not, see the license at < https://eupl.eu/ >.
       *
00023
       ************************************************************************************************************************/
00024
00025 #include "LibZMQUtils/CommandServerClient/common.h"
00026
00027
00028 zmqutils::common::HostClient::HostClient(const std::string &ip, const std::string &name,
00029                                          const std::string &pid, const std::string &info) :
00030     ip(ip),
00031     hostname(name),
00032     pid(pid),
00033     info(info)
00034 {
00035     // Create the host client internal identification.
00036     this->id = ip + "//" + name + "//" + pid;
00037 }
```

5.25 utils.cpp

```
00001
       /************************************************************************************************************************
00002  *   LibZMQUtils (ZMQ Utilitites Library): A libre library with ZMQ related useful utilities.
       *
00003  *
       *
00004  *   Copyright (C) 2023 Degoras Project Team
       *
00005  *                      < Ángel Vera Herrera, avera@roa.es - angeldelaveracruz@gmail.com >
       *
00006  *                      < Jesús Relinque Madroñal >
       *
00007  *
       *
00008  *   This file is part of LibZMQUtils.
       *
00009  *
       *
00010  *   Licensed under the European Union Public License (EUPL), Version 1.2 or subsequent versions of
       the EUPL license   *
00011  *   as soon they will be approved by the European Commission (IDABC).
       *
00012  *
       *
00013  *   This project is free software: you can redistribute it and/or modify it under the terms of the
       EUPL license as    *
00014  *   published by the IDABC, either Version 1.2 or, at your option, any later version.
       *
00015  *
       *
00016  *   This project is distributed in the hope that it will be useful. Unless required by applicable law
       or agreed to in *
00017  *   writing, it is distributed on an "AS IS" basis, WITHOUT ANY WARRANTY OR CONDITIONS OF ANY KIND;
       without even the  *
00018  *   implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the EUPL license to
       check specific   *
00019  *   language governing permissions and limitations and more details.
       *
00020  *
       *
00021  *   You should use this project in compliance with the EUPL license. You should have received a copy
       of the license   *
00022  *   along with this project. If not, see the license at < https://eupl.eu/ >.
       *
00023
       ************************************************************************************************************************/
00024
00025
00026
00027 #include <iomanip>
00028 #ifdef _WIN32
00029 #include <winsock2.h>
00030 #include <ws2tcpip.h>
00031 #include <Windows.h>
00032 #else
00033 #include <sys/socket.h>
00034 #include <ifaddrs.h>
00035 #include <netinet/in.h>
00036 #include <arpa/inet.h>
00037 #include <unistd.h>
```

```
00038 #endif
00039
00040 #include <sstream>
00041 #include <iphlpapi.h>
00042
00043 #include "LibZMQUtils/utils.h"
00044
00045 #ifndef _WIN32_WINNT
00046 #define _WIN32_WINNT 0x0600
00047 #elif _WIN32_WINNT < 0x0600
00048 #undef _WIN32_WINNT
00049 #define _WIN32_WINNT 0x0600
00050 #endif
00051
00052 // ZMQUTILS NAMESPACES
00053 //
      ================================================================================================================
00054 namespace zmqutils{
00055 namespace utils{
00056 //
      ================================================================================================================
00057
00058 //
      ================================================================================================================
00059 using std::chrono::duration;
00060 using std::chrono::duration_cast;
00061 using std::chrono::high_resolution_clock;
00062 using std::chrono::time_point_cast;
00063 //
      ================================================================================================================
00064
00065 std::vector<NetworkAdapterInfo> getHostIPsWithInterfaces()
00066 {
00067     // Result container.
00068     std::vector<NetworkAdapterInfo> adapters;
00069
00070 #ifdef _WIN32
00071
00072     // Buffer size.
00073     ULONG  buff_size = 0;
00074
00075     if (GetAdaptersAddresses(AF_INET, GAA_FLAG_SKIP_ANYCAST | GAA_FLAG_SKIP_MULTICAST |
      GAA_FLAG_SKIP_DNS_SERVER,
00076                              nullptr, nullptr, &buff_size) != ERROR_BUFFER_OVERFLOW)
00077     {
00078         return adapters;
00079     }
00080
00081     std::vector<char> buffer(buff_size);
00082
00083     PIP_ADAPTER_ADDRESSES adapter_addrs = reinterpret_cast<PIP_ADAPTER_ADDRESSES>(&buffer[0]);
00084
00085     if (GetAdaptersAddresses(AF_INET, GAA_FLAG_SKIP_ANYCAST | GAA_FLAG_SKIP_MULTICAST |
      GAA_FLAG_SKIP_DNS_SERVER,
00086                              nullptr, adapter_addrs, &buff_size) != NO_ERROR)
00087     {
00088         return adapters;
00089     }
00090
00091     while (adapter_addrs != nullptr)
00092     {
00093         if (adapter_addrs->OperStatus == IfOperStatusUp)
00094         {
00095             PIP_ADAPTER_UNICAST_ADDRESS unicast_addrs = adapter_addrs->FirstUnicastAddress;
00096             while (unicast_addrs != nullptr)
00097             {
00098                 sockaddr_in* sockaddr =
      reinterpret_cast<sockaddr_in*>(unicast_addrs->Address.lpSockaddr);
00099                 char* ip = inet_ntoa(sockaddr->sin_addr);
00100                 char f_name_ch[260];
00101                 char desc_ch[260];
00102                 char df_char = ' ';
00103
00104                 WideCharToMultiByte(CP_ACP,0,adapter_addrs->FriendlyName,-1, f_name_ch,260, &df_char,
      NULL);
00105                 WideCharToMultiByte(CP_ACP,0,adapter_addrs->Description,-1, desc_ch,260, &df_char,
      NULL);
00106
00107                 NetworkAdapterInfo adaptr;
00108                 adaptr.id = std::string(adapter_addrs->AdapterName);
00109                 adaptr.name = std::string(f_name_ch);
00110                 adaptr.descr = std::string(desc_ch);
00111                 adaptr.ip = std::string(ip);
00112                 adapters.push_back(adaptr);
00113                 unicast_addrs = unicast_addrs->Next;
00114             }
00115         }
```

```
00116
00117         adapter_addrs = adapter_addrs->Next;
00118     }
00119 #else
00120   // TODO
00121 #endif
00122
00123     // Return the ip interface maps.
00124     return adapters;
00125 }
00126
00127 std::string getHostname()
00128 {
00129     std::string name;
00130
00131 #ifdef _WIN32
00132
00133     WSADATA wsaData;
00134
00135     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
00136         return "";
00137
00138     char buffer[256];
00139     if (gethostname(buffer, sizeof(buffer)) != 0)
00140         WSACleanup();
00141
00142     // Clear.
00143     WSACleanup();
00144
00145     // Store the data.
00146     name = std::string(buffer);
00147
00148 #else
00149     // TODO
00150 #endif
00151
00152     // Return the hostname.
00153     return name;
00154 }
00155
00156 void binarySerializeDeserialize(const void *data, size_t data_size_bytes, void *dest)
00157 {
00158     const std::uint8_t* data_byes = reinterpret_cast<const std::uint8_t *>(data);
00159     std::uint8_t* dest_byes = reinterpret_cast<std::uint8_t*>(dest);
00160     std::reverse_copy(data_byes, data_byes + data_size_bytes, dest_byes);
00161 }
00162
00163 std::string timePointToString(const HRTimePointStd &tp, const std::string &format, bool add_ms, bool
      add_ns, bool utc)
00164 {
00165     // Stream to hold the formatted string and the return container.
00166     std::ostringstream ss;
00167     // Convert the time point to a duration and get the different time fractions.
00168     HRTimePointStd::duration dur = tp.time_since_epoch();
00169     const time_t secs = duration_cast<std::chrono::seconds>(dur).count();
00170     const long long mill = duration_cast<std::chrono::milliseconds>(dur).count();
00171     const unsigned long long ns = duration_cast<std::chrono::nanoseconds>(dur).count();
00172     const unsigned long long s_ns = secs * 1e9;
00173     const unsigned long long t_ns = (ns - s_ns);
00174     // Format the duration.
00175     if (const std::tm *tm = (utc ? std::gmtime(&secs) : std::localtime(&secs)))
00176     {
00177         ss « std::put_time(tm, format.c_str());
00178         if(add_ms && !add_ns)
00179             ss « '.' « std::setw(3) « std::setfill('0') « (mill - secs * 1e3);
00180         else if(add_ns)
00181             ss « '.' « std::setw(9) « std::setfill('0') « t_ns;
00182     }
00183     else
00184     {
00185         // If error, return an empty string.
00186         return std::string();
00187     }
00188     // Return the container.
00189     return ss.str();
00190 }
00191
00192 std::string timePointToIso8601(const HRTimePointStd &tp, bool add_ms, bool add_ns)
00193 {
00194     // Return the ISO 8601 datetime.
00195     return timePointToString(tp, "%Y-%m-%dT%H:%M:%S", add_ms, add_ns) + 'Z';
00196 }
00197
00198 std::string currentISO8601Date(bool add_ms)
00199 {
00200     auto now = high_resolution_clock::now();
00201     return timePointToIso8601(now, add_ms);
```

```
00202 }
00203
00204 unsigned getCurrentPID()
00205 {
00206 #if defined(_WIN32)
00207     return GetCurrentProcessId();
00208 #elif defined(__unix__) || defined(__APPLE__) || defined(__linux__)
00209     return getpid();
00210 #else
00211 // Unsupported or unknown platform
00212     return 0;
00213 #endif
00214 }
00215
00216 }} // END NAMESPACES.
00217 //
    ================================================================================================================================
```

Index