**echo_server**

The goal:

Describe common scenario of how to implement asynchronous method (of active object) in the most effective way – with support of Asio custom memory allocation and with handler “parking” until its logical activity ends (this is the most hard part because it isn’t described in Asio examples).

Describe scenario of separation of active objects belonging to different layers/isolated-logics by different (isolated) thread pools (execution resources).

Describe scenario of communication between active objects which probably belong to different boost::asio::io_service instances.

Describe coordinated communication between active objects/other code – “no one will call your callback if you don’t ask him to do it, each call of callback must be preceded by asynchronous operation call”.

**async_basics**

The goal:

Describe scenario of how to separate interface (*.hpp) from implementation (*.cpp) (mostly for the reasons of compilation speedup) when using active object with asynchronous methods being template member functions.

**async_basics2**

The goal:

Describe scenario of how to separate interface from implementation with “classic” usage of virtual methods.

Describe active object without templates used in the public part of class.

**qt_echo_server**

The goal:

Describe common scenario of how to glue the shown above active-object-based design with traditional GUI (single-threaded with the only main loop). One of the most beautiful (I mean only GUI part) GUI libraries is used – Qt. It is described how to glue Qt signal/slot mechanics with functional-object-based callbacks used in separated (from GUI) server implementation and called from separated thread pool(s).

**nmea_client**

The goal:

Show an example of usage of active-object-based design at the “client side”. The shown example implements active cashing of read messages without explicit calls from the program’s main thread.