# IOCOM library use scenarios

191122, updated 22.11.2019/pekka

The IOCOM and underlying operating system abstraction layer, EOSAL are scaled to abstraction level needed by application. This effects greatly to burden of complexity put on the developer and on resources of underlaying devices. We have to consider chooses below, “more the merrier” doesn’t hold: Each yes will add complexity to the developer and require additional resources.

- Use single thread or multi-thread model?
- Do we need transport layer security and device identification?
- Do we want to handle signals, not just bytes of shared memory.
- Do we use signal names to map the data between devices instead of plain numeric memory addresses?
- Is dynamic configuration of IO device network needed?
- Do we want also the application to be portable between micro-controllers and development environments?

Idea here: The requirement for different components of IO network vary. The IOCOM library is linked to both ends of of the communication, and provides IO network level infrastructure for both. The library is scaled to suit it’s use and environment.

### Communication between two devices

This is the simple case, I want my ESP32 to exchange data with my PC, or my Arduino to speak to Raspberry. Werther the communication is Ethernet, WiFi, serial port or blue tooth doesn’t really change how to do it, but effects to device resource use.
We need minimal abstraction, single thread model and data treated as transferred bytes within memory block is enough. My personal choice for this would be to add signals with JSON configuration to the mix, but otherwise to keep it in minimum.
In industrial environment TLS and device identification might be required even for this purpose, which adds a lot to complexity.

### IO interface for a complex automation device

Often an automation device contains a controlling computer (controller) and bunch of IO cards which connect to the control computer. Here the controller knows the IO cards which may connect it. Using signal configuration JSONs allows us to generate C code for both IO card and controller to use same signal map to communicate, for example to allow controller to set a “led_builtin” output on iocard “gina”.

- Here I would use single thread model for the IO device and multi thread model for the controller, signal setup in JSON and statically compile the signal maps (generated from JSONs) in both cotroller and IO devices. TLS and device identification would also be necessary in industrial environment.
- Current test/example code for controller: tito
- Current test/example code for IO device: gina

### IoT and cloud

In cloud application we typically have one server maintaining IO networks for multiple users. All Pekka’s devices must belong to ‘pekkanet’ IO network and speak with each others. At same time the

same server takes care of Markku's devices, which belong to 'markkunet'. No data must flow between 'pekkanet' and 'markkunet' in normal operating mode.
I cloud server we need practically the whole IOCOM in maximum extent:

- We need to be dynamically create IO device networks and and IO devices within these networks, map signals by name, etc.
- We often need user interface for an IO network. Let's say 'pekkanet' is my home automation IO network, and I want to control and monitor it using either dedicated application or web browser.
- Security is paramount. Cloud server must identify itself and present certificate signed by entry which is recognized by IO devices (chain of trust). IO device must identify itself with user name and password and optionally client certificate. All communication must be TLS sockets.
- Current test/example code for cloud server under development: frank

**Mix and match**

The scenarios above are only the ones I was thinking. All functionality can be used at any level, for example we can create IO device which chains other IO devices, or we can use cloud level abstraction running at local network server.

**Note about pins library**

There is optional pins library which allows setting up IO device's pins as JSON. Ideas:

- Microcontroller, platform and development tool independent
- Generates C code from JSON to set/get pin states from IO device code and to map the pins to IOCOM signals.

If you question is pins library needed, you are probably better off without it. Reading and writing IO pins using platform tools is just fine. Application portability from hardware or platform to another, or PC simulation, are not in "to do" list of many projects. In this case using pins is just unnecessary complexity.