# EtherCat Server API Requirements

1. The server code scope will be limited to hiding the details of the Ethercat protocol. Client code will be responsible for the coordination and sequencing of devices: the server code will provide event streams and get/set capabilities for parameters.
2. The addition of a slave should not alter the ‘core’ codebase. It may involve additional configuration and/or supplied header files.
3. The repositioning of a slave should either be seamless or require a simple configuration change.
4. General configuration should be applied using external configuration files (cycle frequency, etc).
5. The ‘server API’ will support OMA configurations/queries (name/value pairs applied to a namespace)
6. The external API should support generating events.
7. Users of the server API should not have to know about PDOs, SDOs, FMMUs, etc.
8. JNI might be a cool thing to do to make this accessible from JVMs.

# EtherCat Server API Design

Each slave represents a namespace. This is not an HCD namespace, but a sub-HCD name, so that a collection of slaves can collectively comprise an HCD. The logical namespace is mapped to a physical slave using configuration (TBD). If a slave is moved to a new position, this should not affect the slave/namespace mapping.

The external API is an object that contains the following methods:

* getDeviceNames()
* getParameterNames(deviceName)
* getParameterValue(deviceName, parameterName) :: String
* setParameterValue(deviceName, parameterName, parameterValue)
* createEventStream(deviceName, parameterName, frequency) :: eventStream
* removeEventStream(eventStream)

## Design Experiment: how to make a slave position change seamless.

Slave positions can be arbitrary and moving a slave can be seamless if each slave’s E2Prom is set to a unique alias. This can be performed using a command line tool:

*ethercat alias -p <absolute ring position> <alias>*

This command appears to set the alias for the device at the ring position *and all subsequent positioned devices*. Repeated usage of this command for each subsequent absolute ring position allows setting of all aliases.

### Task

Given that each slave has been assigned a unique alias, rewrite the code to access slaves using aliases only. The subsequent code can be tested by executing it with the slaves in the nominal positions, then again with the slaves in different positions.

DigOutSlavePos = 0,1 (no alias, position 1), changed to 1111, 0 (alias: 1111, pos 0)

Test result: test worked with alias addressing. Moved card to absolute position 4 and it continued to work because of alias addressing.

Negative test:

Returned to absolute position addressing (position 0, which is wrong once the absolute position was moved, and the result is that no lights came on)

## Design Experiment: how to add a slave without changing the core code

The big problem with the example code is that it is not clear what we need to register and how. Lets look at some examples:

We need to look at examples of running with more than one slave that works. The Digital In must be set up to work.

Once the digital-in works, we can experiment with the domain registration of PDO entries.

### Understanding Domain Registration

ecrt\_slave\_config\_reg\_pdo\_entry() can be used to programmatically register a PDO entry into a domain. This call can be used instead of ecrt\_domain\_reg\_pdo\_entry\_list().

Tested this. Registration worked.

### We need to consider whether code generation is an option or not.

### Deeper Dive: experiment with setting single bits in the domain

Why does the comment say: “the domain registration requires only the base address of the I/O on the modules”? Why is this address the first entry in the slave’s ec\_pdo\_entry\_info\_t array? (index, subindex, bit-length)

EC\_WRITE\_U8(domain1 + offset (0), either 0110 or 1001)

This writes 8 bit unsigned data to domain1 + offset. This further confuses me.

Use EC\_WRITE\_BIT(domain1+off\_dig\_out, bp\_dig\_out, value)

Output should be the correct bit to set.

Experiment conclusions:

1. Ignore the example code comment. Although the resultant domain addressing is sequential, the best way to identify the pdo entry to set is by registering each one. This is more general and makes no assumptions.
2. EC\_WRITE\_BIT works for each bit we need to set. In the completed API, we will require a case statement for each data width, calling EC\_WRITE\_xx where xx maps to the field length.

## Design Experiment: example code: what does disabling CONFIGURE\_PDOS do?

Does the default configuration really need to be set or do we just need to know what it is?

## How would the namespace mapper work?

The problem being solved here is how to give slaves a name.

1. How do we autogenerate for multiple slaves?

We use the PDO comment concatenated with the associated entries:

The offset in the pdo is in the form of : slave\_n\_pdo\_entries + p, for each index of pdo entries the corresponding highest value of p PDO text (without going over) is prepended to the PDO Entry text.

struct tmt\_ec\_slave;

typedef struct {

unsigned int vendor\_id;

unsigned int product\_code;

int slave\_alias;

int slave\_pos;

unsigned int domain\_offset[];

unsigned int domain\_bit\_pos[];

ec\_sync\_info\_t syncs[];

ec\_pdo\_info\_t pdos[];

ec\_pdo\_entry\_info\_t pdo\_entries[];

??? pdo\_entry\_names[];

} tmt\_ec\_slave\_t;

## How do we build/simulate a server in C?

This should ultimately be a C-based OPC-UA server. This would be a really cool thing to demonstrate, but we should wait until Allan has done some work here.

We can simulate a client-server interaction without actually building a server by setting up a command program that sends commands to the API according to a timed script.