Uei-Bridge application

Design, decisions, detailed specification.

## ICD

Ethernet => Device, starts with {aah, 55h).

Device => Ethernet, starts with {55h, aah).

## Names

1. Device (real) name is determines by as Uei api: GetDeviceName()
2. Block sensor device name is **“BlockSensor”**
3. Each device manager inherits DeviceName from its parent (OutputDevice or InputDevice class).
4. Device manager may use its own unique **setup**. It can define **ThisDeviceSetup** property, for example:
   * **public AO308Setup ThisDeviceSetup => \_deviceSetup as AO308Setup;**

## Digital card DIO403

48 bits. Might be configured as input or output

Read/Write is done with an array of Int16. The 8 higher bits are ignored.

Currently, lower 24 bits are considered output, and upper 24 bits are considered input. This is ‘very’ hard coded. Must at least set global constant.

**To do**: ~~when block sensor is active, this card should be dedicated to block sensor.~~done.

## Analog/Out card

**To do**: ~~when block sensor is active, this card should be dedicated to block sensor~~. Done.

## Block sensor

Block-sensor depends on A0308Manager for outputting analog value.

BlockSensor uses analog and digital card. The first devices that he finds!

## To Do

1. Unit test on EthMessage
2. DIO403, replace bits. The input bits should be in lower order. (for the sake of block sensor)
3. ~~Move device managers creation away from Program.cs~~ done.
4. Maybe it is better **not** to use Activator.CreateInstance() For device managers, you earn one and loose two. Something like DeviceManagerFactory might be more efficient.
5. ~~Config2: Use GetSetupEntryForDevice instead of getting setup entry by index.~~done.
6. Add self-inject capability to Digital Input driver (for self-test)

## Converters

**To do 1**: Dedicated converter for each card looks overwhelming. There should be Analog-Convertor which unites 308 and 201 cards, and Digital-Convertor with unites 403 and 470 cards. All others (serials) might be handled be the device manager itself.

**To do 2**: Use

public interface **IConvert2<T>**

{

T DownstreamConvert(byte[] messagePayload);

byte[] UpstreamConvert(T dt);

string DeviceName { get; }

}

Instead of IConvert,

**To do 2**: It is too heavy to enforce a convert to know about the setup of the program. Just give the needed parameters! (min voltage, max voltage, etc…)

## Using Writer/Reader adapters

AO308Manager is now using IWriterAdapter in order to write to device. Thus, is must define a class which obeys IUeiWriterAdapter and knows how to write to the physical card.

The writer is exposed to the world and thus, accommodates unit testing.

This pattern should be applied to other device mangers as well. (To do).

**public interface IWriterAdapter<T>**

## Support multiple cubes

## Device table

### Device list

List< PerDeviceObjects> ~~\_deviceObjectsTable~~ \_deviceObjectsList

There might be more the one entry per slot.

class PerDeviceObjects

{

DeviceName

SlotNumber

CubeUrl

DeviceManager (in or out)

UdpReader (for output device only)

~~SL598Session~~SessionAdapter

}

Get Slot by Device.GetIndex()

### Use case: Config

Upon startup, the main program checks to see if config file exists, if not it asks the config handler to create config files, based on known cube url's.

### Use case: Startup

1. App search for cubes according to file cubelist.txt which contains simple list of cube url's. (If this file doesn't exist, it starts scanning the nearby network.)
2. App inquire each cube for its device list and fills one-dimensional list with 'Device' and cube url. This is real-device-list.
3. App emit to logger a list of the devices.
4. For each entry in read-device-list:
   1. Is there a device manager for the device is this slot? If not, emit log and continue to next entry.
   2. Is there a config entry for this entry? If not, emit log and continue to next entry.
   3. Is the config entry type match the device? If not, emit log and continue to next entry
   4. Build per-device-objects (see use case) and add them to \_deviceManagerList

### Use case: Build per-device-objects for device/slot

1. List<PerDeviceObjects> BuildDeviceObjects( device, cubeUrl)
2. If it is serial, first create sessionObject
3. If DIO403, create both input and output manger.
4. If AO308, create output manager
5. If 470, create output manger.
6. If AI, create input manager
7. If serial, create both input and output manger.
8. After all that, add block-sensor. If block sensor enabled, redirect output of DIO403/input to block sensor and disable inputs to AO308 which is know owned by block sensor.

2V - 6553 – 0x1999

3V – 9829 – 0x2665