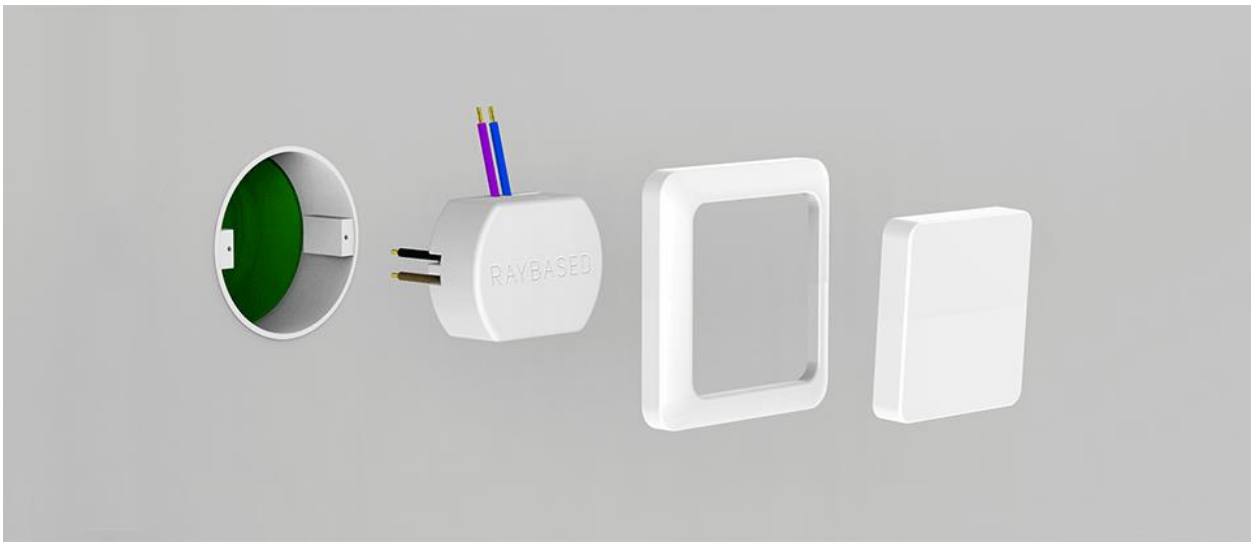
CHIPKIN

Case Study
CAS BACnet Stack
on an
Ubuntu Linux box
to provide a
Self Configuring BACnet Server



Introduction:

Chipkin implements the CAS BACnet stack on an Ubuntu Linux box for the customer. The implementation is in the form of an application which on its client side, reads data from connected RAYBASED devices using Json over HTTP. This data is used to self configure and launch a BACnet IP Server allowing remote monitoring and control of the RAYBASED system.

The BACnet server device will be a interface to a collection of 100s of physical wireless devices IoT devices.

Chipkin Automation Systems

Protocol to protocol – Enabling the IOT Internet of Things

Products that support approx. 140 major protocols. If we don't have a solution for you, we will make you one. More than a dozen customers a year have a custom driver developed for them.

Chipkin are highly regarded for their outstanding support. System integration isn't always trivial even if that is what they tell you.

The Chipkin BACnet stack comes with a 100% copyright infringement indemnity to make corporate lawyers happy. To make your engineers happy, customers get direct access to the stack developers for coaching and problem solving.

RayBased

Raybased began operations on a limited scale in 2011. Since then, Raybased has developed an open wireless platform for advanced building automation.

The system makes it possible to design applications that control and optimize building functions, such as heating, ventilation, lighting, and security. Raybased thus combines the Internet of Things with building automation.

Raybased primarily targets reconstruction of existing commercial properties. These often have a great need of improved energy efficiency but struggle to motivate the high total cost that professional systems entail.

Project Details:

Raybased is creating a REST JSON API that can be used to get the current values of various sensor data as well as set and control setpoints and relays.

Our task is to integrate the data to BACnet using the CAS BACnet stack.
They require a Linux application built in Ubuntu 14.04.

We used the following libraries

- CAS BACnet API - provides BACnet integration
- Curl - handles building, sending, and decoding web requests.
- Rapidjson - handles parsing json documents.

Raybased exposed 3 main REST API endpoints for the BACnet application to poll.

1. Metadata
2. Read
3. Write

Metadata

The metadata endpoint is used to initially setup the BACnet interface. All settings, device properties, objects, and object properties are described in the json metadata endpoint and are used to build both an internal memory database to store the properties and their values as well as the BACnet device and objects that get registered in the CAS BACnet API.

The metadata is also polled on a configurable interval in the event that additional sensors and devices are added to the Raybased network. These updates are then added to the memory database and registered to the CAS BACnet API

An example of the metadata request:
http://{ipaddress:port}/api/v1/bacnet/metadata


```
{
        "result": "OK",
        "value": {
                "bacnetNetwork": 0,
                "bbmdIPAddress": "127.0.0.1",
                "bbmdPort": 47808,
                "device": {
                        "description": "",
                        "instance": 1,
                        "location": "",
                        "name": "DeviceName",
                        "objectList": [{
                                "description": "dummy description",
```



```
                "instance": 2,
                "name": "E00C - Temperature",
                "objectType": "analogInput",
                "units": "degreesCelsius"
            }, {
                "description": "dummy description",
                "instance": 3,
                "name": "E00C - Humidity",
                "objectType": "analogInput",
                "units": "percentRelativeHumidity"
            }, {
                "description": "dummy description",
                "instance": 4,
                "name": "E00C - Presence",
                "objectType": "binaryInput",
                "units": "noUnits"
            }, {
                "description": "dummy description",
                "instance": 5,
                "name": "E00C - Illuminance",
                "objectType": "analogInput",
                "units": "luxes"
            }, {
                "description": "dummy description",
                "instance": 6,
                "name": "E010 - RelayStatus",
                "objectType": "binaryInput",
                "units": "noUnits"
            }, {
                "description": "dummy description",
                "instance": 7,
                "name": "E010 - RelayStatus",
                "objectType": "binaryOutput",
                "units": "noUnits"
            }, {
                "description": "E0011 description",
                "instance": 8,
                "name": "E011 - Temperature",
                "objectType": "analogOutput",
                "units": "degreesCelsius"
            }]
    },
```



```
                "foreignDeviceTimeToLive": 600,
                "metadataPollInterval": 900000,
                "sensorPollInterval": 300000,
                "udpPort": 47808
        }
}
```


Read

The read endpoint contains the current values of all the data points that were mapped to BACnet objects. The endpoint also contains the reliability of the data so should sensors go offline, the data and reliability BACnet properties can be updated to properly reflect that the data is unreliable. The data is stored in the memory database, and is then made available via BACnet.

In the future, Raybased may change the read endpoint to only include values that have changed. As such, as part of the read request, we also send the last updated timestamp that we received in the most recent read response. Raybased handles the filtering of the data that is sent.

An example of the read request:
http://{ipaddress:port}/api/v1/bacnet/read?lastUpdated=123456789


```
{"result": "OK", "timestamp": "123456789"
"values": [
   {
      "instance": 1,
      "presentValue": 25.3,
      "reliability": "no-fault-detected"
   },
   {
      "instance": 2,
      "presentValue": 45.0,
      "reliability": "no-fault-detected"
   }
]
}
```


Write

The write functionality allows the BACnet application to send commands to the REST API to change a setpoint or to activate/deactivate a relay. These are mapped to Analog-Outputs and Binary-Outputs. When the BACnet application receives a WriteProperty, it attempts to send the write request to the web server and if successful, sets the value in the memory database.

The response to the write request contains the result of the write operation. If the operation failed, then the json response contains the error message of the reason for the failure that is then translated into a BACnet error.

An example of the write request:
http://{ipaddress:port}/api/v1/bacnet/write/7

Settings

The metadata contains the following settings:

BACnet Settings

- bacnetNetwork - the network to set the BACnet device on.
- udpPort - the UDP port to use for BACnet communication.

Polling Settings

- metadataPollInterval - how often to poll for the metadata in milliseconds.
- sensorPollInterval - how often to poll for the read data in milliseconds.

Future BBMD Settings

Raybased expressed interest in implementing Foreign Device Registration. As such, they will need to add the following settings:

- bbmdIPAddress - the IP address of the BBMD the device is trying to register to as a Foreign Device.
- bbmdPort - the port of the BBMD device
- foreignDeviceTimeToLive - how often to send the RegisterForeignDevice BACnet message.

