Purpose of this document - currently PHASE 1

PHASE 1
To define a C API which enables easy integration of the Octave system into basic microcontroller projects

PHASE 2
In addition describe how the C API wraps the ORP (Octave Resource Protocol) wrt to the remote microcontroller

Goals

- Code written in as generic as possible C
- Targets microcontrollers only
- Doesn't require an OS
- The API should be simple to use
- The final code should be portable to other microcontrollers
- Limited set of host microcontroller software dependencies
- Implemented so that the users code provides buffer resources (no mallocs etc in the provided code)

Description

End users will include the code in their project and use the API to

- Send data to Octave
- Receive unsolicited data from Octave

The host microcontroller will provide certain interfaces for example serial port / UART interfaces - full info TBD

Limitations

It is proposed at this stage to implement a subset of the ORP protocol supporting only

- Octave inputs
- Octave outputs

Terms

Remote - this is the microcontroller Octave Edge - this the WP7702 or whatever Octave device is in the field
Octave Cloud - this is Sierra's cloud compute sytem

C API design philosophy

KEY / VALUE pairs will be implemented as strings

Values and Keys will be truncated at a size defined by C storage. The user will need to handle any errors caused by truncation

Octave output data

Data received from Octave is restricted to individual KEY / VALUE pairs. This removes the need to decode incoming data in a difficult to parse in C format like JSON.

My thinking is that if a consistent KEY format is used it will be easy to post process the data in the Octave cloud or the edge.

Example output VALUE
Note that strings have been used to represent all types.

Numbers - expect a float

> "101"
> "23.27"

Strings

> "powerOff"
> "Display this"

Bool

> "true"
> "false"

Example output KEY
A suggestion is that a meaningful KEY name is used which describes both the function and the datatype

A suggestion is to embed the datatype in the KEY path

> str
> num
> bool

Remote device KEY example

> adc1/num

Appears as an Octave path like this

> /remote/adc1/num/value

Octave inputs

Data sent to Octave is encoded in simple JSON strings as it's easy to encode in C using prebuilt functions like snprintf(). By using JSON the value encoding used in JSON can be used as JSON data is supported by the Octave edge system.

For example the following JSON encoded data could be sent to the Octave edge

{“temperature”:123.2,“humidity”:70,“pressure”:9997,“powerOn”:true}

Octave serial ORP interaction - remote initiated

The basic Octave ORP cycle is

1. Remote sends a message to Octave Edge
2. Octave edge responds with an ack / nack message

It is proposed to handle this cycle by implementing

- Blocking function sends an ORP message to Octave Edge then returns
- The Octave system then processes the ORP message

C API

```
// Low level platform dependencies
/// serial port from Octave to Micro

The target micro provides these functions to the micro Octave remote C code
serial_rxByte(uint8_t data);
serial_rxByte(uint8_t data),

// The micro Octave remote C API
octave_init(
    &serial_txByte(uint8_t),
    &serial_rxByte(uint8_t data),
    const uint8_t *HDLC_frameBuffer,
    uint16_t HDLC_frameBuffer_length),
    uint16_t HDLC_maxFrameLength,      // maybe hide this?

)

// Outputs are output from Octave
octaveResponse  octave_registerOutput(char * key, outputCallbackHandler);

*prototype of callback*
void outputCallbackHandler (char * key, char * value);

octaveResponse  octave_registerInput(char * key, char * value);
```

Notes

Is this blocking or none blocking What about failures?