Solar Cube is a java-script/HTML visualization module for the MBMC solar power monitor system. The web server is hosted on a PIC32 controller that receives system data from a PIC18 primary DAQ system that monitors and controls the charge controller , (controller and energy bank) batteries with remote DAQ TCP clients monitoring the remote panels.

One requirement of the system is a simple way to tell at a glance the current operating mode and relative power levels using a mobile device like Android tablets or phones. Tabular or graphic chart displays of data are great for understanding complex systems data in detail but I wanted a intuitive non-numeric display of the most important items of current system data using symbolic visual parameters of power flow/direction, voltage and current levels. For this I chose a 3D cube (currently wire-frame) model and a 2D screen-view of the color, size (Z) and X/Y rotation speed of that cube.

There are several levels of processing needed to display the data on the browser. The Cube display uses three.js as the JavaScript 3D library. That library and the display program run on the browser of the display device so the PIC32 only has to upload the code.
https://github.com/mrdoob/three.js/

From the server JavaScript code: (The first number is a callback function code for the web server)

```
 //     We need several parameters to visualize the system operation
 //     130   z       cube size 0,1000:            changes the camera z to the absolute power of the
system
 //     131   mycolor       cube base color 0x000000:   HTML HEX color codes, green shades
positive solar power in-flow,
//                                                  red shades load power out-flow, blue shades
positive charger power in-flow
//      132   x       cube rotation speed 0,10:    solar voltage indicator
//      133   y       cube rotation speed 0,10:    solar or battery current indicator
...
```

This puts the PIC32 data in the correct format for the JavaScript program to use.

```
 camera.position.z = ~mbmcdata(130)~; // this sets up the data for other cube functions so it must be
first
 mycolor = ~mbmcdata(131)~;
 mesh.rotation.x += (0.03 * ~mbmcdata(132)~);
 mesh.rotation.y += (0.03 * ~mbmcdata(133)~);
 mesh.material.color.setHex(mycolor);
```

or the equivalent XML variable data (cube(x)) that's updated a few times a second to animate the cube in real-time.

XML data exchange file:

```
<response>
<led0>~led(0)~</led0>
<led1>~led(1)~</led1>
<led2>~led(2)~</led2>
<led3>~led(3)~</led3>
```

```
<led4>~led(4)~</led4>
<led5>~led(5)~</led5>
<led6>~led(6)~</led6>
<led7>~led(7)~</led7>
<btn0>~btn(0)~</btn0>
<btn1>~btn(1)~</btn1>
<btn2>~btn(2)~</btn2>
<btn3>~btn(3)~</btn3>
<pot0>~pot~</pot0>
<cube0>~mbmcdata(130)~</cube0>
<cube1>~mbmcdata(131)~</cube1>
<cube2>~mbmcdata(132)~</cube2>
<cube3>~mbmcdata(133)~</cube3>
</response>
```

…
The PIC32 C code for a typical callback function: It replaces ~mbmcdata(133)~ with a string of the current data.

```
        case 133:
                // fixup battery charging current for Solar Cube
                mbmc_buffer.current = mbmc_buffer.currentin; // default on PV current to CC
                if (mbmc_buffer.boc == 1) { // charging 1
                        mbmc_buffer.current = cell[1].current;
                }
                if (mbmc_buffer.boc == 2) { // charging 2
                        mbmc_buffer.current = cell[2].current;
                }
                mbmc_put(mbmc_buffer.current / 5, 2);
                break;
```

The PIC18 C ADC code that actually reads the current sensor. This data is processed, formatted and sent to the PIC32 as a high speed RS232 9bit data-stream with CRC error checking.

```
...
        ADC_zero();
        // very long measurement sample time to get several PWM cycles from CC
        Vin = ADC_get(ADC_CH5, ADC_SAMP_S, ADC5_OFF, adc_cal[5], 1, TRUE); // RF0  for
input current AMP300, battery current
        a300 = Vin; // raw ADC value
```

Some of the PIC18 data-stream structures:

```
...
// SDCARD data format structure using TypeDefs for pic32 host
#if defined(__18CXX)

typedef struct mbmcdata { // C18 uses byte alignment
```

```
#else

typedef struct __attribute__((aligned(1))) mbmcdata { // force C32 to byte alignment, 32bit alignment
is still critical
#endif
    int32_t ccvoltage, inputvoltage, primarypower_B1, primarypower_B2, systemvoltage;
    int32_t currentin, current, currentload;
    int32_t current_B1, current_B2;
    int32_t thermo_batt, cef_boc; // cef*100
    int32_t PRIPOWEROK, DIPSW;
    int32_t pick, boi, boc, alert, bn;
    int32_t misc1, misc2;
    int32_t MBMCID, UTC;
    struct harvesttype harvest;
    struct portstype ports;
    struct diversiontype diversion;
    int32_t crc;
}
volatile mbmctype;
```

The current project JavaScript is located here: https://github.com/nsaspook/mbmc/blob/master/TCPIP%20Demo%20App/WebPages2/dynvars_a.htm