3. Publisher tutorial

In this article we will turn Zubax Babel into a simple UAVCAN publisher. All the code here includes the code from the previous tutorials and extends it.

> **OBSOLETE**
> THE MATERIALS PRESENTED HERE ARE BASED ON **UAVCAN v0** – AN EXPERIMENTAL VERSION OF THE PROTOCOL THAT IS NOW OBSOLETE. TO LEARN ABOUT THE LONG-TERM STABLE REVISION OF THE PROTOCOL, PROCEED TO UAVCAN.ORG.

Goal

Our goal is to publish a couple of different values via UAVCAN and to check if everything works properly by plotting graphs in the UAVCAN GUI Tool's plotter utility.

Implementation

We are going to use the UAVCAN message `uavcan.protocol.debug.KeyValue`. The UAVCAN specification says that `float32` values can be broadcasted this way. We can use it to broadcast some custom sensor data, ADC data, or just any named value. For the sake of simplicity, in this tutorial we will broadcast sine values. But, assuming that the MCU resources are quite constrained, we will take the values from a lookup table. We will also broadcast the second value – the current table index.

canard_publish

```
#define PUBLISHER_PERIOD_mS                             25
#define UAVCAN_PROTOCOL_DEBUG_KEYVALUE_ID               16370
#define UAVCAN_PROTOCOL_DEBUG_KEYVALUE_SIGNATURE        0xe02f25d6e0c98ae0
#define UAVCAN_PROTOCOL_DEBUG_KEYVALUE_MESSAGE_SIZE     62

static void publishCanard(void)
{
    static uint32_t publish_time = 0;
    static int step = 0;
    if(getUptime() < publish_time + PUBLISHER_PERIOD_mS) { return; }    //
rate limiting
    publish_time = getUptime();

    uint8_t buffer[UAVCAN_PROTOCOL_DEBUG_KEYVALUE_MESSAGE_SIZE];
    memset(buffer,0x00,UAVCAN_PROTOCOL_DEBUG_KEYVALUE_MESSAGE_SIZE);
    step++;
    if (step == 256)
    {
        step = 0;
    }

    float val = sine_wave[step];
    static uint8_t transfer_id = 0;
    canardEncodeScalar(buffer, 0, 32, &val);
    memcpy(&buffer[4], "sin", 3);
    canardBroadcast(&g_canard,
                    UAVCAN_PROTOCOL_DEBUG_KEYVALUE_SIGNATURE,
                    UAVCAN_PROTOCOL_DEBUG_KEYVALUE_ID,
                    &transfer_id,
                    CANARD_TRANSFER_PRIORITY_LOW,
                    &buffer[0],
                    7);
    memset(buffer,0x00,UAVCAN_PROTOCOL_DEBUG_KEYVALUE_MESSAGE_SIZE);

    val = step;
    canardEncodeScalar(buffer, 0, 32, &val);
    memcpy(&buffer[4], "stp", 3);
    canardBroadcast(&g_canard,
                    UAVCAN_PROTOCOL_DEBUG_KEYVALUE_SIGNATURE,
                    UAVCAN_PROTOCOL_DEBUG_KEYVALUE_ID,
                    &transfer_id,
                    CANARD_TRANSFER_PRIORITY_LOW,
                    &buffer[0],
                    7);
}
```

Important note. Integer and float values have very different bit-structure. As libcanard expects a float data type for `uavcan.protocol.debug.KeyValue`, it is important to give it exactly what it wants - a float value. So, despite the fact that we have an unsigned integer (even `uint8_t`) typed sine table, it is important to provide the `canardEncodeScalar` function with a `float` type parameter.

Now, let's add our publisher to `main()`, so now it looks like this:

main

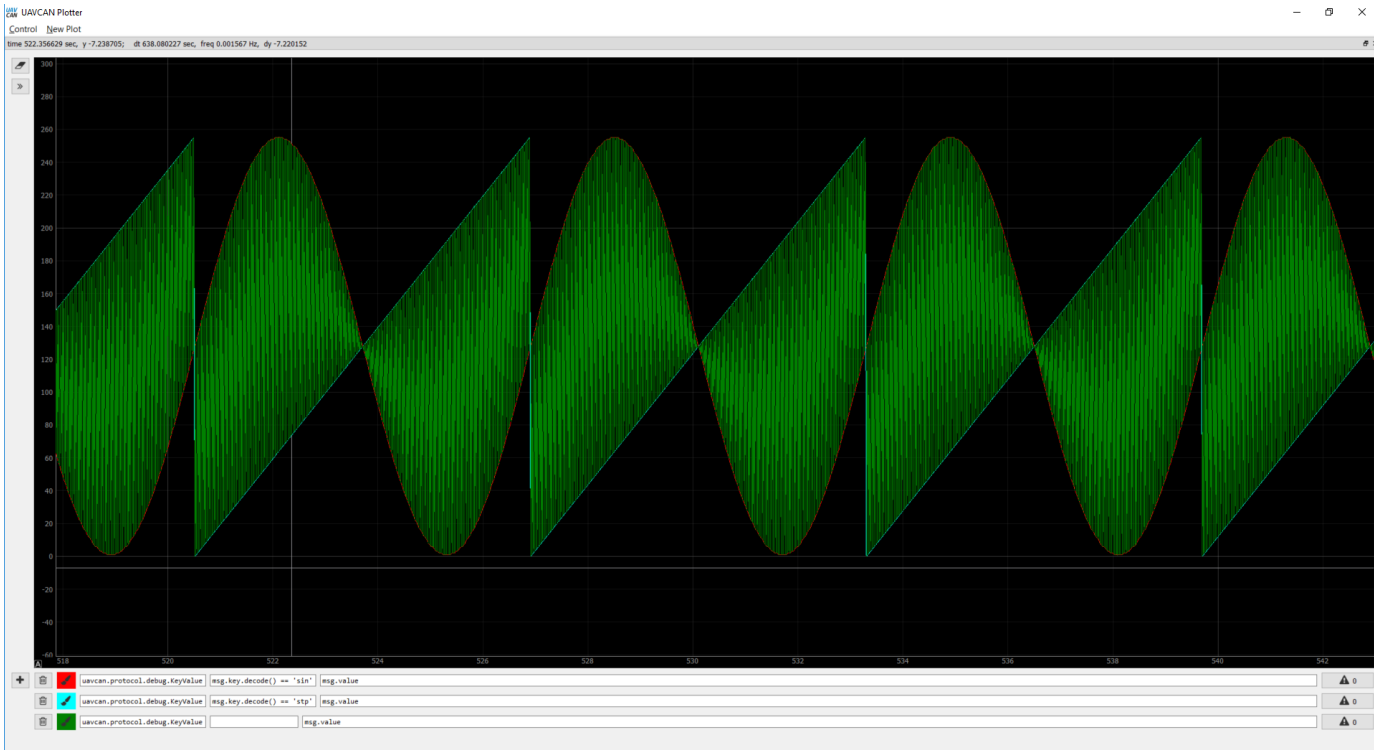
```
int main(void)
{
    /*!< At this stage the microcontroller's clock setting is already
configured,
       this is done through SystemInit() function which is called from
startup
       file (startup_stm32f37x.s) before branching to the application's
main().
       To reconfigure the default setting of SystemInit() function, refer
to the
       system_stm32f37x.c file */
    RCC_GetClocksFreq(&RCC_Clocks); //To make sure RCC is initialised
properly
    hwInit();
    swInit();
    SysTick_Config(SystemCoreClock / 1000);
    while(1)
    {
        sendCanard();
        receiveCanard();
        spinCanard();
        publishCanard();
    }
}
```

Now it's time to flash the firmware to Babel and check if everything is working as expected. You should find a bunch of new messages in the Bus monitor panel:

Now let's open `UAVCAN Plotter`, which can be found in the `Tools -> Plotter` menu.

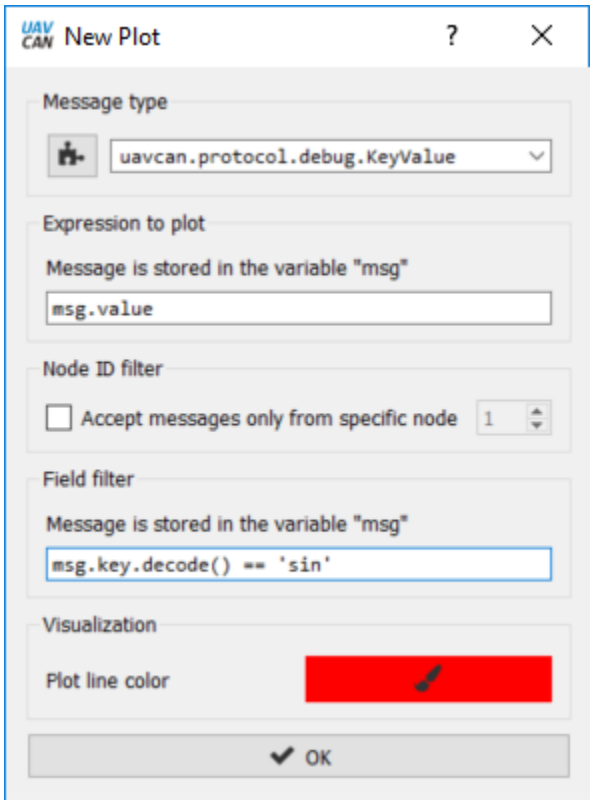
Firstly, let's just plot every message of type `uavcan.protocol.debug.KeyValue`. For that, open the Plotter and hit that + button in bottom left corner of the window. Fill in the plot details as shown below and hit `OK`:

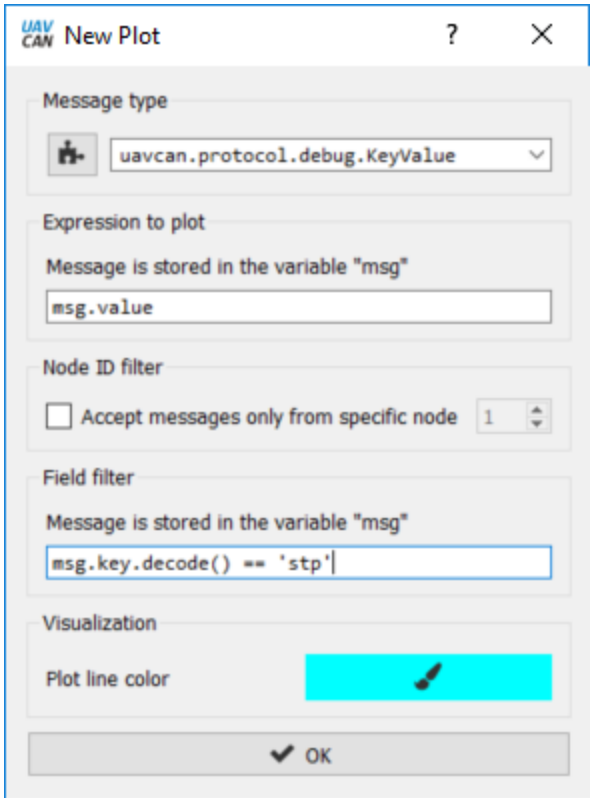
Now you should see something like this:



> **Important note**
> You may adjust the scale of time and value axes by moving the mouse while holding the right button pressed.

The plot looks pretty weird because the plotter tries to display all values of type `debug.KeyValue`, and Babel transmits two diifferent values simultaneously: the sine value and the step. Let's delete this plot and add two more, one for each separate value:





Now the graphs look properly: