4. Service tutorial

Introduction

In this article we will implement a UAVCAN service.

> **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

The most obvious example of a service in UAVCAN is the node configuration service. We will use the `uavcan.protocol.param.GetSet` data type to read and write some parameters in Babel.

Implementation

So let's give our Zubax Babel some parameters that can be changed from the UAVCAN GUI Tool. For the sake of simplicity, in this example we will use only integer numeric parameters, although UAVCAN allows you to have parameters of just any type you want.

parameters type

```
typedef struct
{
    uint8_t* name;
    int64_t val;
    int64_t min;
    int64_t max;
    int64_t defval;
} param_t;
```

In this example we will have three integer parameters. It is not necessary but it can be considered a good practice to specify the default value and the acceptable value range for numeric parameters.

parameters

```
static param_t parameters[] =
{
    {"param0", 0, 10,20, 15},
    {"param1", 1, 0, 100, 25},
    {"param2", 2, 2, 8,  3 },
};
```

We will also need a couple of ways to access these parameters: by index and by name with some safety checks.

parameters access

```
static inline param_t* getParamByIndex(uint16_t index)
{
    if (index >= ARRAY_SIZE(parameters))
    {
        return NULL;
    }
    return &parameters[index];
}


static inline param_t* getParamByName(uint8_t * name)
{
    for (uint16_t i = 0; i < ARRAY_SIZE(parameters); i++)
    {
        if (strncmp(name, parameters[i].name, strlen(parameters[i].name))
== 0)
        {
              return &parameters[i];
        }
    }
    return NULL;
}
```

Now we need to add a new type of UAVCAN messages that we want to process to the function `shouldAcceptTransfer` and add a handler call to the function `onTransferReceived`.

shouldAcceptTransfer

```
static bool shouldAcceptTransfer(const CanardInstance* ins,
                                 uint64_t* out_data_type_signature,
                                 uint16_t data_type_id,
                                 CanardTransferType transfer_type,
                                 uint8_t source_node_id)
{
    if ((transfer_type == CanardTransferTypeRequest) &&
        (data_type_id == UAVCAN_GET_NODE_INFO_DATA_TYPE_ID))
    {
        *out_data_type_signature =
UAVCAN_GET_NODE_INFO_DATA_TYPE_SIGNATURE;
        return true;
    }

    if (data_type_id == UAVCAN_EQUIPMENT_ESC_RAWCOMMAND_ID)
    {
        *out_data_type_signature =
UAVCAN_EQUIPMENT_ESC_RAWCOMMAND_SIGNATURE;
        return true;
    }

    if (data_type_id == UAVCAN_PROTOCOL_PARAM_GETSET_ID)
    {
        *out_data_type_signature = UAVCAN_PROTOCOL_PARAM_GETSET_SIGNATURE;
        return true;
    }
    return false;
}
```

onTransferReceived

```
static void onTransferReceived(CanardInstance* ins, CanardRxTransfer*
transfer)
{
    if ((transfer->transfer_type == CanardTransferTypeRequest) &&
        (transfer->data_type_id == UAVCAN_GET_NODE_INFO_DATA_TYPE_ID))
    {
        getNodeInfoHandleCanard(transfer);
    }

    if (transfer->data_type_id == UAVCAN_EQUIPMENT_ESC_RAWCOMMAND_ID)
    {
        rawcmdHandleCanard(transfer);
    }

    if (transfer->data_type_id == UAVCAN_PROTOCOL_PARAM_GETSET_ID)
    {
        getsetHandleCanard(transfer);
    }
}
```

We should also write a handler for `UAVCAN_PROTOCOL_PARAM_GETSET_ID`. Here it is:

canard_getset_handle

```
static void canardGetsetHandle(CanardRxTransfer* transfer)
{
    uint16_t index = 0xFFFF;
    uint8_t tag    = 0;
    int offset     = 0;
    int64_t val    = 0;

    canardDecodeScalar(transfer, offset,  13, false, &index);
    offset += 13;
    canardDecodeScalar(transfer, offset, 3, false, &tag);
    offset += 3;

    if (tag == 1)
    {
        canardDecodeScalar(transfer, offset, 64, false, &val);
        offset += 64;
    }

    uint16_t n = transfer->payload_len - offset / 8 ;
    uint8_t name[16] = "";
    for (int i = 0; i < n; i++)
    {
        canardDecodeScalar(transfer, offset, 8, false, &name[i]);
        offset += 8;
    }
```

```
    param_t* p = NULL;

    if (strlen((char const*)name))
    {
        p = getParamByName(name);
    }
    else
    {
        p = getParamByIndex(index);
    }

    if ((p)&&(tag == 1))
    {
        p->val = val;
    }

    uint8_t  buffer[64] = "";
    uint16_t len = canardEncodeParam(p, buffer);
    int result = canardRequestOrRespond(&g_canard,
                                        transfer->source_node_id,

UAVCAN_PROTOCOL_PARAM_GETSET_SIGNATURE,
                                        UAVCAN_PROTOCOL_PARAM_GETSET_ID,
                                        &transfer->transfer_id,
                                        transfer->priority,
                                        CanardResponse,
                                        &buffer[0],
```

```
                                        (uint16_t)len);

}
```

We also need to create a function that encodes an integer param to the `uavcan.protocol.param.GetSet` message. It will be a little nasty, as bit operations always are, but here it is.

canardEncodeParam

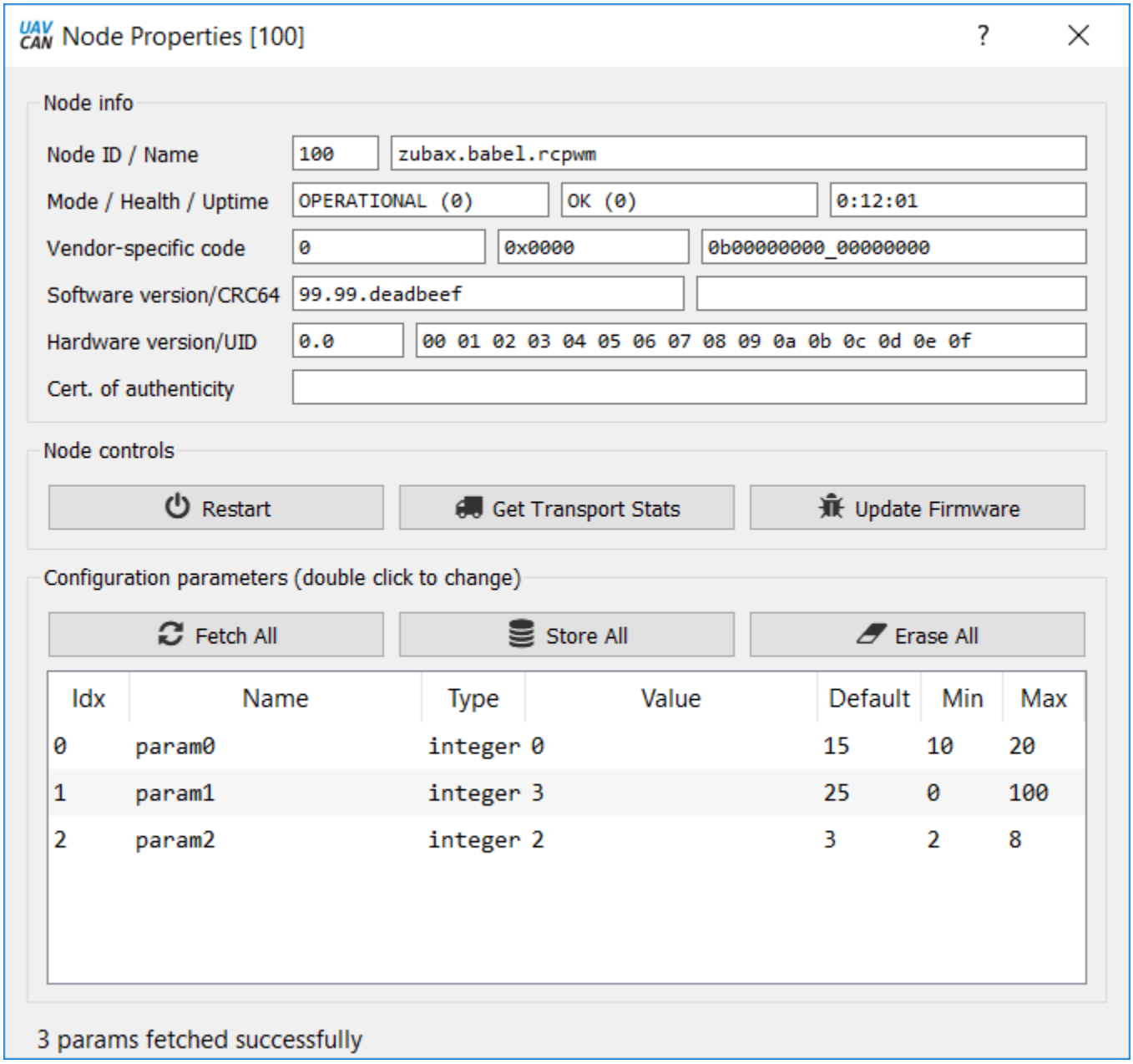
```
static uint16_t encodeParamCanard(param_t * p, uint8_t * buffer)
{
    uint8_t n     = 0;
    int offset    = 0;
    uint8_t tag   = 1;

    if (p == NULL)
    {
        tag = 0;
        canardEncodeScalar(buffer, offset, 5, &n);
        offset += 5;
        canardEncodeScalar(buffer, offset,3, &tag);
        offset += 3;

        canardEncodeScalar(buffer, offset, 6, &n);
        offset += 6;
        canardEncodeScalar(buffer, offset,2, &tag);
        offset += 2;

        canardEncodeScalar(buffer, offset, 6, &n);
        offset += 6;
        canardEncodeScalar(buffer, offset, 2, &tag);
        offset += 2;
        buffer[offset / 8] = 0;
        return ( offset / 8 + 1 );
    }

    canardEncodeScalar(buffer, offset, 5,&n);
    offset += 5;
    canardEncodeScalar(buffer, offset, 3, &tag);
    offset += 3;
    canardEncodeScalar(buffer, offset, 64, &p->val);
    offset += 64;

    canardEncodeScalar(buffer, offset, 5, &n);
    offset += 5;
    canardEncodeScalar(buffer, offset, 3, &tag);
    offset += 3;
    canardEncodeScalar(buffer, offset, 64, &p->defval);
    offset += 64;

    canardEncodeScalar(buffer, offset, 6, &n);
    offset += 6;
```

```
canardEncodeScalar(buffer, offset, 2, &tag);
offset += 2;
canardEncodeScalar(buffer, offset, 64, &p->max);
offset += 64;

canardEncodeScalar(buffer, offset, 6, &n);
offset += 6;
canardEncodeScalar(buffer, offset,2,&tag);
offset += 2;
canardEncodeScalar(buffer, offset,64,&p->min);
offset += 64;

memcpy(&buffer[offset / 8], p->name, strlen((char const*)p->name));
```

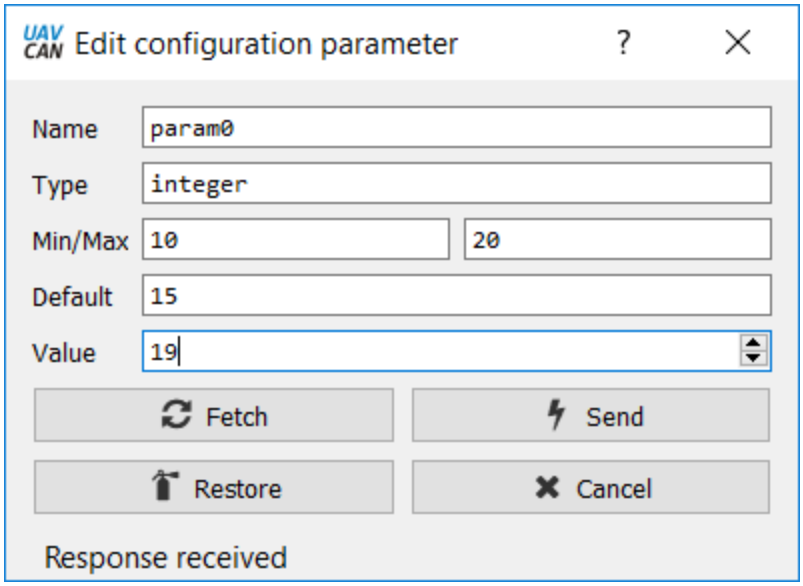
```
    /* See important note below */
    return  (offset/8 + strlen((char const*)p->name));
}
```

According to the `UAVCAN DSDL specification`, section "Dynamic arrays", there should be a bit field (often 8 bits wide) representing the length of the array prepending the array field. But there is one important detail, which plays a role in this particular case. DSDL also defines a `tail array optimization`, which means that in the case when the array is the last field in the UAVCAN message, there is no need to specify its length and it must be skipped. That is why in the function above we did not specify the length of the parameter name.

Now its time to go to the UAVCAN GUI Tool and check our newly-created parameters. Double-click on the node record in the online nodes table and you will see "Node properties" window. Click the button `Fetch all` to read all parameters from Babel. You should see something like this:



Now double click on any paramter and try changing its value.



Now close the "Node properties" window and re-open it and refetch all the parameters to make sure you have actually updated the parameter value.