CBUS Library for Arduino

Introduction

This set of libraries implements a complete CBUS module using the Arduino environment. A minimum of additional code is required to create a fully-functional FLiM-compliant module.

CBUS	- an abstract base class containing the commom methods
CBUS2515	- an implementation of CBUS specifically for the MCP2515/25625 controller
CBUSLED	- non-blocking LED management
CBUSswitch	- non-blocking pushbutton switch management
CBUSconfig	- event and node variable storage in on-chip or external EEPROM

No additional code need be written to integrate with FCU or JMRI or to learn events.

Features include:

- CBUS switch and FLiM/SLiM LEDs
- transition to and from FLiM/SLiM, tested with FCU and JMRI Node Manager
- event learning and storage, with configurable number of event variables
- node variable (NV) configuration and storage
- storage can use the on-chip 1K EEPROM or external I2C EEPROM up to 64K
- reset capability to return the module to an empty configuration
- a user-assignable function to be called when a learned event is received
- power-on reset capability to empty any stored events and NVs

All four libraries a required although the LED and switch libraries can be used standalone in other projects if you find them useful. The download links are:

CBUS	- https://github.com/obdevel/CBUS
CBUS2515	- https://github.com/obdevel/CBUS2515
CBUSLED	- https://github.com/obdevel/CBUSLED
CBUSswitch	- https://github.com/obdevel/CBUSswitch
CBUSconfig	- https://github.com/obdevel/CBUSconfig

You will also need to install the following two 3rd party libraries:

ACAN2515	- https://github.com/pierremolinaro/acan2515
Streaming	- https://github.com/janelia-arduino/Streaming

For people who may have used other CAN bus libraries (e.g. MCP_CAN), note that the ACAN2515 library implements interrupt handling and configurable send/receive buffers, so there is no need to code this yourself.

Hardware

The minimum hardware requirement to create a CBUS module is:

- an Arduino processor board, e.g. Uno, Nano, Mega, Pro Mini, etc

- a CAN bus module based on the MCP2515 chip (available from multiple eBay sellers)
- two LEDs (green and yellow/amber) with 1K resistors
- a pushbutton switch

As an alternative, I have designed a generic through-hole PCB containing all the above. It has no module-specific components, but all spare IO pins are brought out to headers. The design files are available on the MERG wiki at:
https://www.merg.org.uk/merg_wiki/doku.php?id=projects:canxmas

If using separate components, connect up as follows:

Arduino Uno pin	Module pin
5V	Vcc
GND	GND
10 (SS)	CS
12 (MISO)	SO
11 (MOSI)	SI
13 (SCK)	SCK
2 (INT0)	INT

Connect the green LED with its current-limiting resistor between Arduino pin 4 and GND
Connect the yellow LED with its current-limiting resistor between Arduino pin 5 and GND
Connect the pushbutton switch between Arduino pin 6 and GND

You may find a breadboard handy for the connections.

Using the library

The CBUS library includes a starter sketch in the example folder. This creates a complete but ‘empty’ module with no specific personality.

Here is a commentary on the example code:

1. include the required libraries

```
// 3rd party libraries
#include <Streaming.h>

// CBUS library header files
#include <CBUS2515.h>             // CAN controller and CBUS class
#include <CBUSswitch.h>           // pushbutton switch
#include <CBUSLED.h>              // CBUS LEDs
#include <CBUSconfig.h>           // module configuration
#include <cbusdefs.h>             // MERG CBUS constants

// local header
#include "defs.h"
```

2. create the CBUS objects

```
CBUS2515 CBUS;                    // CBUS object
CBUSConfig config;                // configuration object
CBUSLED ledGrn, ledYlw;           // LED objects
CBUSSwitch pb_switch;             // switch object
```

3. create the module parameter variables and set the module’s name:

```
// CBUS module parameters
unsigned char params[21];

// module name, e.g. CANEMPTY ☺
unsigned char mname[7] = { 'E', 'M', 'P', 'T', 'Y', ' ', ' ' };
```

4. in the setup() function:

(a) set the module’s event and NV storage configuration:

```
// set config layout parameters
config.EE_NVS_START = 10;
config.EE_NUM_NVS = 10;
config.EE_EVENTS_START = 50;
config.EE_MAX_EVENTS = 64;
config.EE_NUM_EVS = 1;
config.EE_BYTES_PER_EVENT = (config.EE_NUM_EVS + 4);

// initialise and load configuration
config.setEEPROMtype(EEPROM_INTERNAL);
config.begin();
```

(b) set the module’s parameters:

```
// set module parameters
params[0] = 20;                      //  0 num params
params[1] = 0xa5;                    //  1 manf = MERG, 165
params[2] = VER_MIN;                 //  2 code minor version
params[3] = MODULE_ID;               //  3 module id, 99 = undefined
params[4] = config.EE_MAX_EVENTS;    //  4 num events
params[5] = config.EE_NUM_EVS;       //  5 num evs per event
params[6] = config.EE_NUM_NVS;       //  6 num NVs
params[7] = VER_MAJ;                 //  7 code major version
params[8] = 0x05;                    //  8 flags = 5, FLiM, consumer
params[9] = 0x32;                    //  9 processor id = 50
params[10] = PB_CAN;                 // 10 interface protocol = CAN, 1
params[11] = 0x00;
params[12] = 0x00;
params[13] = 0x00;
params[14] = 0x00;
params[15] = '3';
params[16] = '2';
params[17] = '8';
params[18] = 'P';
params[19] = CPUM_ATMEL;
params[20] = VER_BETA;

// assign to CBUS
CBUS.setParams(params);
CBUS.setName(mname);
```

(c) configure the pushbutton switch and check for power-on reset:

```
// initialise CBUS switch
pb_switch.setPin(SWITCH0, LOW);

// module reset - if switch is depressed at startup and module is in SLiM mode
pb_switch.run();
```

```
if (pb_switch.isPressed() && !config.FLiM) {
  Serial << F("> switch was pressed at startup in SLiM mode") << endl;
  config.resetModule(ledGrn, ledYlw, pb_switch);
}
```

(d) register the user-defined function to be called when a learned event is received:

```
// register our CBUS event handler, to receive event messages of learned events
CBUS.setEventHandler(eventhandler);
```

(e) configure the LED pins and indicate the current module mode (FLiM or SLiM):

```
// set LED and switch pins and assign to CBUS
ledGrn.setPin(LED_GRN);
ledYlw.setPin(LED_YLW);
CBUS.setLEDs(ledGrn, ledYlw);
CBUS.setSwitch(pb_switch);

// set CBUS LEDs to indicate the current mode
CBUS.indicateMode(config.FLiM);
```

(f) configure the CAN bus parameters and start CBUS message processing:

```
// configure and start CAN bus and CBUS message processing
CBUS.setNumBuffers(4);
CBUS.setPins(10, 2);
CBUS.begin();
```

4. implement a simple loop() function:

```
void loop() {

  //
  /// do CBUS message, switch and LED processing
  //

  CBUS.process();
}
```

5. implement a simple user-defined function to handle received events (note that EVs and NVs number from one, not zero):

```
void eventhandler(byte index, CANFrame *msg) {

  // as an example, display the opcode and first EV of this event

  Serial << F("> event handler: index = ") << index << F(", opcode = 0x") << _HEX(msg-
>data[0]) << endl;

  byte ev = 1;
  byte eeaddress = config.EE_EVENTS_START + (index * config.EE_BYTES_PER_EVENT) + 4 + (ev -
1);
  Serial << F("> EV1 = ") << config.readEEPROM(eeaddress) << endl;

  return;
}
```

Header File

The example sketch includes a module-specific header file (defs.h) which defines some local constants, as follows:

```
#include <Arduino.h>                          // for definition of byte datatype

// constants
static const byte VER_MAJ = 1;                // code major version
static const char VER_MIN = 'a';              // code minor version
static const byte VER_BETA = 0;               // code beta sub-version
static const byte MODULE_ID = 99;             // CBUS module type

static const byte LED_GRN = 4;                // CBUS green SLiM LED pin
static const byte LED_YLW = 5;                // CBUS yellow FLiM LED pin

static const byte SWITCH0 = 6;                // CBUS push button switch pin
```

Adding the module's personality

The foregoing implements a module that has no useful functionality.

Consumer Modules

The user-defined function is called whenever a message that matches a learned event is received; this is the entry point for implementing a Consumer module. It goes without saying that this function will not be called until the module has been taught at least one event.

The following useful data is passed to this function when it is called:

```
byte index;
```

This is the index into the module's event table. The Event Variables (EVs) can then be located.

```
CANFrame *msg;
```

A pointer to a CANFrame object containing the following:

```
msg->id                   // the CANID of the sending node
msg->len                  // the number of data bytes in the frame's payload
msg->data[0]              // the opcode of the received message
msg->data[1] and [2]      // the Node Number (NN) of the sending module
msg->data[3] and [4]      // the Event Number (EN) of the received message
```

Bytes 5-7 are additional data bytes that are send by some opcodes. See the CBUS Developers' Guide for more information on opcodes.

Producer Modules

A Producer module will need to send CBUS messages when something of interest happens, e.g. a switch is pressed, a loco is detected, etc. It is up to you to define the following items for any CBUS messages you wish to send:

- the opcode, depending on whether it is a simple on/off event (e.g. ACON/ACOF) or something more complex with additional data bytes (e.g. ACON3/ACOF3)

- the Event Number (EN)

This example shows how to send a simple ON event message with event number 1 and no additional data bytes:

```
// create and initialise a message object
CANFrame msg;
memset(&msg, 0, sizeof(msg));

// populate the object's parameters
// the CANID
msg.id = config.CANID;

// the size of the data payload
msg.len = 5;

// the opcode
msg.data[0] = OPC_ACON;

// the module's node number (NN)
msg.data[1] = highByte(config.nodeNum);
msg.data[2] = lowByte(config.nodeNum);

// the event number (EN)
msg.data[3] = 0;
msg.data[4] = 1;

// send the message

bool sent_ok = CBUS.sendMessage(&msg);
```

This function returns `true` if the message was successfully sent or `false` if not.

Note that the NN and EN are both 16-bit integers and each occupy two bytes of the message. The Arduino macros highByte() and lowByte() return the appropriate (8-bit) byte from a 16-bit integer. See the CBUS Developers' Guide for a full discussion of short and long events and the meaning of NN and EN.

Module Reset

The Arduino environment provides no easy way to program the microcontroller's on-chip EEPROM. To ensure that the contents of the EEPROM are cleared and set to sensible defaults, the CBUSconfig library provides a simple method for resetting the module. This is shown in the example program included with the library.

Hold down the pushbutton switch as you power-on the module. Then, as a safety precaution, press and hold the switch for a further 5 seconds. The module will then reset the EEPROM contents and reboot. The Node Number and CANID will both be set to zero.

The module can only be reset whilst in SLiM mode (with the green LED illuminated). If, due to random EEPROM data, the module starts up in FLiM mode (with the yellow LED illuminated) or you want to reset the module at any time in the future, hold the switch down for 6 seconds to revert to SLiM mode. You can then proceed to reset the module.

CBUS SLiM Mode

The library does not currently support SLiM mode; that is, there is no provision for setting the node number or learning events by hardware switches. Therefore, you must use FCU or JMRI to configure your module in FLiM mode. This is in common with most newer MERG CBUS modules.

Support for SLiM configuration and event learning may be added as a future enhancement if demand exists.

Arduino Serial Port

The library prints every received event to the module's serial port (115200 baud, 8N1). This is useful for testing the bus connection or simply for monitoring the CBUS traffic.

The library also prints copious debug information but by default this code is commented out. Selected lines can be uncommented to help with code debugging and development.