

Falcon BMS to Arduino Interface Tool (BMSAIT)



Autor	Robin „Hummer“ Bruns
Dokumentversion	1.4
Softwareversion	1.8.4
BMS Version	4.37
Datum	17.01.2023

Inhalt

1.	Einleitung.....	4
1.1.	Grundsätzliches	4
1.2.	Quickstart	5
2.	Grundlogiken	6
2.1.	Datenaustausch zwischen PC und dem Arduino Board	6
2.1.1.	Hardwareverbindung.....	6
2.1.2.	Softwareverbindung.....	6
2.1.3.	PUSH-Prinzip.....	7
2.1.4.	PULL-Prinzip.....	9
2.2.	Synchronisierung	12
2.2.1.	Motorenkalibrierung	12
2.2.2.	Motoren in Nullstellung bringen	13
2.2.3.	Synchronisierung der Schalterstellungen	13
3.	Windows Anwendung	14
3.1.	Erste Schritte	14
3.2.	BMSAIT Windows Software.....	15
3.2.1.	Das Hauptformular	15
3.2.2.	Basiseinstellungen	21
3.2.3.	Geräte einrichten	23
3.2.4.	Formular Variablenauswahl	24
3.2.5.	Formular Eingabekommandos	25
3.3.	(optional) Virtuelle Joysticks aktivieren	28
3.4.	(work in progress) DCS Integration	29
4.	Das Arduino Programm	31
4.1.	Beschreibung der Vorbereitungen	31
4.1.1.	Vorwort.....	31
4.1.2.	Das Programmieren eines Arduino mit dem BMSAIT Sketch.....	31
4.1.3.	Herunterladen des Programmcodes	31
4.1.4.	Installation einer Entwicklungsumgebung	31
4.1.5.	Auswahl des Arduino Boards.....	33
4.1.6.	Auswahl des COM-Port.....	33
4.1.7.	Prüfen der Arduino-Software	33
4.1.8.	Hochladen der Arduino-Software	33
4.2.	Beschreibung des Arduino Sketches	34

4.2.1.	Das Modul BMSAIT_Vanilla	34
4.2.2.	Das Modul UserConfig.....	35
4.2.3.	Das Modul Switches	39
4.2.4.	Das Modul ButtonMatrix.....	42
4.2.5.	Das Modul Encoder	43
4.2.6.	Das Modul Analogachse	44
4.2.7.	Das Modul LED.....	45
4.2.8.	Das Modul LED-Matrix.....	46
4.2.9.	Das Modul LCD	48
4.2.10.	Das Modul SSegMAX7219	48
4.2.11.	Das Modul SSegTM1367.....	49
4.2.12.	Das Modul Servo.....	50
4.2.13.	Das Modul ServoPWMSHield.....	51
4.2.14.	Das Modul Stepper	53
4.2.15.	Das Modul StepperX27	53
4.2.16.	Das Modul StepperVID	54
4.2.17.	Das Modul MotorPoti.....	56
4.2.18.	Das Modul OLED	57
4.2.19.	Das Modul Speedbrake Indicator	59
4.2.20.	Das Modul FFI.....	61
4.2.21.	Das Modul DED/PFL.....	62
5.	Anlagen.....	64
5.1.	Quellen / Verweise	64
5.2.	Datenfeldbeschreibungen	65
5.2.1.	Datenvariablen (BMSAIT-Variablen.csv, Windows-App).....	65
5.2.2.	Variablenzuordnung (Windows-App).....	66
5.2.3.	Kommandos (Windows-App)	66
5.2.4.	Globalvariablen (Arduino)	67
5.2.5.	Schriftart DED/PFL.....	69

1. Einleitung

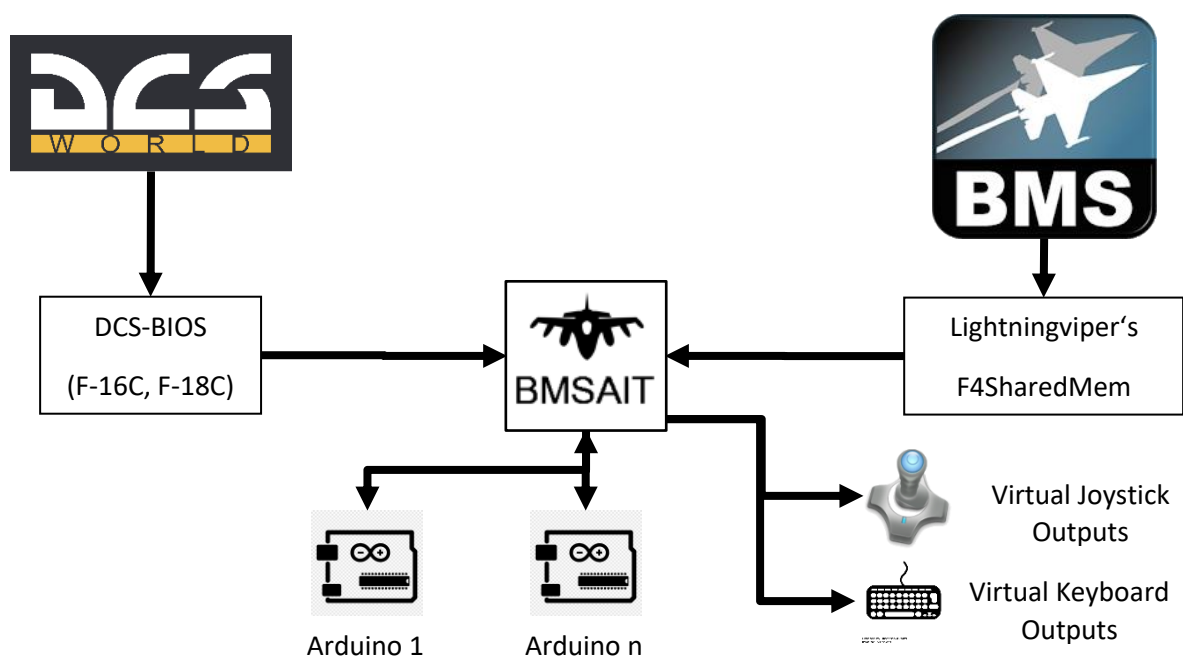
1.1.Grundsätzliches

Das Falcon BMS to Arduino Interface Tool (BMSAIT) ist ein Hilfsmittel, um einerseits Fluginformationen aus Falcon BMS zu extrahieren und zur Ansteuerung von Geräten beim Cockpitbau zur Verfügung zu stellen sowie andererseits zur Verarbeitung und Versendung von Steuerungssignalen an die Flugsimulation. Das Tool unterteilt sich dabei in zwei Bausteine:

1. Das Windows-Programm
 - a. extrahiert Daten aus der SharedMem von Falcon BMS und gibt Daten über eine serielle Schnittstelle aus.
 - b. Nimmt Steuerungsinformationen eines Arduino entgegen und setzt diese in Tastatursignale oder Joystickbefehle um.
2. Das Arduino-Sketch konfiguriert ein Arduino-Board, um
 - a. Daten über die serielle Schnittstelle zu empfangen und über ausgewählte Geräte (LEDs, LCDs, Servos, weitere) auszugeben.
 - b. Verschiedene Eingabegeräte auszulesen und Steuerungssignale an das Windows-Programm zurück zu senden.

Für die Extraktion von Daten der BMS SharedMem wurden in der Vergangenheit bereits viele Tools bereitgestellt (nur um einige zu nennen: FAST, BMS Cockpit, DEDuino, F4toSerial).

In vielen Bereichen kann BMSAIT mit diesen Programmen nicht konkurrieren und soll es auch nicht. BMSAIT wurde aus der Not heraus entwickelt, dass viele der genannten Tools entweder nicht mehr weiterentwickelt werden (und damit nicht alle Daten bereitstehen, die in der SharedMem von Falcon BMS mit den jüngsten Updates verfügbar sind) oder einen begrenzten Funktionsumfang besitzen und damit nicht alle Cockpitprojekte unterstützen können. BMSAIT ist im aktuellen Entwicklungsstadium daher als Ergänzung für Anwendungsfälle zu sehen, die durch die gängigen Programme nicht abgedeckt sind.



Die Nutzung von BMSAIT war zuerst als reines Ausgabeprogramm gedacht; die Erzeugung von Steuersignalen über Eingabegeräte war nur eine Randfunktion. Mit der Weiterentwicklung der Eingabefunktionen hat sich aber ein großes Potential ergeben, dass ein Arduino mit BMSAIT Software nun in der Lage ist, beim Cockpitbau theoretisch alle Funktionen eines Panels abzubilden. Eine Verkabelung von Cockpitpanels zu zentralen Controllern können zu komplexen und ggf. auch fehleranfälligen Strukturen führen (ich weiß, wovon ich hier rede!) BMSAIT ermöglicht es, dass ein oder mehrere Panel mit nur einem, direkt hinter dem Panel befindlichen Arduino, verbunden werden, der sämtliche In- und Outputs des Panels abdeckt. Nach außen hin benötigt das Panel damit nur ein einziges USB Kabel (und ggf. eine externe Spannungsversorgung). Dies hilft den Cockpitbau übersichtlich zu gestalten.

Grundsätzlich ist vorgesehen, dass das Windows-Tool in Kombination mit dem Arduino-Sketch genutzt wird. Es ist aber theoretisch möglich, beide Tools unabhängig voneinander im Zusammenspiel mit anderer Software zu nutzen.

Wenn ihr BMSAIT nutzen wollt, werdet ihr euch mit hardwaretechnischen Fragen und ein wenig mit der Programmierung von Arduinos mittels C++ beschäftigen müssen. Mit dieser Dokumentation hoffe ich alle notwendigen Erklärungen bereitzustellen, um euch in die Lage zu versetzen auch ohne Programmierkenntnisse in die Lage zu versetzen die Software einzurichten. Im Zweifel spricht mich gerne an - im Rahmen freier zeitlicher Kapazitäten werde ich euch bei der Einrichtung oder euren Ideen zu Erweiterungen gerne unterstützen.

BMSAIT kann theoretisch mit allen gängigen Arduino-Boards verwendet werden. Ich habe aber Probleme in der Kommunikation mit dem Leonardo/Pro Micro festgestellt, so dass ich die Verwendung des Uno, Micro, Nano oder Mega empfehle. Eine Unterstützung für den Due ist seit Version 1.2 vorhanden. Die Besonderheiten dieses Typs können aber zu Problemen führen.

Für manche Anwendungsfälle ist es vorteilhaft, wenn der Microcontroller über mehr Leistung und mehr Speicher verfügt (z.B. zur Ansteuerung des DED). Hierfür ist es möglich, BMSAIT auch auf Controllern wie dem ESP32 zu nutzen.

1.2.Quickstart

Lest zu Anfang die Kapitel 3.1 (Erste Schritte Windows-Anwendung) und Kapitel 4.1 (Vorbereitung Arduino) um die Voraussetzungen für die Nutzung von BMSAIT herzustellen.

Für die ersten Versuche mit BMSAIT empfehle ich die verfügbaren [Beispiele](#) auszuprobieren. Die Beispielprogramme umfassen einen vorgefertigten Sketch für den Arduino (*.ino und die dazugehörigen .h Dateien), eine Konfigurationsdatei (*.ini) für die Windows-Anwendung und eine gesonderte Dokumentation mit genauer Beschreibung der erforderlichen Verkabelung.

Wenn die Funktionsfähigkeit der Software dadurch auf euren Rechnern nachwiesen ist und Ihr die Grundzüge der Software dadurch verstanden habt, wird es euch leichter fallen, eure eigenen Projekte umzusetzen.

2. Grundlogiken

2.1. Datenaustausch zwischen PC und dem Arduino Board

2.1.1. Hardwareverbindung

Der Datenaustausch zwischen Windows und Arduino erfolgt über eine serielle Datenverbindung. Diese wird üblicherweise über die USB-Verbindung des Arduino mit dem Windows-Rechner erreicht.

Das Arduino-Board bekommt dabei einen COM-Port zugewiesen. Solange das Board immer an dem gleichen USB-Port angeschlossen wird, wird dem Board die gleiche COM-Port Nummer zugeordnet. Sollten Ihr das Arduino-Board an einem anderen USB-Anschluss anschließen, muss beachtet werden, dass das Board auf einem anderen COM-Port zu erreichen ist.

Das Arduino-Board wird über den USB-Port über den PC mit Spannung versorgt. Sollte der Arduino genutzt werden, um Geräte mit hohem Strombedarf anzusteuern (z.B. Servomotoren) sollte zudem eine zusätzliche Spannungsversorgung genutzt werden, um Schäden am Arduino-Board zu vermeiden.

Hinweis: Sollten Ihr zusätzliche Spannungsquellen für eines an dem Arduino-Board angeschlossenen Geräte verwenden, dann achtet bitte darauf, dass die Masse-Anschlüsse von Arduino und Spannungsquelle verbunden werden!

2.1.2. Softwareverbindung

Ergänzend zur Hardware-Verbindung sind auch Softwareeinstellungen Voraussetzung für eine Kommunikation zwischen Windows und Arduino. Erster Schritt hierbei ist die Wahl der BAUD-Rate (die Übertragungsgeschwindigkeit). Die BAUD-Rate wird für den Arduino in dem darauf laufenden Programm vorgegeben. Möchte man die BAUD-Rate verändern, muss das Arduino Sketch angepasst und neu hochgeladen werden (siehe 4.2.2).

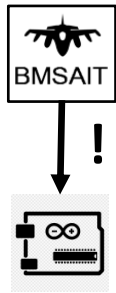
Im Windows-Tool kann die BAUD-Rate vor Verbindungsaufbau bei den Device-Einstellungen (siehe 3.2.3) festgelegt werden. Eine Kommunikation ist nur möglich, wenn beide BAUD-Raten übereinstimmen.

BMSAIT unterstützt zwei verschiedene Kommunikationsformen für den Informationsaustausch mit entsprechend programmierten Arduino-Boards:

- Das PUSH-Prinzip
- Das PULL-Prinzip

Seit Version 1.3 ist BMSAIT nicht auf eines der Verfahren festgelegt, sondern es ist möglich die Verarbeitungslogik für jeden Arduino individuell festzulegen.

2.1.3. PUSH-Prinzip



Beim PUSH-Prinzip liegt die Federführung für den Datenaustausch bei der Windows-App. Hier erfolgt daher die Auswahl der relevanten Informationen der SharedMem aus Falcon BMS. Die App verfügt dafür über einen Editor, mit dem die gewünschten Daten ausgewählt und konfiguriert werden können (siehe 3.2.4).

Das Arduino-Board nimmt lediglich Daten entgegen und bringt diese zur Anzeige. Der Vorteil liegt hier darin, dass das Arduino-Board um etwas Rechenleistung entlastet wird und das Testen der Funktionen erleichtert ist.

2.1.3.1. Ablauflogik der Windows-Software beim PUSH

Verarbeitungsbeginn:

Mit einem Klick auf die Schaltfläche „Verbindung starten“ wird die Verarbeitung der folgenden Schritte begonnen.

Wurde in den Basiseinstellungen der „Autostart“ aktiviert, wird die Verarbeitung sofort beim Starten des Windows-Programms BMSAIT.exe begonnen.

Initialisierung:

Die Initialisierung erfolgt einmalig beim Aufbau der Verbindung

- Schnellzugriff auf selektierte SharedMem-Variablen herstellen
- COMPort zum Arduino öffnen

Schleife:

Die Schleife wird endlos durchlaufen, bis der Nutzer die Verbindung beendet. Die Häufigkeit des Schleifendurchlaufs kann durch den Wert *polltime* in den Basiseinstellungen geregelt werden (siehe 4.2.2).

- Aktuelles Abbild der SharedMem aus dem Falcon BMS laden
- Daten aus den Datenmonitor ausgeben
- Für jede in der Konfiguration definierte Datenvariable wird folgendes Verfahren durchlaufen:
 - Anhand des Datentyps und der ID der Datenvariable wird der gewünschte Wert aus dem Abbild der SharedMem gelesen.
 - Der gelesene Wert wird in ASCII-Code umgewandelt.
 - Ein Datenpaket für die Übertragung wird aufbereitet (Ergänzung der Datenvariable um Steuerungsinformationen).
 - Das Datenpaket über den seriellen Port gesendet.
- Horchen, ob auf einem COM-Port eine Antwort eines Arduino vorliegt.
 - Nachrichten werden in der Textkonsole des Hauptformulars ausgegeben
 - Anweisungen zu Tastaturkommandos werden verarbeitet und Tastensignale ausgelöst.

Verarbeitungsende:

Mit einem Klick auf die Schaltfläche „Verbindung beenden“ wird die Verarbeitung beendet.

Das Schließen des Programms BMSAIT beendet die Verarbeitung naheliegenderweise ebenfalls.

2.1.3.2. Ablauflogik der Arduino-Software beim PUSH

Verarbeitungsbeginn:

Die Verarbeitung beginnt, sobald der Arduino eine Spannungsversorgung erhält und ein gültiges Programm geladen ist.

Initialisieren (erfolgt bei jedem neuen Verbindungsaufbau):

- Allgemeine Variablen vorbelegen
- Software für die Ansteuerung der Peripherie starten (Einbinden von Bibliotheken und belegen von Variablen)
- COM-Port öffnen

Schleife:

Die Schleife wird vom Arduino bis zum Verarbeitungsende durchlaufen.

- Stellung angeschlossener Schalter/Analogachsen überprüfen.
Bei Änderung: Senden von Kommandos an den PC.
- Horchen, ob Systemkommandos vom PC vorliegen.
Wenn ja, werden diese Kommandos verarbeitet.
- Horchen, ob Simulationsdaten vom PC vorliegen.
Wenn Daten vorliegen: Daten einlesen und zwischenspeichern
- Anzeigen auf angeschlossenen Geräten aktualisieren.

Verarbeitungsende:

Eine Verarbeitung kann nur beendet werden, indem der Arduino vom Stromnetz getrennt wird oder das bestehende Programm durch ein anderes Programm überschrieben wird.

2.1.3.3. Datensatzformat einer Übertragung vom PC an den Arduino beim PUSH-Prinzip

<START-Byte><INFO-Byte><Formatstring1><Daten><Formatstring2>

START-Byte:

Das Byte 255 signalisiert den Beginn einer Übertragung (siehe 4.2.2)

INFO-Byte:

Mit dem Infobyte wird dem Arduino angezeigt, um was für eine Art von Nachricht es sich handelt:

Handshake:

Wenn hier ein definiertes Byte übertragen wird (siehe 4.2.2 „HANDSHAKE“), ist das ein Zeichen für den Arduino, dass die Windows-App gerade nach angeschlossenen Arduino-Boards sucht. Der Arduino wird mit einem definierten Antwortsignal (ID des Boards, siehe 4.2.2.2) antworten, um der Windows-App anzuzeigen, dass die Nachricht erhalten wurde und das Board bereit ist.

Steuerung PULL:

Siehe 2.1.4.4.

Datenübertragung:

Bei einer Datenübertragung wird mit dem INFO-Byte die Positionsnummer angegeben, in der der Wert der aktuellen Übertragung im vorbereiteten Datencontainer des Arduino zwischengespeichert werden soll.

Formatstring1:

In der Windows-App können bei der Variablenzuordnung Formatierungsangaben mitgegeben werden. In Formatstring1 sind die Zeichen enthalten, die vor dem eigentlichen Datenwert stehen

sollen. Standardmäßig erwartet das BMSAIT-Arduino-Sketch hier das Zeichen ‚<‘ (siehe 4.2.2 „VAR_BEGIN“).

Daten:

Hierbei handelt es sich um die aus der SharedMem ausgelesenen Informationen. Dies kann ein einzelnes Zeichen sein (z.B. ein Wahr/Falsch Wert) oder ein String mit vielen Zeichen (z.B. eine Zeile des DED). Die maximale Länge für einen Datensatz kann im Arduino-Sketch festgelegt werden. Der Wert sollte sich zwischen 1 (ausschließlich Verarbeitung von LED mit Wahr/Falsch-Wert) und 25 (Länge einer Zeile im DED oder PFD) bewegen.

Formatstring2:

In der Windows-App können bei der Variablenzuordnung Formatierungsangaben mitgegeben werden. In Formatstring2 sind die Zeichen enthalten, die nach dem eigentlichen Datenwert stehen sollen. Der mitgelieferte Arduino-Sketch erwartet hier das Zeichen ‚>‘. (siehe 4.2.2 „VAR_ENDE“).

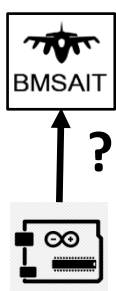
2.1.3.4. Datensatzformat einer Übertragung vom Arduino zum PC beim PUSH-Prinzip

Eine Übertragung besteht aus der Angabe des Übertragungstyps und dem eigentlichen Datensatz:

<Übertragungstyp><Datensatz>

Übertragungstyp	Inhalt Datensatz
t	Der übertragene Datensatz soll in der Konsole von BMSAIT angezeigt werden
k	Es wird der aktuelle Status eines Schalters übertragen (1- aktiv, 0-nicht aktiv)
a	Der übertragene Datensatz enthält die Positionsangabe einer Analogachse: <Befehl-ID>,<AchsenPosition> Die Befehl-ID ist eine dreistellige Zahl, die auf einen Eintrag in der Kommandoverwaltung der Windows-App referenziert. Wenn dem Arduino in der Windows-App ein Kommando mit dieser ID zugeordnet und mit einer Analogachse eines vjoy verbunden wird, wird die vJoy Analogachse auf die vom Arduino übertragene Position (0..1024) festgelegt.
g	Der Arduino meldet, dass der Eingangspuffer leer ist und daher neue Daten empfangen werden können.
s	Der Arduino sendet ein internes Kommando an die BMSAIT App

2.1.4. PULL-Prinzip



Beim PULL-Prinzip liegt die Federführung für die Datenübertragung bei der Arduino-Software. Die Windows Software wartet auf Anfragen des Arduino und sendet nur dann Daten, wenn der Arduino diese explizit anfordert. Der Vorteil liegt hier darin, dass man die Konfiguration der gewünschten Datenvariablen im Gegensatz zum PUSH nur einmal (nämlich nur im Arduino) einrichten muss. PULL erleichtert zudem die zeitliche Synchronität des Datenaustauschs, so dass zeitkritische Daten schneller auf den Arduino gelangen. Alle benötigten Datenvariablen müssen – wie beim PUSH auch – in dem Arduino-Sketch (BMSAIT-UserKonfig.h) eingegeben werden.

2.1.4.1. Ablauflogik der Windows-Software beim PULL

Die PULL-Verarbeitung kann in der Windows-App bei den Einstellungen eines Arduino-Gerätes ausgewählt werden.

Verarbeitungsbeginn:

Mit einem Klick auf die Schaltfläche „Verbindung starten“ in der Windows-App wird bei den Arduinos, die auf die PULL-Verarbeitung eingestellt wurden, die Verarbeitung gestartet:

Initialisierung:

Die Initialisierung erfolgt einmalig beim Aufbau der Verbindung.

- Die Windows-App sendet dazu ein Codesignal an den Arduino, um bei diesem die PULL Logik zu aktivieren.
- Es wird eine Verbindung zur SharedMem von BMS oder DCS aufgebaut

Schleife:

- Aktuelle Daten aus der SharedMem lesen
- Die Daten der SharedMem auf dem Datenmonitor ausgeben
- Horchen, ob Anfragen oder Kommandos des Arduino vorliegen
 - Ausgeben von Nachrichten in der Textkonsole des Hauptformulars.
 - Verarbeiten von Datenanfragen:
 - Lesen des entsprechenden Wertes gemäß Datenanfrage aus dem Abbild.
 - Umwandeln des gelesenen Werts in ASCII-Code.
 - Aufbereiten des Datenpaketes für die Übertragung (Ergänzung der Datenvariablen um Steuerungsinformationen).
 - Senden des Datenpaketes über den seriellen Port.
 - Auslösen von Tastatur-/Joysticksignalen
 - Bewegen von Analogachsen

Verarbeitungsende:

Mit einem Klick auf die Schaltfläche „Verbindung beenden“ wird die Verarbeitung beendet. Dazu wird ein Signal an den Arduino übertragen, das diesen veranlasst das Senden von PULL-Anfragen zu beenden.

2.1.4.2. Ablauflogik der Arduino-Software beim PULL

Verarbeitungsbeginn:

Die Verarbeitung beginnt, sobald der Arduino eine Spannungsversorgung erhält und ein gültiges Programm geladen ist.

Initialisierung:

- Variablen vorbelegen
- Software für die Ansteuerung der Peripherie starten
- COM-Port öffnen

Schleife:

Die Schleife wird vom Arduino bis zum Verarbeitungsende durchlaufen.

- Horchen, ob Kommandos vom PC vorliegen.
 - Wenn ja, werden die Kommandos verarbeitet.

- Wird das Kommando zur Aktivierung des PULL-Prinzips empfangen, wird die folgende PULL-Verarbeitung verarbeitet
- Wird das Kommando zur Deaktivierung des PULL-Prinzips empfangen, wird die folgende PULL-Verarbeitung nicht mehr verarbeitet
- PULL-Verarbeitung (nur wenn aktiviert)
 - Durchlaufen aller im Datencontainer eingetragenen Datenvariablen
 - Aufbereitung eines Kommandos zur Datenanfrage für die Datenvariable
 - Senden der Datenanfrage
- Horchen, ob serielle Daten vom PC vorliegen.
 - Wenn Daten vorliegen: Daten einlesen und zwischenspeichern
- Zwischengespeicherte Daten über die angeschlossene Peripherie ausgeben
- Stellung angeschlossener Schalter überprüfen. Bei Änderung werden Kommandos an den PC gesendet.

Verarbeitungsende:

Eine Verarbeitung kann nur beendet werden, indem der Arduino vom Stromnetz getrennt wird.

2.1.4.3. Datensatzformat einer Übertragung vom Arduino zum PC beim PULL-Prinzip

Ergänzend zu dem Übertragungsformat gem. PUSH (siehe 2.1.3.4) kommen hier die Übertragungstypen „d“ und „u“ zum Einsatz, um Datenanfragen an den PC zu senden:

d	<p>Der übertragene Datensatz enthält eine Datenanfrage im Format: <Datenposition>,<Variablentyp>,<VariablenID></p> <p>Datenposition Dies ist die Position im Datencontainer des Arduino-Sketch</p> <p>Variablentyp Hier wird mit einem Byte der Variablentyp der Datenvariablen angezeigt, z.B. {f} für eine float-Zahl. Der Wert wird aus dem Datencontainer gelesen (Datenfeld.format). Wenn im Datencontainer ein falsches Format angegeben ist, wird die Anfrage durch BMSAIT nicht bearbeitet werden können. Siehe auch die Beschreibung der <i>Datenvariable</i> in der Anlage</p> <p>VariablenID Hier wird aus dem Datencontainer die ID der Datenvariablen übernommen (Datenfeld.id). Für die weitere Verarbeitung ist wichtig, dass es sich um einen Text mit genau vier Zeichen handelt. In der Befüllung des Datencontainers ist daher darauf zu achten, dass führende Nullen mitgegeben werden!</p>
u	<p>Mit diesem Kommando wird die BMSAIT App um Übertragung der aktuellen Daten des DED („uDED“) oder PFL („uPFL“) gebeten. Die WinApp wird daraufhin die fünf Zeilen des DED/PFL aufbereiten und nacheinander an den anfordernden Arduino senden.</p>

2.1.4.4. Datensatzformat einer Übertragung vom PC zum Arduino beim PULL-Prinzip

Es gilt das Format gemäß Kapitel 2.1.3.3. mit der Ergänzung, dass zwei Info-Bytes definiert sind, die die PULL-Verarbeitung des Arduino steuern:

PULL-Verarbeitung starten:

Wenn als Info-Byte der Wert 0xAA (170) übertragen wird, ist das ein Zeichen für den Arduino, dass die PULL-Verarbeitung gestartet werden soll (siehe 5.2.4 „STARTPULL“).

PULL-Verarbeitung beenden:

Wenn als Info-Byte der Wert 0xB4 (180) übertragen wird, ist das ein Zeichen für den Arduino, dass die PULL-Verarbeitung beendet werden soll (siehe 5.2.4 „ENDPULL“).

2.2.Synchronisierung

2.2.1. Motorenkalibrierung

Bei Steppermotoren kann die Software in der Regel die aktuelle Position der Anzeige nicht speichern. Beim Neustart des Arduino ist der Software daher nicht bekannt, in welcher Position ein Zeiger vor dem Neustart stand.

Aus diesem Grund müssen Steppermotoren vor der Nutzung in der Simulation kalibriert werden, um die Anzeigen in eine Stellung zu bringen, in der die softwareseitige und die hardwareseitige Position übereinstimmen.

Eine Motorenkalibrierung sieht vor, dass ein Motor in die Nullstellung gefahren wird (voller Ausschlag gegen den Uhrzeigersinn), dann das voll Bewegungsspektrum im Uhrzeigersinn vollführt und anschließend wieder in die Nullstellung zurückfährt. Damit sollte erreicht werden, dass Motoren immer in der jeweiligen Nullstellung stehen.

Die Kalibrierung kann auf drei Weisen erfolgen:

Auf dem Hauptformular der BMSAIT Windows App kann die Schaltfläche „Motoren kalibrieren“ betätigt werden. Dies sendet ein Signal an alle eingerichteten Arduinos und weist diese an die angeschlossenen Motoren zu kalibrieren.

Über das Auswahlménü der eingerichteten Devices kann eine spezifische Kalibrierung eines bestimmten Arduino angewiesen werden.

Als letzte Möglichkeit kann ein Button an einem mit BMSAIT konfigurierten Arduino verwendet werden, um die Kalibrierung über ein internes Kommando an die BMSAIT WinApp zu senden (Internes Kommando 255).

2.2.2. Motoren in Nullstellung bringen

Eine vollständige Motorenkalibrierung ist in manchen Fällen nicht angebracht, da diese zu lange dauert. Daher gibt es als weitere Möglichkeit eine „kleine“ Kalibrierung durchzuführen, bei der die Motoren lediglich in die Nullstellung gefahren werden.

Dies kann auf die folgenden Weisen erreicht werden:

1. Man betätigt die Schaltfläche „Motoren Kalibrieren“ auf dem Hauptformular der BMSAIT Windows App,
2. Zudem ist eine automatisierte Kalibrierung vorgesehen. Diese wird beim Ausstieg aus der 3D Welt von BMS bzw. beim Aufruf der 2D Welt von DCS angestoßen,
3. Als letzte Möglichkeit kann ein an einem Arduino angeschlossener Button verwendet werden, um die Aufforderung einer Synchronisierung über ein internes Kommando an die BMSAIT WinApp zu senden (Internes Kommando 254).

2.2.3. Synchronisierung der Schalterstellungen

Beim Starten der Simulation kann es sein, dass die Schalterstellungen in der Simulation nicht mit denen im Home-Cockpit übereinstimmen. Erst wenn ein physikalischer Schalter betätigt wird, wird die entsprechende Stellung in der Simulation gesetzt und eine Synchronisation ist hergestellt.

Um eine schnelle Synchronisation zu erreichen, besteht die Möglichkeit den Arduinos mit BMSAIT einen Befehl zu geben, die Position aller Schalter durchzugeben. Dies kann benutzt werden, um damit die Stellung der physikalischen Schalter auf die Simulation zu übertragen.

Hierzu gibt es drei Möglichkeiten (die aus naheliegenden Gründen aber nur dann Sinn ergeben, wenn man sich in der 3D Welt der Simulation befindet):

1. Man betätigt die Schaltfläche „Motoren Kalibrieren“ auf dem Hauptformular der BMSAIT Windows App,
2. Die Synchronisierung wird automatisch beim Eintritt in die 3D Welt von BMS ausgelöst,
3. Als letzte Möglichkeit kann ein an einem Arduino angeschlossener Button verwendet werden, um die Aufforderung einer Synchronisierung über ein internes Kommando an die BMSAIT WinApp zu senden (Internes Kommando 253).

3. Windows Anwendung

Die Windows-Anwendung wurde mit dem Windows Visual Studio 2019 als Windows-App in C# für x64-Prozessoren geschrieben. Dabei wird auf .net-Komponenten der Version 4.7.2 zugegriffen.

Für den Betrieb werden folgende Bibliotheken verwendet:

„F4SharedMem.dll“ von LightningViper zum Zugriff auf die SharedMem des Falcon BMS

„WindowsInputLib.dll“ (Input Simulator Plus) von Michael Noonan, Theodoros Chatzigiannakis für die Erzeugung von Tastatursignalen

„vJoyInterface.dll“ und „vJoyInterfaceWrap.dll“ für die Erzeugung von Joystickbefehlen

Die Projektdateien mit meinem Sourcecode stelle ich auf Anfrage gerne zur Verfügung.

3.1. Erste Schritte

Zum Starten des Programms sind die folgenden Dateien erforderlich, die gemeinsam in einem Ordner des Windows-Systems abgelegt werden müssen:

- BMSAIT.exe
- F4SharedMem.dll
- WindowsInput.dll
- WindowsInputLib.dll
- vJoyInterface.dll
- vJoyInterfaceWrap.dll
- BMSAITVariablen.csv
- BMSAIT-GaugeTable.csv
- Unterordner „de“ mit der Datei BMSAIT.resources.dll für deutsche Spracheinstellungen
- Unterordner „en“ mit der Datei BMSAIT.resources.dll für englische Spracheinstellungen

Beim ersten Start des Programms wird es zu einer Fehlermeldung kommen, da zuerst die Basiseinstellungen festgelegt werden müssen.

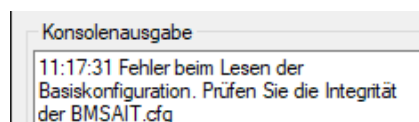


Abbildung 1: Fehlermeldung bei fehlender Basiskonfiguration

Die Aktualisierung der Basiseinstellungen wird in Kapitel 3.2.2 beschrieben.

3.2.BMSAIT Windows Software

3.2.1. Das Hauptformular

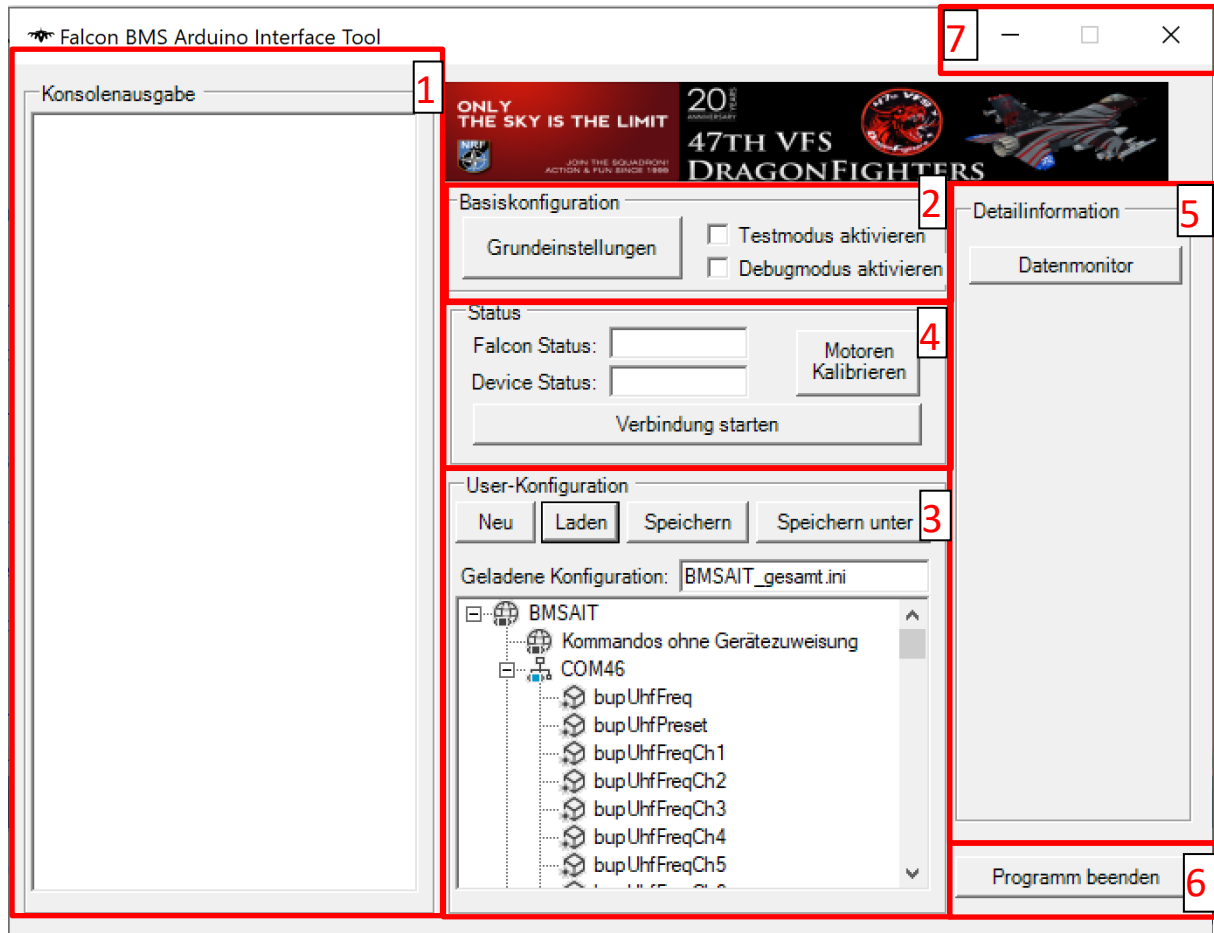
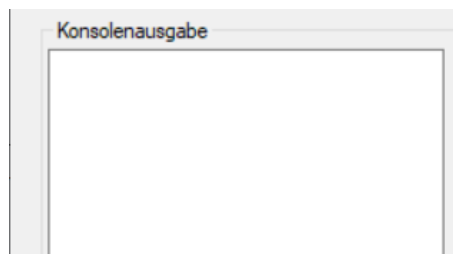


Abbildung 2: BMSAIT Hauptformular

Beim Start des Programms wird das Hauptformular aufgerufen. Das Hauptformular dient zur Information über die aktuellen Einstellungen und ermöglicht die Steuerung der Anwendung.

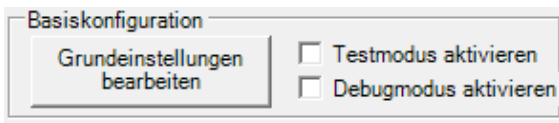
Das Hauptformular ist in Bereiche unterteilt.

Hauptformular Bereich 1 – Konsolenausgabe



In diesem Feld werden Statusmittlungen und Fehlermeldungen angezeigt.

Hauptformular Bereich 2 – Basiskonfiguration

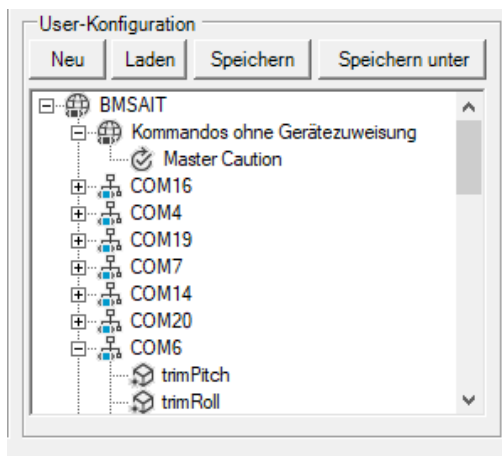


Mit der Schaltfläche „Grundeinstellungen“ wird ein Formular geöffnet, in dem die Basiseinstellungen des Programms angepasst werden können.

Das Flag Testmodus die Möglichkeit, eine Datenverbindung des Windows-Programms mit angeschlossenen Arduinos zu testen ohne Falcon BMS dafür starten zu müssen. Wenn dieser Flag gesetzt ist und die Verarbeitung gestartet wird, wird keine Verbindung mit der SharedMem des Falcon BMS aufgebaut. Stattdessen werden Testdaten verwendet, die in der Konfiguration bei der Definition einer Datenvariablen eingegeben wurden. Im Testmodus werden zudem die von den Arduinos gemeldeten Eingabekommandos ausgelöst, auch wenn Falcon BMS und DCS nicht gestartet sind.

Im Debugmodus werden Status- und Fehlermeldungen des Arduino für Analysen in der Konsole angezeigt.

Hauptformular Bereich 3 – User-Konfiguration



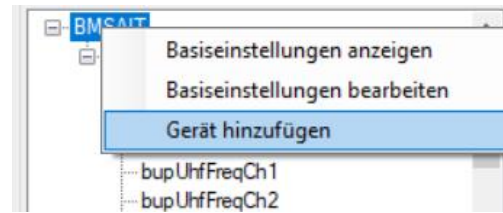
In diesem Bereich werden Werkzeuge zur Erstellung einer Konfiguration von Geräten (Devices), Datenvariablen der SharedMem (Variablen) und Eingabekommandos angezeigt.

In dem Fenster werden drei Ebenen angezeigt. Ebene 1 ist der TOP-Knoten. Unterhalb des TOP-Knotens werden Devices (COM-Ports, die einen Arduino repräsentieren) angelegt. Auf der untersten Ebene werden unterhalb der Devices die diesen zugeordneten Variablen und Eingabekommandos angezeigt.

Der Knoten „Kommandos ohne Gerätezuweisung“ zeigt Eingabekommandos an, die keinem Arduino speziell zugeordnet worden. Da Eingabekommandos erst mit Version 1.3 direkt einem Arduino zugeordnet sind, werden beim Laden alter Konfigurationen die Eingabekommandos unter diesem Knoten angezeigt. Für neue Konfigurationen ist dieser Knoten ohne Belang.

Ein linker Mausklick auf einen Eintrag im Fenster der User-Konfiguration wählt diesen aus. In Bereich 5 des Hauptformulars werden Detailinformationen zum gewählten Objekt angezeigt.

Über einen rechten Mausklick auf den **TOP-Knoten** stehen die folgenden Optionen zur Auswahl:



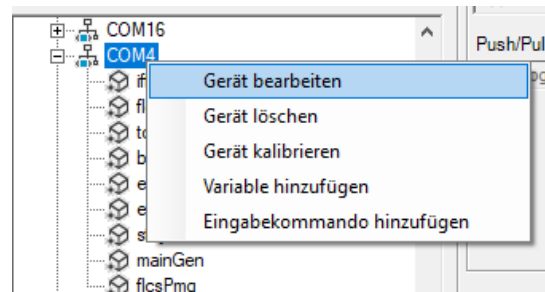
Basiseinstellungen anzeigen/bearbeiten

Es wird ein Formular geöffnet, in dem die Basiseinstellungen des Programms angepasst werden können.

Gerät hinzufügen

Es wird ein Fenster zum Hinzufügen eines Arduinos geöffnet.

Über einen rechten Mausklick auf ein **Gerät** stehen die folgenden Optionen zur Auswahl:



Gerät bearbeiten

Es wird ein Fenster geöffnet, in dem die aktuellen Einstellungen des gewählten Gerätes angezeigt und angepasst werden können.

Gerät löschen

Das gewählte Gerät inkl. aller Variablenzuordnungen wird gelöscht.

Gerät kalibrieren

Am gewählten Gerät werden die Motoren neu kalibriert.

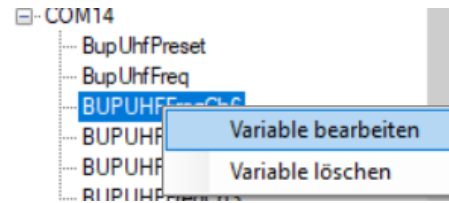
Variable hinzufügen

Es wird ein Fenster zum Hinzufügen einer Variablen geöffnet

Eingabekommando hinzufügen

Es wird ein Fenster zum Hinzufügen von Input-Kommandos geöffnet

Über einen rechten Mausklick auf eine **Variable** stehen die folgenden Optionen zur Auswahl (nur beim Push-Prinzip):



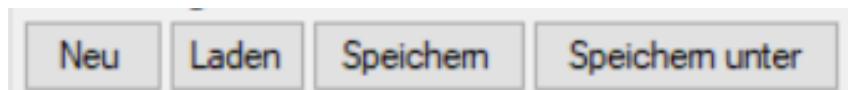
Variable bearbeiten

Es wird ein Fenster geöffnet, in dem die aktuellen Einstellungen der gewählten Variablen angezeigt und angepasst werden können.

Variable löschen

Die Zuordnung der Variable zum Gerät wird gelöscht.

Im Bereich 3 gibt es zudem die folgenden Schaltflächen:



Neu

Die aktuelle User-Konfiguration wird gelöscht und durch eine leere Konfiguration ersetzt.

Laden

Es wird ein Datei-Auswahldialog geöffnet, über den eine User-Konfiguration ausgewählt werden kann. Wird eine gültige Konfiguration gewählt, werden die entsprechenden Daten in diesem Bereich aktualisiert.

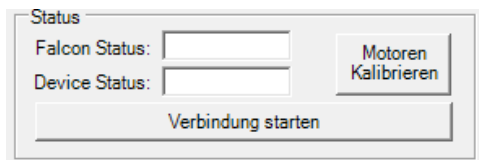
Speichern

Die aktuelle User-Konfiguration wird am Standard-Ablageort abgespeichert. (Der Standard-Ablageort ist in den Basiseinstellungen hinterlegt)

Speichern unter

Die aktuelle User-Konfiguration wird in einer neuen Datei gespeichert. Es wird ein Datei-Auswahldialog geöffnet, über den der gewünschte Ablageort und die Dateibezeichnung gewählt werden können.

Hauptformular Bereich 4 – Verarbeitungssteuerung



In diesem Bereich wird die eigentliche Verarbeitungslogik des Programms gestartet.

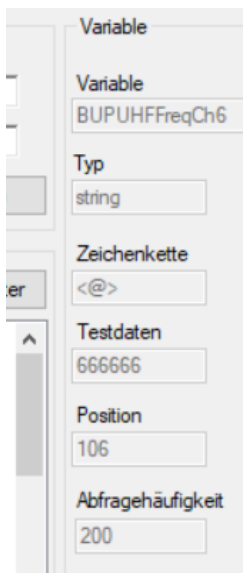
Das Feld „Falcon Status“ zeigt den Verbindungsstatus mit Falcon BMS bzw. DCS an. Unterschieden wird zwischen keiner Verbindung, Verbindung (2D-Welt) und Verbindung (3D-Welt).

Das Feld „Device Status“ zeigt bei laufender Verarbeitung an, ob eine COM-Verbindung mit dem oder den Arduinos hergestellt werden kann. Die Anzeige ist grün, wenn zu allen definierten Arduinos eine Verbindung besteht. Besteht keine Verbindung, ist die Anzeige rot. Wenn mehrere Arduinos definiert wurden und nicht zu allen eine Verbindung aufgebaut werden konnte, ist die Anzeige gelb.

Die Schaltfläche „Motoren Kalibrieren“ stellt eine vorübergehende Verbindung mit allen eingerichteten Arduinos her und sendet den Befehl zur Kalibrierung aller angeschlossenen Geräte. Dies ist insbesondere bei der Nutzung von Stepper-Motoren relevant, um die Position dieser Motoren in die Nullstellung zu bringen. Siehe Kapitel 2.2.

Die Schaltfläche „Verarbeitung starten“ löst die Hauptschleife des Programms aus. Die Hauptschleife liest kontinuierlich die aktuellen Daten der SharedMem des Falcon BMS aus und verwaltet die Datenübertrag an den/die Arduinos und die Veranlassung zur Versendung von Signalen über einen Joystick/Tastaturcontroller.

Hauptformular Bereich 5 – Detailinformationen



In diesem Bereich werden Detailinformationen angezeigt. Die Anzeige ist abhängig davon, ob im Konfigurationsfester (Bereich3) der TOP-Knoten, ein Gerät, Variable oder Eingabekommando ausgewählt wurde.

Mit einem Klick auf die Schaltfläche „Datenmonitor“ kann dieser gestartet werden.

The screenshot shows the 'Datenmonitor' application window. At the top, there are tabs: 'A/C Data', 'Systems', 'EngineData' (selected), 'Warning Lights', 'ECM TWP CMDS', 'ADI HSI', and 'DED PFL'. The main area is divided into three panels: 'Engines', 'Fuel', and 'Other'. The 'Engines' panel contains fields for Nozzle, RPM, FTIT, OilPress, and Fuel Flow. The 'Fuel' panel contains fields for Internal Fuel, External Fuel, Fuel Fwd, Fuel Aft, and Fuel Total. The 'Other' panel contains checkboxes for PU on, Hydrazine, and Air, and a field for EPU Fuel. At the bottom, there are fields for 'Flightdata1 Version' and 'Flightdata2 Version', a 'Sync' section with 'On' and 'Off' radio buttons, and a 'Schließen' button.

Der Datenmonitor umfasst fast alle in der SharedMem verfügbaren Datenfelder. Aus Gründen der Übersichtlichkeit wurden diese in verschiedene Reiter unterteilt.

A close-up of the 'Sync' control. It shows a label 'Sync' and two radio buttons: 'On' (selected) and 'Off'.

Der Datenmonitor umfasst zwei Funktionen, die über den Synchronisationsmodus gesteuert werden.

Sync = on

Der Datenmonitor dient zur Anzeige der aus der SharedMem von BMS/DCS gelesenen Daten und kann zur Fehlersuche genutzt werden, wenn Anzeigen sich nicht wie gewünscht verhalten. Datenfelder können im synchronisierten Modus nicht verändert werden.

Sync=off

Der Datenmonitor dient als Konsole zur Eingabe von Daten. Die Datenfelder können bearbeitet werden. Änderungen werden an die Arduinos übertragen und können zum Prüfen von Funktionen (LED, Motoren usw.) genutzt werden.

Hauptformular Bereich 6 – Programm beenden

A rectangular button with the text 'Programm beenden' in a blue font.

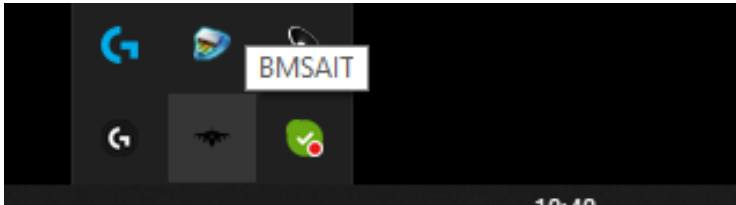
Hier wird das Programm beendet.

Hauptformular Bereich 7 – Fenstermanagement

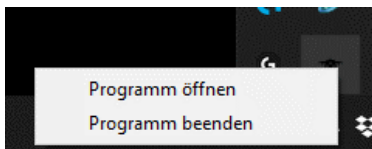
Three standard window management icons: a horizontal line for minimize, a square for maximize, and an 'X' for close.

Hier kann das Programm durch einen Klick auf das Kreuz beendet werden.

Ein Klick auf den linken Strich minimiert die Anzeige des Programms. Eine laufende Verarbeitung läuft dabei weiter.

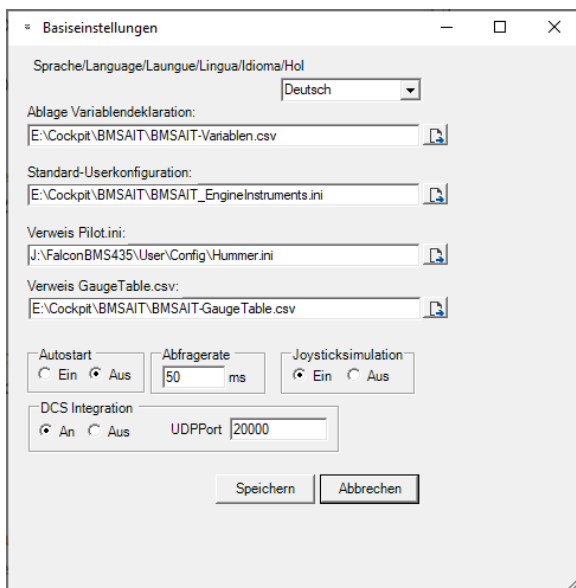


Das Programm BMSAIT wird dann als Symbol in der Task-Leiste angezeigt.



Um das Programm wieder als Fenster zu öffnen, ist bei dem Icon mit einem Rechtsklick der Befehl „Programm öffnen“ auszuwählen.

3.2.2. Basiseinstellungen



Die Software legt in dem Ordner, in dem sich die ausführbare Datei befindet, eine Konfigurationsdatei „BMSAIT.cfg“ ab, in der die gewählten Basiseinstellungen gespeichert und automatisch beim Start des Programms geladen werden.

Abbildung 3: BMSAIT Basiseinstellungen

Zu den Basiseinstellungen gehören:

- Auswahl der Sprache
Hier kann eine Sprache für die Anzeige von BMSAIT umgeschaltet werden (derzeit Deutsch oder Englisch).

- Ablage Variablendeklaration
Über die Auswahlhilfe ist der Ordner festzulegen, in dem sich die Datei BMSAIT-Variablen.csv befindet. Diese Angabe ist zwingend erforderlich.
- Standard-Userkonfiguration
Hier wird festgelegt, welche Konfiguration (Zuordnung von Arduino-Geräten, Variablen, Tastaturkommandos) automatisch beim Start des Programms geladen werden soll. Die Angabe ist optional.
- Verweis Pilot.ini
Für die Ausgabe ergänzender manueller Frequenzen auf dem Backup-Radio ist hier der Ablageort der Pilot.ini im User/Config-Ordner des BMS anzugeben. Die Angabe ist nur erforderlich, wenn Angaben der Backup-Frequenzen gewünscht sind (siehe Beispiel BUPRadio).
- Ablage GaugeTable.csv
Über die Auswahlhilfe ist der Ordner festzulegen, in dem sich die Datei BMSAIT-GaugeTable.csv befindet. Diese Angabe ist zwingend erforderlich.
- Autostart
Hiermit wird festgelegt, ob beim Start des Programms sofort die Kommunikation mit Falcon BMS und den gewählten Arduinos aufgenommen wird. Standardwert ist „Aus“.
- Abfragerate
Hiermit wird festgelegt, wie lange zwischen zwei Abfragen aktueller Daten der SharedMem des Falcon gewartet wird. Standardwert sind 200ms. Bei der Ansteuerung präziser Instrumente (z.B. Analoganzeigen über Stepper/Servomotoren) sollte der Wert gesenkt werden.
- Joysticksimulation
Wird dies aktiviert, nimmt BMSAIT Verbindung mit einem oder mehreren virtuellen Joysticks auf, um darüber Signale an BMS senden zu können. Dies setzt voraus, dass auf dem Rechner die „vjoy“-Software installiert wurde und mindestens ein virtueller Joystick eingerichtet ist (siehe Kapitel 3.3).
- DCS Integration
Wird diese Option aktiviert, kann BMSAIT auch zur Auswertung und Übertragung von Fluginformationen aus DCS genutzt werden. Siehe Kapitel.
- UDP Port
Hier wird festgelegt, auf welchem UDP Port die Fluginformationen aus DCS über DCSBIOS empfangen werden sollen.

3.2.3. Geräte einrichten

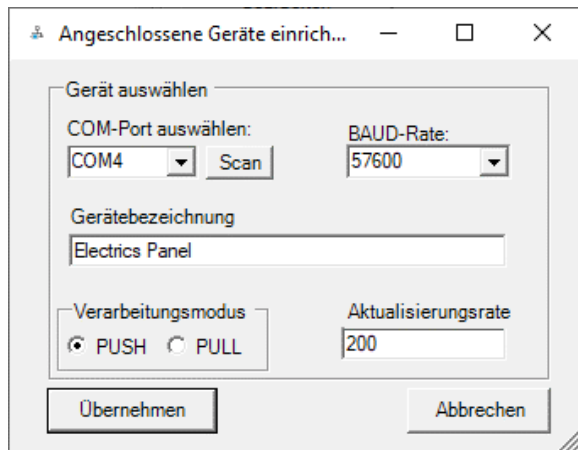


Abbildung 4: BMSAIT Geräteeinstellungen

Über dieses Formular erfolgt die Einrichtung eines neuen Gerätes oder die Bearbeitung eines bereits hinterlegten Gerätes.

Im Feld COM-Port ist der Port auszuwählen, über den mit dem Arduino kommuniziert werden soll. In der Auswahlliste werden alle Windows bekannten COM-Ports angeboten. Sollte der COM-Port eines Arduino hier nicht auftauchen, ist die USB-Verbindung zum Arduino zu trennen und neu zu verbinden. Nach einem Neuaufruf dieses Formulars sollte der COM-Port verfügbar sein.

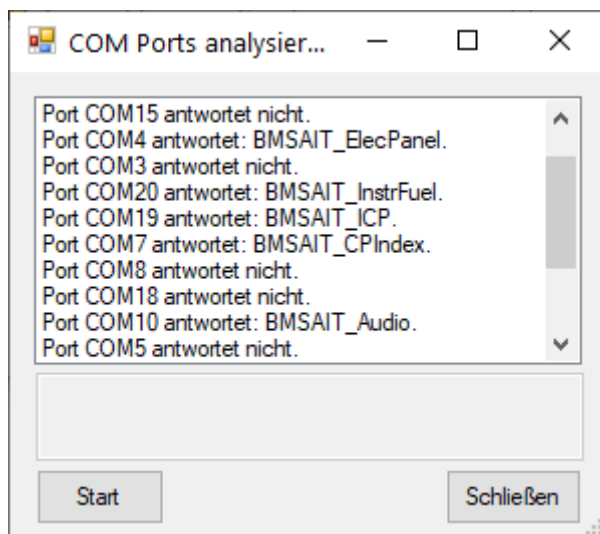
Im Feld BAUD-Rate ist die Verbindungsgeschwindigkeit für die Kommunikation zwischen PC und dem Arduino festzulegen. Der hier eingegebene Wert muss mit dem Wert übereinstimmen, der in der Programmierung des Arduino verwendet wird (siehe 4.2.2).

Im Feld Gerätebezeichnung kann ein Name zur besseren Identifizierung des Arduinos hinterlegt werden.

Mit der Verarbeitungsmethode wird festgelegt, nach welcher Logik der Datenaustausch zwischen der Windows-App und dem Arduino erfolgt. (siehe Kapitel 2.1.3 und 2.1.4)

Die Aktualisierungsrate legt fest, wie oft Daten zur Auffrischung an den Arduino gesendet werden. Der Wert sollte zwischen 50ms und 500ms liegen. Standard ist 200ms.

Die Schaltfläche Übernehmen führt nach einer Plausibilitätsprüfung der Eingaben die Speicherung der vorgenommenen Änderungen durch.



Mit der Schaltfläche Scan wird ein Unterformular gestartet. Hier werden mit der Schaltfläche Start alle verfügbaren COM-Ports des PC überprüft und eine Rückmeldung über den Status der Anschlüsse und ggf. gefundene Arduino-Boards zurückgegeben. Wurde eines der BMSAIT Sketches auf einen Arduino geladen, wird hier die einprogrammierte ID des Arduino (siehe 4.2.2 „ID“) angezeigt.

3.2.4. Formular Variablenauswahl

Auswahl Datenextraktion

Gewähltes Gerät
COM-Port: 0 COM14

Gewählte Datenvariablen

ID	Typ	Bez
11...	string	BupUhf...
11...	string	BupUhf...
16...	string	BUPUH...

Übernehmen
Abbrechen

Neue Variable auswählen

ID	Gruppe	Typ	Bez	Beschreibung
1050	NAVIGAT...	float	xDot	Ownship North Rate (ft/sec)
1060	NAVIGAT...	float	yDot	Ownship East Rate (ft/sec)
1070	NAVIGAT...	float	zDot	Ownship Down Rate (ft/sec)
1080	NAVIGAT...	float	latitude	Ownship latitude in degrees (as known by avionics)
1090	NAVIGAT...	float	longitude	Ownship longitude in degrees (as known by avionics)
1100	DED/PFL	string[]	PFLLines	25 usable chars
1110	DED/PFL	string[]	PFLInvert	25 usable chars
1120	RADIO	string	BupUhfFreq	BUP UHF channel frequency
1130	RADIO	string	BupUhfPreset	BUP UHF channel preset

Formatierungsstring: <@> **Testdaten:** 06 **Position Variable im Arduino:** 0 **Aktualisierungshäufigkeit (s):** 1

Abbildung 6: BMSAIT Variablenauswahl

Das Formular wird über das Hauptformular aufgerufen, wenn dort in der User-Konfiguration mit einem Rechtsklick auf einen COM-Port der Befehl „Variable hinzufügen“ oder bei einer Variablen der Befehl „Variable bearbeiten“ gewählt wird.

Das Formular zur Bearbeitung von Datenvariablen ist in folgende Bereiche unterteilt:

Bereich 1 – Bezugsdaten

In diesem Bereich wird das Gerät/der COM-Port angezeigt, der auf dem Hauptformular aktuell gewählt ist. Zudem wird eine Liste mit Variablen angezeigt, die dem Gerät bereits zugeordnet sind.

Bereich 2 – Abschluss

Mit der Schaltfläche Übernehmen werden die vorgenommenen Eingaben zu einer neuen oder bestehenden Datenvariable hinzugefügt. Vorher erfolgen mehrere Plausibilitätsprüfungen, um Fehleingaben zu verhindern. Nach Abschluss der Speicherung wird das Formular geschlossen.

Mit der Schaltfläche Abbrechen werden keine Änderungen gespeichert und das Formular geschlossen.

Bereich 3 – Auswahl einer Datenvariablen

In diesem Fenster werden alle verfügbaren Datenvariablen des BMSAIT angezeigt. Die Variablen wurden dazu bei Programmstart aus der externen Datei „BMSAIT-Variablen.csv“ eingelesen.

Die Sortierung der Auflistung kann durch einen Klick auf die Spaltenüberschriften verändert werden.

Aus der Liste kann ein Element ausgewählt werden. Möchte man einem Gerät mehrere Datenvariablen zuordnen, muss dies in mehreren Aufrufen dieses Formulars erfolgen.

Eine Beschreibung der verschiedenen Spalten findet sich im Anhang dieses Dokumentes bei der Beschreibung des Datensatzes „Variable“.

Bereich 4 – Ergänzende Merkmale

Hier können ergänzende Merkmale zu einer Datenvariablen eingegeben werden. Ohne eine Befüllung des Feldes Position ist eine Speicherung nicht möglich. Alle anderen Felder sind optional, ich empfehle diese aber mit sinnvollen Werten zu befüllen.

Eine Beschreibung der verschiedenen Felder findet sich im Anhang dieses Dokumentes bei der Beschreibung des Datensatzes „Zuordnung“.

3.2.5. Formular Eingabekommandos

The screenshot shows the 'Eingabekommandos' window. At the top, there are fields for 'Arduino' (set to 'COM20'), 'Lower Center Console', and 'KommandoID' (set to '51'). Below these is a table with columns 'ID' and 'Beschreibung'. The table lists various commands like 'CRS INC +1', 'CRS DEC -1', etc. To the right of the table is a 'Kommandobeschreibung' field containing 'CRS DEC -1'. Below this is a section for defining the input, with tabs for 'Tastatur', 'JoyTaste', and 'JoyAchse'. The 'Tastatur' tab is active, showing a 'Taste' dropdown set to 'TasteV', checkboxes for 'SHIFT', 'ALT', and 'STRG' (all checked), and an 'Aktion' dropdown set to 'drücken+loslassen'. At the bottom of the window are five buttons: 'Hinzufügen', 'Löschen', 'Speichern', 'Speichern und Schließen', and 'Abbrechen'. Red boxes and numbers 1-4 highlight specific areas: 1 points to the 'Hinzufügen' button, 2 points to the 'KommandoID' field, 3 points to the 'Kommandobeschreibung' field, and 4 points to the 'Speichern' button.

Abbildung 7: BMSAIT Eingabekommandos

Dieses Formular wird über die Schaltfläche bei den Basiseinstellungen auf dem Hauptformular aufgerufen. Mit dem Formular Eingabekommandos können Keyboard- oder Joysticksignale definiert werden. Diese Signale werden einer Identifikationsnummer (Kommando-ID) zugeordnet. Wird vom Arduino eine Kommando-ID gesendet, wird das entsprechende Signal durch die WindowsApp in ein Tastatur- oder Joysticksignal umgewandelt und so an die gerade aktive Anwendung gesendet.

Hinweis: BMSAIT sendet nur dann ein Signal, wenn Falcon BMS oder DCS gestartet sind (2D oder 3D) oder der Testmodus aktiviert wurde.

Das Formular ist in drei Bereiche unterteilt:

Bereich 1 – Auswahlliste

ID	Beschreibung
1	UHF Cycle down
2	UHF Cycle Up
3	UHF Both
4	UHF Mode Cycle down
5	UHF Mode Cycle Up
6	UHF Mode GRD
7	dxBtn UHF off
8	dxBtn UHF Test
9	Analog UHF VOL

In der Auswahlliste sind alle Kommandos zu sehen, die zu einem gewählten Arduino-Gerät in der aktuellen Userkonfiguration gespeichert sind. Ein bestehender Eintrag kann durch einen Klick auf einen Eintrag in der Liste ausgewählt werden. Die dazugehörigen Informationen werden dann auf der rechten Seite des Formulars angezeigt und können dort bearbeitet werden.

Mit der Schaltfläche Hinzufügen kann ein neues Kommando angelegt werden.

Mit der Schaltfläche Löschen wird das gerade ausgewählte Kommando gelöscht.

Bereich 2 – Kopfdaten

Arduino:	<input type="text" value="COM20"/>	<input type="text" value="Lower Center Console"/>	KommandoID:	<input type="text" value="51"/>									
<table><tr><th>ID</th><th>Beschreibung</th><th></th></tr><tr><td>50</td><td>CRS INC +1</td><td></td></tr><tr><td>51</td><td>CRS DEC -1</td><td></td></tr></table>			ID	Beschreibung		50	CRS INC +1		51	CRS DEC -1		Kommandobeschreibung:	<input type="text" value="CRS DEC -1"/>
ID	Beschreibung												
50	CRS INC +1												
51	CRS DEC -1												

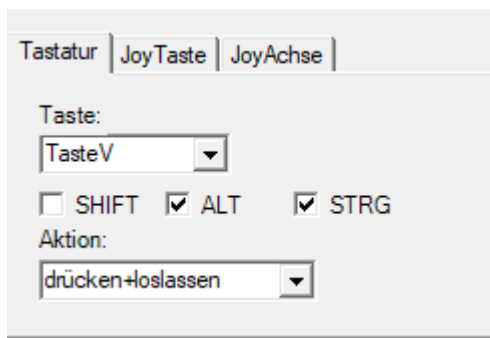
In den Kopfdaten wird das aktuell gewählte Arduino-Gerät angezeigt, für das die Kommandos verwaltet werden sollen. Im Feld KommandoID wird die Identifikationsnummer des Signals angezeigt. Bei einem bislang noch nicht gespeicherten Kommando wird hier bis zur Speicherung der Wert NEU angezeigt.

Darunter befindet sich ein Feld für eine verbale Beschreibung des Kommandos. Dies dient zur besseren Identifikation der Kommandos innerhalb BMSAIT und kann frei belegt werden.

Bereich 3 – Modusauswahl und Detailinformationen

BMSAIT unterstützt sowohl das Auslösen von Tastatursignalen als auch die Nutzung von Joystickkommandos oder -achsen. In der Modusauswahl kann eine Festlegung getroffen werden, welche Art von Signal das Kommando auslösen soll. Wurde die Joystickverarbeitung aktiviert und wurde die vJoy Software installiert (siehe 3.3), stehen drei Optionen zur Auswahl. Wurde vJoy nicht aktiviert, steht nur die Tastatursignalooption zur Verfügung.

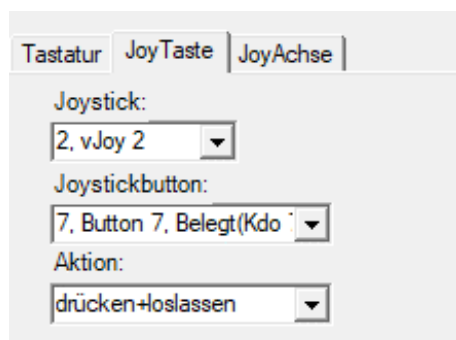
Reiter Tastatursignale



Im Feld Zeichen kann aus einer Auswahlliste ein Tastendruck ausgewählt werden. Soll der Tastendruck in Kombination mit einem Modifikator (Shift, Alt oder Strg) erfolgen, ist ein Haken bei dem entsprechenden Modifikator zu setzen.

Mit der Aktion kann das Tastenverhalten (nur drücken, nur loslassen, drücken&loslassen) festgelegt werden.

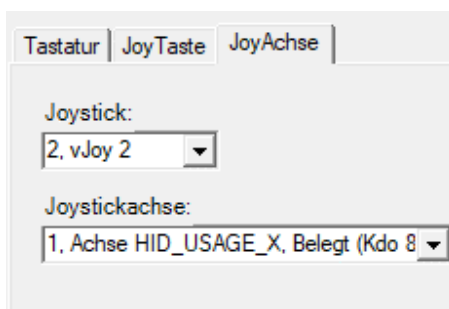
Reiter Joystickbutton



Zuerst ist im Feld Joystick aus der Auswahlliste der gewünschte Joystick auszuwählen. Im Feld Joystickbutton werden anschließend alle verfügbaren Buttons des Joysticks angezeigt (in der vJoy Software kann die Zahl der Button eingestellt werden). Zu jedem Button wird angegeben, ob dieser noch frei ist oder bereits durch ein anderes Kommando belegt wurde. Der gewünschte Button ist auszuwählen. Mit der Aktion kann das

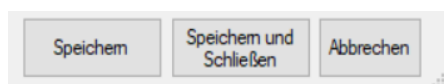
Tastenverhalten (nur drücken, nur loslassen, drücken&loslassen) festgelegt werden.

Reiter Joystickachse



Zuerst ist im Feld Joystick aus der Auswahlliste der gewünschte Joystick auszuwählen. Im Feld Joystickachse werden anschließend alle verfügbaren Achsen des Joysticks angezeigt (in der vJoy Software können die verfügbaren Achsen eingestellt werden). Zu jeder Achse wird angegeben, ob diese noch frei ist oder bereits durch ein anderes Kommando belegt wurde. Die gewünschte Achse ist auszuwählen.

Bereich 4 – Speichern/Beenden



Mit der Schaltfläche **Speichern** wird eine gerade vorgenommene Änderung gespeichert. Das Formular bleibt dabei für weitere Anpassungen geöffnet.

Mit der Schaltfläche **Speichern und Schließen** wird die Änderung übernommen und das Formular geschlossen.

Mit der Schaltfläche **Abbrechen** wird die aktuelle Änderung verworfen und das Formular geschlossen.

Hinweis: Die Speicherung im Formular Tastaturkommandos erfolgt vorerst nur im Arbeitsspeicher. Um die Änderungen auch beim nächsten Programmstart vorzufinden, muss auf dem Hauptformular die Userkonfiguration in die Auslagerungsdatei geschrieben werden (Hauptformular Klick auf „Speichern“/„Speichern unter“).

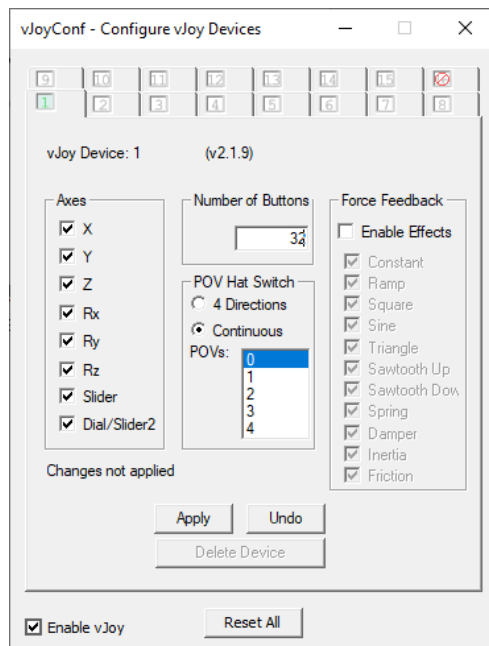
3.3.(optional) Virtuelle Joysticks aktivieren

BMSAIT erlaubt es euch, die Schaltersignale entweder als Tastaturkommandos oder als Joysticksignale an die Flugsimulation weiterzugeben.

Tastaturkommandos lassen sich über die eingerichtete Bibliothek InputSimulator.dll umsetzen. Sollte es aber gewünscht/erforderlich sein, auch Joysticksignale (Buttons oder Analogachsen) auszulösen, ist hierfür noch eine Vorarbeit erforderlich. Joysticksignale lassen sich nämlich nur erzeugen, wenn man in Windows dafür ein Gerät eingerichtet hat. BMSAIT kann dieses Gerät dann übernehmen und im Namen dieses Gerätes Signale an die Flugsimulation senden.

Um hier nicht extra einen physikalischen Joystick anschließen zu müssen, gibt es die Möglichkeit des Anlegens virtueller Joysticks. Für BMSAIT nutze ich hier die vJoy Software. Ihr könnt diese Software hier gratis herunterladen:

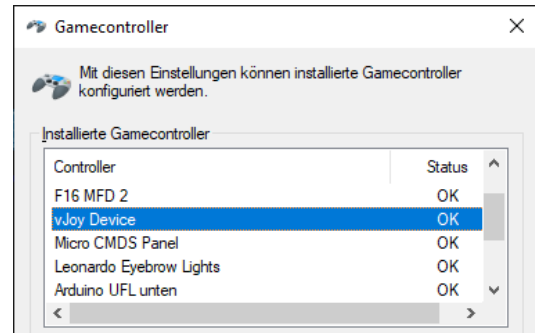
<http://vjoystick.sourceforge.net/site/index.php/download-a-install/download>



Nach der Installation ruft Ihr den Konfigurator (VJoyConf) auf. Darin aktiviert Ihr einen Joystick. Standard sollte ein aktivierter Joystick mit 32 Buttons und allen Analogachsen sein. ForceFeedback wird nicht benötigt und kann deaktiviert werden. Soll BMSAIT mehr als 32 Buttons oder 8 Analogachsen steuern, könnt ihr weitere Joysticks aktivieren (Klick auf einen der nummerierten Reiter am oberen Rand und anschließend ein Klick unten auf den Button „Add Device“).

Habt ihr Änderungen vorgenommen, muss Windows neu gestartet werden, bevor diese wirksam werden.

Unter der Gamecontroller-Einstellung von Windows sollte der vJoy nun sichtbar sein. Wenn in BMSAIT die Joystickverarbeitung aktiviert ist, kann in der Eingabeverarbeitung nun darauf zugegriffen und ein Eingabekommando eines Arduino den Buttons und Achsen des vJoy zugewiesen werden (siehe 3.2.5).



3.4.(work in progress) DCS Integration

Wie man dem Namen dieses Projektes entnehmen kann, geht es bei BMSAIT um eine Schnittstelle für Falcon BMS. Dennoch kann es vorkommen, dass man auch mal in DCS fliegen und nicht auf die Funktionen des Homecockpits verzichten möchte. Bei Arduino Lösungen gibt es sowohl auf der BMS Seite (F4toSerial) als auch auf der DCS Seite (DCS BIOS) gute Lösungen. Allerdings müsste man beim Wechsel der Software auch die Programmierung der Arduinos anpassen.

Mit BMSAIT versuche ich einen konfigurationsfreien Wechsel zu ermöglichen. Die BMSAIT App registriert automatisch ob BMS oder DCS gestartet ist und verarbeitet die aus den Simulationen empfangen Daten soweit, dass keine Veränderungen der Arduino Programmierungen erforderlich sind.

Die Funktion ist noch in Erprobung. Einige Basisfunktionen sind bereits funktionsfähig (z.B. IndexerLights, einige Engine Instruments), aber viele weitere Daten müssen noch geprüft und soweit abgeglichen werden, dass Daten wirklich in identischer Form vorliegen und von den Arduinos richtig zur Anzeige gebracht werden können.

Um die Funktion zu nutzen, ist die Einstellung „DCS Integration“ in den Basiseinstellungen zu aktivieren und ein UDP Port anzugeben, auf dem Daten von DCS empfangen werden sollen.

In DCS ist eine Installation des Mods „[DCS-BIOS](#)“ erforderlich. Diese Software ermöglicht eine Extraktion von Fluginformationen. Die Daten werden über das TCP/UDP Protokoll versendet und auf diesem Wege von BMSAIT empfangen. DCSBIOS sendet normalerweise auf Port 5010. In der Export.lua von DCS kann eine Option aktiviert werden, dass DCSBIOS auch auf anderen Ports senden kann. Der konkrete Port ist in der BIOSconfig.lua im Installationsorder von DCSBIOS einzustellen.

```
BIOS.protocol_io.connections = {  
    BIOS.protocol_io.DefaultMulticastSender:create(),  
    BIOS.protocol_io.TCPServer:create(),  
    BIOS.protocol_io.UDPSender:create({ port = 20000, host = "127.0.0.1" }},  
    BIOS.protocol_io.UDPListener:create({ port = 7778 })  
}
```

Abbildung 8: Senderangaben DCS-BIOS in BIOSconfig.lua

(Ich persönlich nutze Port 20000, da ich auf Port 5010 Probleme mit der Verbindung hatte).

3.5. GaugeTable

BMSAIT liegt neben der Variablenliste eine zweite CSV-Datei, die „BMSAIT-GaugeTable.csv“ bei.

Diese Datei enthält Vorgaben zur Ansteuerung von Analog-Rundanzeigen (derzeit RPM und FTIT). Die Tabelle ist erforderlich, da die Skala mancher Anzeigen nicht linear ist, d.h. der Abstand zwischen zwei Schritten ist auf der Skala nicht überall gleich (Beispiel: Auf der RPM Skala werden zwischen 0° (0% RPM) und 90° (50% RPM) 50% der RPM abgedeckt; zwischen 180° (80%) und 270° (100%) nur noch 20%.

Die Ausgabe aus der SharedMem berücksichtigt dies nicht, sondern liefert am Beispiel der RPM Anzeige lediglich den Wert der RPM (bspw. 65%).

Die GaugeTable ermöglicht es, dass die Zeigerbewegungen angepasst werden können, ohne dass die BMSAIT App umprogrammiert werden muss. In der Table werden Zuordnungen hinterlegt, bei welchen Werten einer Skala welcher Winkel der Zeigerbewegung erreicht werden soll. Es steht jedem Nutzer frei hier wenige oder viele Zuordnungen zu hinterlegen. BMSAIT wird zwischen den die Bewegung zwischen den vorgegebenen Positionen interpolieren.

	A	B	C	D	E	F	G	H	I	J	K
1	<RPM>										
2	Gauge value	60	70	75	80	85	90	100	105	107	110
3	Gauge positi	22500	26000	32250	38500	51000	55750	57250	63500	65535	65535

Der Screenshot zeigt die Vorgaben für die RPM Skala. Das Beispiel mit den 65% würde anhand dieser Einstellungen der GaugeTable den Zeiger in die Position 24250 (entspricht 133°) fahren.

4. Das Arduino Programm

4.1. Beschreibung der Vorbereitungen

4.1.1. Vorwort


















Arduinos bringen eine schier unerschöpfliche Zahl an Anschlussmöglichkeiten von Ein- und Ausgabegeräten mit sich. Aufgrund dessen ist es nicht möglich, eine Software bereitzustellen, die alle Möglichkeiten auf Knopfdruck abdeckt. Das hier vorliegende Coding stellt daher keine vollständigen Lösungen, sondern eher nur Anwendungsbeispiele bereit. Beim Anschluss eines jeden Gerätes muss das Zusammenspiel von Arduino-Programmierung/Customizing und der Verkabelung der Peripherie beachtet werden. Dies betrifft z.B. die genaue Belegung der PINs, ein besonderes Gerät oder der Wahl der richtigen Vorwiderstände bei der Ansteuerung von LEDs.

4.1.2. Das Programmieren eines Arduino mit dem BMSAIT Sketch

Damit der Arduino Daten des Windows-Programms BMSAIT empfangen und an angeschlossene Geräte ausgeben kann, muss eine Software auf den Arduino geladen werden. Die Software liegt als C++ Programmcode vor.

4.1.3. Herunterladen des Programmcodes

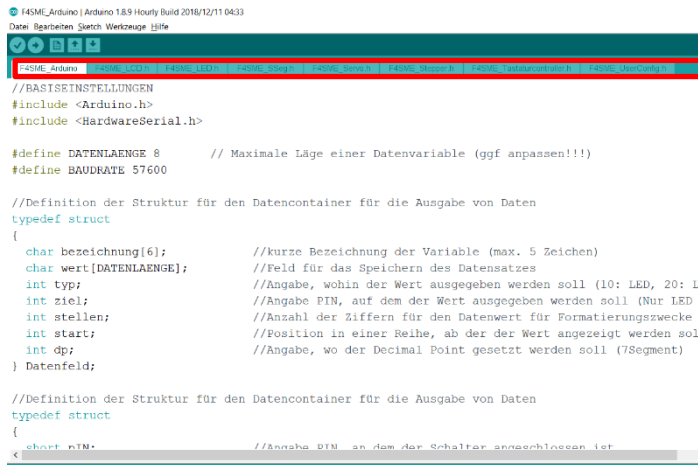
Der Programmcode für die BMSAIT Funktionen besteht aus mehreren Dateien (einer .ino Datei, mehrere .h oder .cpp Dateien). Diese müssen zusammen in einem Ordner abgelegt werden:

-  BMSAIT_Vanilla.ino
-  BMSAIT_UserConfig.h
-  BMSAIT_Switches.h
-  BMSAIT_StepperX27.h
-  BMSAIT_StepperVID.h
-  BMSAIT_Stepper28BYJ48.h
-  BMSAIT_SSegTM1637.h
-  BMSAIT_SSegMAX7219.h
-  BMSAIT_ServoPWMShield.h
-  BMSAIT_Servo.h
-  BMSAIT_Placeholder.h
-  BMSAIT_MotorPoti.h
-  BMSAIT_LED.h
-  BMSAIT_LCD.h
-  BMSAIT_Encoder.h
-  BMSAIT_Buttonmatrix.h
-  BMSAIT_Analogachse.h

4.1.4. Installation einer Entwicklungsumgebung

Diese Dateien enthalten lesbaren Text und können daher theoretisch mit einem normalen Editor angesehen werden. Zur Bearbeitung und vor allem zum Übertragen der Software auf den Arduino wird eine gesonderte Software benötigt. Ich empfehle hier die kostenlose Arduino IDE (arduino.cc/en/main/software).

Wenn die Arduino-IDE installiert ist, kann die Arduino-Software durch einen Doppelklick auf die .ino-Datei aufgerufen werden.



```

145ME_Arduino | Arduino 1.8.9 Hourly Build 2018/12/11 04:33
Datei Bearbeiten Sketch Werkzeuge Hilfe

//BASTISEINSTELLUNGEN
#include <Arduino.h>
#include <HardwareSerial.h>

#define DATENLAENGE 8 // Maximale Länge einer Datenvariable (ggf anpassen!!!)
#define BAUDRATE 57600

//Definition der Struktur für den Datencontainer für die Ausgabe von Daten
typedef struct
{
    char bezeichnung[6]; //kurze Bezeichnung der Variable (max. 5 Zeichen)
    char wert[DATENLAENGE]; //Feld für das Speichern des Datensatzes
    int typ; //Angabe, wohin der Wert ausgegeben werden soll (10: LED, 20: L
    int ziel; //Angabe PIN, auf dem der Wert ausgegeben werden soll (Nur LED
    int stellen; //Anzahl der Ziffern für den Datenwert für Formatierungszwecke
    int start; //Position in einer Reihe, ab der der Wert angezeigt werden sol
    int dp; //Angabe, wo der Decimal Point gesetzt werden soll (7Segment)
} Datenfeld;

//Definition der Struktur für den Datencontainer für die Ausgabe von Daten
typedef struct
{
    char nPIN; //Angabe PIN, an dem der Schalter angeschlossen ist
}

```

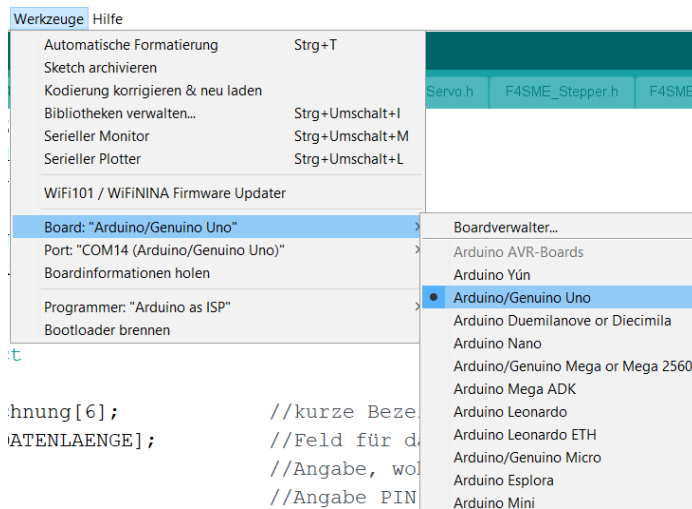
Bitte achtet beim Laden darauf, dass die verschiedenen .h / .cpp Dateien als Reiter auftauchen. Wenn nicht, wurden die Dateien nicht gemeinsam mit der ino-Datei abgelegt. Das Programm wird dann wahrscheinlich nicht lauffähig sein. Es werden aber nur die .h/.cpp Dateien benötigt, die Ihr in eurem Projekt aktiviert. Nicht benötigte Moduldateien können gefahrlos gelöscht werden.

Vor der erstmaligen Nutzung ist es erforderlich, in der Arduino IDE zusätzliche Bibliotheken (Programmerweiterungen) zu installieren. Welche Bibliotheken erforderlich sind, hängt von den Geräten ab, die Ihr über den Arduino ansteuern wollt.

In den mitgelieferten Beispielen wurden folgende Bibliotheken verwendet:

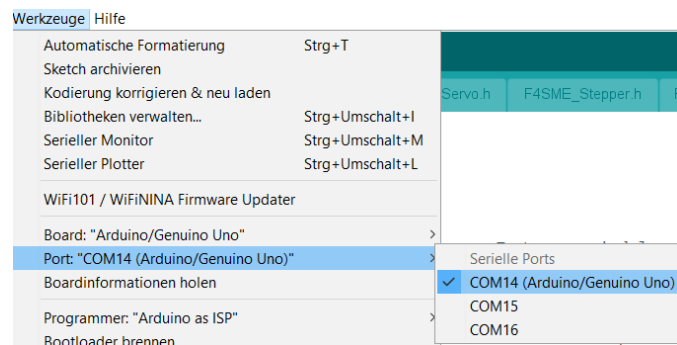
Programmteil	Bibliothek
LCD (16x2 oder 20x4 Displays mit HD44067 Controller)	LiquidCrystal_I2C.h https://github.com/johnrickman/LiquidCrystal_I2C
7-Segment-Anzeige (MAX7219 Controller), LED Matrix	LedControl.h http://wayoda.github.io/LedControl/
7-Segment-Anzeige (TM1367 Controller)	LEDDisplayDriver.h http://lygte-info.dk/project/DisplayDriver%20UK.html
Servo-Motoren (direkt)	Servo.h https://github.com/arduino-libraries/Servo
Servo-Motor-Shield (PCA9685 Controller)	Adafruit_PWMServoDriver.h https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library
Stepper-Motoren (Standard)	Stepper.h https://github.com/arduino-libraries/Stepper
Stepper-Motoren (x27.168)	SwitecX25.h https://github.com/clearwater/SwitecX25
Stepper-Motor-Shield (VID6606)	SwitecX12.h https://github.com/clearwater/SwitecX25
OLED Displays (OLED, FFI, SBI, DED/PFL)	U8g2.h https://github.com/olikraus/u8g2

4.1.5. Auswahl des Arduino Boards



Um die Software auf den Arduino zu laden, muss in der Arduino IDE das Arduino-Board ausgewählt werden, das ihr nutzen wollt. Dies erfolgt über den Reiter Werkzeuge. Beim Eintrag „Board: xxx“ muss die Art des genutzten Boards ausgewählt werden (z.B. Arduino UNO).

4.1.6. Auswahl des COM-Port



Beim Eintrag „Port“ ist der Serielle Port auszuwählen, der dem Arduino-Board in Windows zugewiesen ist.

Der einem Board zugeordnete Port kann über den Windows-Gerätemanager, die Arduino-Entwicklungsumgebung (Arduino IDE) oder auch in BMSAIT in Erfahrung gebracht werden.

4.1.7. Prüfen der Arduino-Software

Die Arduino IDE prüft die Software im Rahmen des Hochladevorgangs auf Fehler. Sollten Programmfehler gefunden werden, wird der Ladevorgang abgebrochen.

Ich empfehle daher die Software vor dem Hochladen bereits selbst auf Programmfehler zu prüfen. Dazu ist im Menü unter „Sketch“ der Befehl Überprüfen/Kompilieren auszuwählen.

Wenn dies das erste Mal durchgeführt wird, kann die Software Fehler anzeigen. Dies liegt daran, dass der Programmcode Zugriffe auf Bibliotheken haben möchte, die noch nicht installiert sind (siehe 4.1.4).

4.1.8. Hochladen der Arduino-Software

In der Arduino IDE findet sich im Menü „Sketch“ der Befehl Hochladen. Mit diesem Befehl wird, wenn das richtige Arduino-Board / der richtige COM-Port gewählt wurde und die aktuelle Programmierung fehlerfrei ist, die Software auf den Arduino geladen. Der Arduino startet daraufhin neu und wird die programmierte Verarbeitung beginnen.

4.2. Beschreibung des Arduino Sketches

Die Arduino Software besteht aus mehreren Einzeldateien, die im folgenden Module genannt werden. Die Trennung in Module dient der Übersichtlichkeit des Programmcodes, da unterschiedliche Anwendungsgebiete (Arten von Geräten, die über den Arduino angesteuert werden sollen) unterschiedliche Programmierungen zur Ansteuerung benötigen. Die Aufteilung in Module erlaubt es, dass der Arduino nur die Programmteile lädt, die für eine Anwendung erforderlich sind. Neben dem BMSAIT Programm und dieser Dokumentation solltet ihr auch mehrere Arduino Projekte finden. Das Projekt BMSAIT Vanilla bildet hierbei die Basis und sollte genutzt werden, um eigene Projekte zu starten. Die anderen beiliegenden Projekte sind Beispiele, die jeweils ein fertiges, lauffähiges Projekt für häufige Anwendungsfälle bereitstellen. Die folgende Beschreibung bezieht sich auf das Vanilla-Projekt. Eine Beschreibung der Beispielprogramme findet ihr im Dokumentations-Unterverzeichnis beim Beispielprogramm.

4.2.1. Das Modul BMSAIT_Vanilla

In dem Hauptmodul (der „.ino“ Datei) werden

1. die erforderlichen weiteren Module aufgerufen
2. Globale Werte und Variablen eingerichtet (siehe 5.2.4)
3. der Arduino eingerichtet (Funktion Setup)
4. die Hauptschleife für die kontinuierliche Verarbeitung gestartet (Funktion Loop)
5. Die Kommunikation mit dem PC gesteuert (Funktionen ReadResponse, SendSysCommand, SendMessage, PullRequest, DebugReadback)

Zur Vermeidung von Fehlern sollte in diesem Modul nichts angepasst werden.

Ausnahme: Eine Anpassung kann aber erforderlich sein, wenn ein neues, im Standardcoding nicht berücksichtigtes Modul abgebildet werden soll (z.B. Ansteuerung eines Gerätes, das bislang nicht berücksichtigt ist). Die neuen Typen müssen an der entsprechenden Stelle der Hauptschleife ergänzt werden, um den Aufruf einer eigenen Funktion zur Ansteuerung des Typs zu programmieren:

1. Das neue Modul ist einzubinden
2. Verweis auf das Setup des neuen Moduls
3. Verweis auf eine Kalibrierung des neuen Moduls (nur bei Motoren)
4. Verweis auf die Aktualisierung beim Endlos-Schleifendurchlauf

```
#ifdef ServoMotor
case 40: //Servos
    Update_Servo(x);
    break;
#endif

#ifdef newDevice //define this flag in the top of F4SME_UserConfig.h to activate this block ("define newDevice")
case 69: //assign this type to a variable in the data container to call a new method
    Update_newDevice(x); //program a new method void Update_newDevice(int p){command1;command2;...}to enable your device
    break;
#endif
```

4.2.2. Das Modul UserConfig

In diesem Modul sind wichtige Einstellungen zusammengefasst, die der Nutzer für die Einrichtung des gewünschten Ablaufs anpassen kann/soll. Dieses Basismodul ist für jedes Projekt erforderlich.

4.2.2.1. Funktionsauswahl (Module Selection)

```

4 //MODULE SELECTION - uncomment the modules
5
6 // #define LED //drive
7 // #define LEDMatrix //drive
8 #define LCD //drive LCD
9 // #define SSegMAX7219 //drive
10 // #define SSegTM1637 //drive
11 // #define ServoMotor //drive
12 // #define ServoPWM //drive
13 // #define StepperBYJ //drive
14 #define StepperX27 //drive stepper
15 // #define StepperVID //drive
16 // #define MotorPoti //motor
17 // #define DED_PFL //Enable
18 // #define SpeedBrake //Enable
19 #define Switches //use the switches
20 // #define ButtonMatrix //use the buttons
21 // #define RotEncoder //use the encoder
22 // #define AnalogAxis //use the axis
23 // #define NewDevice //place the device
24

```

Hier wird definiert, welche Module über den Arduino angesteuert werden sollen. Wählt nur die Module aus, die auch wirklich genutzt werden, da jedes Modul Ressourcen beansprucht, die für die eigentlich erforderlichen Arbeiten des Arduino sonst später fehlen können.

Um ein Modul auszuwählen, entfernt die Zeichen „//“ vor dem Wort „#define“

Um ein Modul abzuwählen, setzt vor dem Wort „#define“ die Zeichen „//“

4.2.2.2. Konstanten (Basic Settings)

```

33 //BASIC SETTINGS
34 #define BAUDRATE 57600 // serial connection speed
35 #define POLLTIME 200 // set time between requests
36 #define PULLTIMEOUT 30 // set time to wait for answer
37 // #define PRIORITIZE_OUTPUT //uncomment this to prioritize output
38 // #define PRIORITIZE_INPUT //uncomment this to prioritize input
39 const char ID[] = "BMSAIT_VANILLA"; //Set the ID for the device

```

BAUDRATE

Hier wird die Verbindungsgeschwindigkeit festgelegt, mit der der PC mit dem Arduino zu kommunizieren versucht. Die Angabe muss mit dem Wert übereinstimmen, der in der Windows-App für diesen Arduino angegeben wurde.

POLLTIME

Hier wird eine Zeit in Millisekunden angegeben, die der Arduino zwischen zwei PULL-Anfragen wartet. Standard ist 200ms.

PULLTIMEOUT

Hier wird eine Zeit in Millisekunden angegeben, die der Arduino nach einer PULL-Anfrage auf eine Antwort wartet, bevor die Verarbeitung mit der nächsten Anfrage fortgesetzt wird.

Priorisierung

Arduinos haben nur sehr begrenzte Rechenleistungen und kommen daher schnell an die Grenze der Leistungsfähigkeit. Es kann passieren, dass ein Arduino daher eine eingehende Nachricht erst verspätet oder gar nicht einliest, weil gerade eine Datenausgabe verarbeitet wird, die eventuell etwas länger dauert. Umgekehrt kann es sein, dass ein Ausgabegerät (z.B. ein Motor) nicht gleichmäßig bewegt wird, weil der Arduino zwischendurch mit dem Einlesen der neuen Daten beschäftigt ist. Hier

PRIORITIZE_OUTPUT aktiviert zusätzliche Aufrufe zur Aktualisierung der Datenausgabe und legt somit einen Schwerpunkt auf eine schnelle Ausgabe.

PRIORITIZE_INPUT aktiviert zusätzliche Abfragen angeschlossener Schalter, um möglichst Schnell auf Eingaben reagieren zu können.

Ist keiner der beiden Priorisierungen aktiviert, sind die Ressourcen gleichmäßig zwischen dem Einlesen frischer Daten, der Auswertung der Inputs und der Aktualisierung der Outputs verteilt.

ID

Hier kann dem Arduino eine Bezeichnung zugewiesen werden, mit der sich das Board gegenüber der Windows-App identifizieren kann. Standard ist „BMSAIT_Vanilla“.

4.2.2.3. Gerätetyp (Board Selection)

```
42 //BOARD SELECTION
43
44 #define UNO          //uncommen
45 // #define NANO      //uncomm
46 // #define MICRO     //uncomm
47 // #define LEONARDO  //uncomm
48 // #define MEGA      //uncomm
49 // #define DUE       //uncomm
50 // #define DUE_NATIVE //uncomm
51 // #define ESP        //uncomm
```

In diesem Bereich wird ausgewählt, welcher Typ von Arduino genutzt wird. Die Auswahl ist relevant bei der Ansteuerung der OLED Displays sowie der Anpassung der Kommunikationsschnittstelle bei einer Nutzung des DUE über den nativen USB Anschluss.

4.2.2.4. Datencontainer (Data Variables)

```
68 Datenfeld datenfeld[]=
69 {
70     //Description ID      DT      OT      target  Ref2  Ref3  Ref4  Ref5  RQ  IV
71     { "RTRIM", "1370", 'f', 60, 0, 0, 0, 0, 0, "", "0.0"} //Example
72     , { "PTRIM", "1360", 'f', 60, 1, 0, 0, 0, 0, "", "0.0"} //Example
73 };
74 const int VARIABLENANZAHL = sizeof(datenfeld)/sizeof(datenfeld[0]);
```

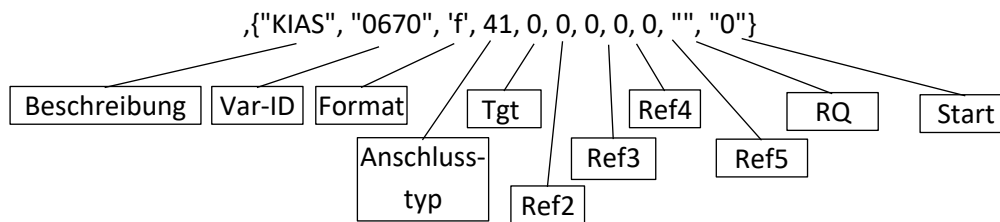
Abbildung 9: Datencontainer

Dies ist wichtigste Einstellung, die von euch vorgenommen werden muss. In diesem Bereich wird definiert, welche Daten der SharedMemory vonBMS auf dem Arduino genutzt werden sollen und welche am Arduino angeschlossene Hardware diese Daten zur Anzeige bringt. Hier können theoretisch bis zu 98 Datenvariablen hinzugefügt werden. Bei einer großen Zahl von Variablen kann es aber zu Problemen kommen, da die Rechenkapazität des Arduino begrenzt

ist und es zu Störungen oder Verzögerungen bei der Anzeige kommen kann. Ich habe bislang nicht feststellen können, wo die Grenze des Machbaren liegt.

Wichtig: Bitte das Komma am Anfang der zweiten bis zur letzten Zeile nicht übersehen! Nur in der ersten Zeile des Datencontainers ist dies nicht vorhanden.

Erläuterung der einzelnen Positionen einer Zeile des Datencontainers:



1. Beschreibung

Diese Eingabe hat nur informatorischen Charakter. Die Angabe kann aber genutzt werden, um neben dem Wert auch die Variablenbezeichnung bspw. auf einem LCD-Display mit anzugeben. (max. 5 Zeichen)

2. Variablen-ID

Dieser Eintrag wird nur benötigt, wenn man plant den Datenaustausch nach dem PULL-Prinzip durchzuführen. Die ID wird vom Windows-Programm benutzt, um den richtigen Datensatz aus der SharedMem zu identifizieren.

Anzugeben ist die 4-stellige Zahl (ggf. führende Nullen ergänzen), die der ID aus der Variablenliste BMSAIT-Variablen.csv entspricht.

3. Format

Dieser Eintrag wird nur benötigt, wenn man plant den Datenaustausch nach dem PULL-Prinzip durchzuführen.

Anzugeben ist ein Schlüssel für das Datenformat der gewünschten Datenvariablen, die auch in der BMSAIT-Variablen.csv angegeben ist.

4. Anschlusstyp

Hiermit wird festgelegt, welches angeschlossene Gerät genutzt werden soll, um den Inhalt dieser Datenvariablen anzuzeigen.

- 10-LED (PIN ist Anode und wird über die LED mit GND verbunden)
- 11-LED (PIN ist Kathode und wird über die LED mit V_{cc} verbunden)
- 12-LEDMatrix (Viele LED teilen sich die SteuerPins des Arduino)
- 20-LCD
- 30-7Segment Max7219
- 31-7Segment TM1367
- 32- DotMatrix Modul SLx2016
- 40-Servo einzeln
- 41-Servo über pwm motor shield
- 50-Stepper Motor (28BYJ-48 oder vergleichbar)
- 51-Stepper Motor (X27.168 direkt am Arduino)
- 52-Stepper Motor (X27.168 über Motorcontroller)
- 53-Stepper Motor mit Kompass-Funktion

60-Motorpotentiometer
70-OLED
71-OLED mit Speedbrake Indicator Funktion
72-OLED mit Fuel Flow Indicator Funktion
80-Ansteuerung eines Backlighting

5. Tgt

Die Nutzung dieses Feldes ist vom Anschlusstyp abhängig.

Beim Anschlusstyp 11 (LED) wird hier angegeben, an welchem PIN die LED angeschlossen ist, auf dem der (Bool-)Wert angezeigt werden soll.

Bei der Nutzung von Motoren (Typ 40,41,50,51,52,60) wird dieses Feld genutzt, um einen Verweis auf einen Eintrag der Motorliste herzustellen. In den Motorlisten der jeweiligen Module finden sich Steuerungsinformationen zu jedem Motor (siehe 4.2.12).

Hinweis: Bitte aufpassen, dass ihr hier nicht gleich den PIN des Motors angebt. Diese Info gehört in die Motorliste des entsprechenden Moduls.

Bei der Anzeige auf einem LCD-Screen wird mit diesem Wert die Zeile festgelegt, in die der Wert geschrieben werden soll.

6. Referenz 2

Die Nutzung dieses Feldes ist vom Anschlusstyp abhängig.

Bei der Ansteuerung einer LED wird über diesen Bereich bspw. die Leuchtstärke der LED festgelegt.

7. Referenz 3

Die Nutzung dieses Feldes ist vom Anschlusstyp abhängig.

Hier wird bspw. bei LCD festgelegt, wie viele Zeichen des Datensatzes zur Angezeigt gebracht werden sollen. Dies gewährleistet bspw. auf LCD-Anzeigen oder bei 7-Segment-Tubes die korrekte Positionierung des Datensatzes. Ist der Datensatz länger als die hier festgelegte Zahl x, werden nur die ersten x Zeichen der Datenvariablen angezeigt.

8. Referenz 4

Die Nutzung dieses Feldes ist vom Anschlusstyp abhängig.

Hier wird bspw. festgelegt, ob der Datensatz linksbündig auf einem LCD-Display oder einem 7-Segment-Tube erscheint oder eingerückt. Die hier angegebene Zahl rückt den Text auf dem Ausgabegerät um x Stellen nach rechts.

9. Referenz 5

Hiermit kann festgelegt werden, dass auf 7-Segment-Tubes an der hier angegebenen Stelle ein Dezimalpunkt gesetzt wird.

10. RQ

Hierbei handelt es sich um einen Platzhalter, in der die Arduinos beim PULL Prinzip das Kommando einer Datenanfrage vorbereiten.

11. Startwert

Der hier eingegebene Wert gibt den Standardwert an, der beim Einschalten des Arduino angezeigt werden soll.

4.2.3. Das Modul Switches

Wenn dieses Modul aktiviert wird und in der Konfiguration Schalter definiert wurden, wird kontinuierlich geprüft, ob Taster/Schalter betätigt werden. Wenn dies passiert, werden Kommandos an den PC gesendet. In der Windows-App kann definiert werden, welche Tastatur-/Joysticksignale bei Vorliegen eines Kommandos ausgelöst werden soll. (siehe 3.2.5)

Die Programmierung dieses Moduls geht davon aus, dass alle Schalter den Arduino PIN mit GND verbunden werden.

In dem Modul können zwei Typen von Schaltern eingerichtet werden:

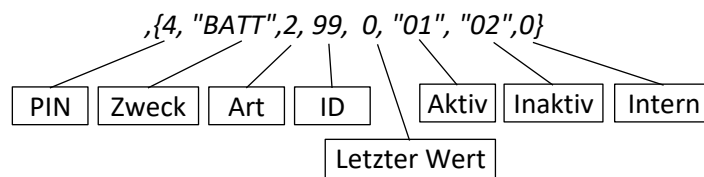
4.2.3.1. Digitale Taster/Kippschalter

Es können beliebig viele Schalter mit dem Arduino (begrenzt durch die Anzahl der PINs des Arduino) verbunden werden.

In dem Datenblock ist für jeden zu lesenden Schalter eine Zeile einzurichten.

```
28//Switch definition. If you add a switch, add a line to the following list
29Switch switches[]=
30{
31 // <PIN>,<description>,<type>,<rotarySwitchID>, 0, <commandID when pressed>,<commandID when released>,<internal command>
32 {4, "BUP", 2, 0, 0, "01", "02", 0} // DigitalBUP - Backup / Off
33 {5, "AFLAP", 2, 0, 0, "03", "04", 0} // AltFlaps - Extend / Norm
```

Format einer Zeile für den Schalter:



Erläuterung der einzelnen Positionen der Zeile:

1. PIN

Der PIN, an dem der Schalter angeschlossen ist Dieser Wert wird benutzt, um den PIN zu identifizieren, an dem nach veränderten Werten gesucht wird.

2. Zweck

Eine kurze Beschreibung des Zwecks des Schalters. Kein besonderer Zweck, dient nur der Übersichtlichkeit.

3. Art

Die Art des Schalters. Hiermit wird gesteuert, wie der Schalter ausgelesen wird.

- 1 - digitaler Taster,
- 2 - digitaler Kippschalter,
- 3 - analog zu lesender Drehschalter

4. ID

Die Identifikationsnummer wird benötigt, um bei mehreren analog zu lesenden Drehschaltern den jeweiligen Kommandoblock ansprechen zu können.

5. Letzter Wert

Platzhalter für den letzten gemessener Wert (Standard: 0). Wird zur Laufzeit benötigt. Keine Definition erforderlich.

6. Aktiv

Hier wird definiert, welches Kommando an die Windows-App gesendet wird, wenn ein Schalter gerade betätigt wird (Spannung geht von 1 auf 0). Bitte hier „00“ setzen, wenn kein Kommando gesendet werden soll.

7. Inaktiv

Hier wird definiert, welches Kommando an die Windows-App gesendet wird, wenn ein Schalter gelöst wird (Spannung geht von 0 auf 1). Bitte hier „00“ setzen, wenn kein Kommando gesendet werden soll.

8. Intern

Platzhalter für Arduino-interne Kommandos (1..199). Wenn das Tastensignal benutzt werden soll, um Arduino-interne Dinge zu steuern, kann hier ein Wert gespeichert werden, der später verarbeitet werden kann. (Standard: 0) (dies wird im komplexen Beispiel BUPRadio angewendet).

Interne Systemkommandos:

Wie eben beschrieben kann einem Button oder Schalter ein internes Kommando hinterlegt werden.

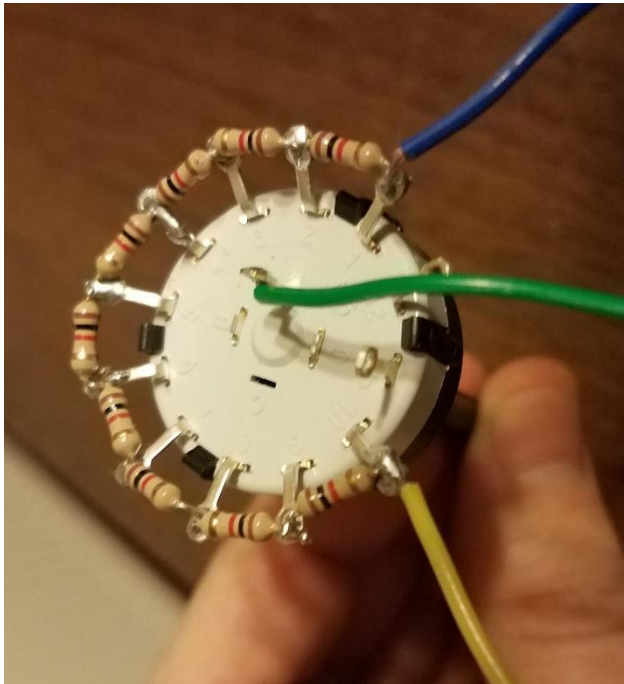
Hier gibt es aktuell drei besondere Kommandos, die zudem als Systemkommando vom Arduino an die BMSAIT Windows App versendet werden, um Synchronisierungen auszulösen (siehe Kapitel 2.2).

253: Wird ein Button gedrückt, dem das interne Systemkommando 253 zugeordnet ist, wird eine Schaltersynchronisierung ausgelöst.

254: Wird ein Button gedrückt, dem das interne Systemkommando 254 zugeordnet ist, werden alle Steppermotoren in die Nullstellung gefahren.

255: Wird ein Button gedrückt, dem das interne Systemkommando 255 zugeordnet ist, werden alle Steppermotoren kalibriert.

4.2.3.2. Digitale Drehschalter mit analoger Auswertung



Hierbei handelt es sich um eine Besonderheit. Um einen digitalen Drehschalter über einen Arduino auszulesen, wird i.d.R. jede einzelne Rastung des Drehschalters mit einem PIN des Arduino verbunden. Dies kann schnell alle verfügbaren PINs eines Arduino aufbrauchen. Mit der hier beschriebenen Vorgehensweise wird eine Möglichkeit eröffnet, einen Drehschalter nicht digital, sondern analog auszulesen, um Drehschalter mit beliebig vielen Rastungen über nur einen einzelnen (analogen!) PIN des Arduino auszulesen.

Wird der Drehschalter wie dargestellt verkabelt, wird mit Bewegung des Drehschalters beeinflusst, wie vielen Widerständen ein Steuersignal (grün) mit der Spannungsversorgung (gelb) trennt. Dies führt

zu unterschiedlichen Spannungen, die von einem analogen Port des Arduino gelesen und unterschieden werden können. Je nach gemessener Spannung kann darüber bestimmt werden, in welcher Stellung sich der Drehschalter aktuell befindet. Der gemessene Wert wird vom Arduino in Form einer Zahl (0..1024) ausgegeben.

Damit dies funktioniert, muss für jeden angeschlossenen Drehschalter eine Steuerungstabelle nach folgendem Vorbild angelegt werden:

```
//define commands for analog readings of a rotary switch or poti. This is an example for a 10-position switch
//{<CommandID>,<externalCommand>,<low threshold>,<high threshold>}
Drehschalter analogSchalter[1][12]=
{
  { "04",true,0,64}           //send command 04 if analog read is between 0 and 64 (of 1024)
  ,{"05",true,65,177}        //send command 05 if analog read is between 65 and 177 (of 1024)
  ,{"06",true,178,291}        //send command 06 if analog read is between 178 and 291 (of 1024)
  ,{"07",true,292,405}        //send command 07 if analog read is between 292 and 405 (of 1024)
  ,{"08",true,406,518}        //send command 08 if analog read is between 406 and 518 (of 1024)
  ,{"09",true,519,632}        //send command 09 if analog read is between 519 and 632 (of 1024)
  ,{"10",true,633,746}        //send command 10 if analog read is between 633 and 746 (of 1024)
  ,{"11",true,747,859}        //send command 11 if analog read is between 747 and 859 (of 1024)
  ,{"12",true,860,973}        //send command 12 if analog read is between 860 and 973 (of 1024)
  ,{"13",true,974,1024}       //send command 13 if analog read is between 974 and 1024 (of 1024)
}
};
```



1. Kommando

Hier wird definiert, welches Kommando an die Windows-App gesendet wird, wenn die aktuelle Rastung eingenommen wird (Analogwert liegt zwischen unterer und oberer Grenze dieser Zeile). Bitte hier „00“ setzen, wenn kein Kommando gesendet werden soll.

2. Extern

Hier ist „true“ anzugeben, wenn das zuvor definierte Kommando an die Windows-App gesendet werden soll, wenn die aktuelle Rastung eingenommen wird. Wird hier „false“ gesetzt, wird das Kommando nicht an die Windows-App versendet. Dies kann sinnvoll sein, wenn das Signal nur innerhalb des Arduino verarbeitet werden soll.

3. Untere Grenze

Wird ein Schalter analog ausgelesen, wird ein Wert zwischen 0 (Spannung am PIN entspricht der Versorgungsspannung) und 1024 (Spannung am PIN entspricht der Masse) ausgegeben. Um Toleranzen zuzulassen, ist der Bereich zwischen 0 und 1024 in so viele Bereiche einzuteilen, wie der Drehschalter über Rastungen verfügt. Hier wird die untere Grenze des Toleranzbereiches für die aktuelle Rastung angegeben. Der Wert sollte genau an die Obergrenze des vorherigen Toleranzbereiches anschließen.

4. Obere Grenze

Wird ein Schalter analog ausgelesen, wird ein Wert zwischen 0 (Spannung am PIN entspricht der Versorgungsspannung) und 1024 (Spannung am PIN entspricht der Masse) ausgegeben. Um Toleranzen zuzulassen, ist der Bereich zwischen 0 und 1024 in so viele Bereiche einzuteilen, wie der Drehschalter über Rastungen verfügt. Hier wird die obere Grenze des Toleranzbereiches für die aktuelle Rastung angegeben. Der Wert sollte genau an die Untergrenze des nächsten Toleranzbereiches anschließen.

4.2.4. Das Modul ButtonMatrix

Soll ein Arduino digitale Schalter auslesen, kommt das Arduino Board durch die begrenzte Zahl der verfügbaren PINs recht schnell an seine Grenzen. Eine Lösung besteht darin, dass nicht jeder Schalter an einem eigenen PIN angeschlossen, sondern eine Matrix verwendet wird.¹ Die Schalter schließen dabei jeweils eine Kombination zweier digitaler Ausgangs-PINs des Arduino kurz. Beispielsweise wird bei verfügbaren 20 PINs die Zahl der ansteuerbaren Schalter/Taster am Arduino von 20 (direkte Ansteuerung) auf 100 (Matrixanordnung) erhöht.

Werden nur Taster angeschlossen, ist die Verkabelung relativ trivial. Der Grund dafür ist, dass bei Tastern immer nur ein Kontakt der Matrix geschlossen wird und der gedrückte Taster so eindeutig identifiziert werden kann.

Sollen neben Tastern aber auch Kippschalter angeschlossen werden - was dazu führt, dass mehrere Kontakte in der Matrix gleichzeitig geschlossen sind - sind weitere Vorkehrungen erforderlich, um die aktivierten Schalter richtig zu bestimmen. Die Vorkehrungen bestehen darin, dass vor jedem Taster/Schalter eine Diode verbaut werden muss.

¹ Zur Vertiefung siehe <https://www-user.tu.chemnitz.de/~heha/Mikrocontroller/Tastenmatrix.htm>

In dem Arduino Modul müssen folgende Einstellungen vorgenommen werden:

In dem Modul sind die PINs zu definieren, die als Spalten/Reihen der Schaltermatrix dienen sollen. Hinterlegt zwischen den geschweiften Klammern die PINs des Arduino, an denen Ihr bei Euch die Schalter verbinden wollt.

```
byte rows[] = {2,3,4,5,6,7,8,9};
const int rowCount = sizeof(rows)/sizeof(rows[0]);

byte cols[] = {10,11,12,14,15};
const int colCount = sizeof(cols)/sizeof(cols[0]);
```

Zudem müsst ihr hier eine Tabelle hinterlegen, in der die Signale angegeben werden, die der Arduino bei Registrierung eines gedrückten Tasters/Schalters an die BMSAIT Windows-App senden soll. Bitte beachtet, dass die Zahl der Spalten (col) und Reihen (row) mit der Zahl der PINs übereinstimmen muss, die ihr im vorherigen Schritt hinterlegt habt.

```
byte keysignal[colCount][rowCount]=
{
// row1 row2 row3 row4 row5 row6 row7 row8
{ 1, 2, 3, 4, 5, 6, 7, 8, } //col 1
,{ 9, 10, 11, 12, 13, 14, 15, 16, } //col 2
,{ 17, 18, 19, 20, 21, 22, 23, 24, } //col 3
,{ 25, 26, 27, 28, 29, 30, 31, 32, } //col 4
,{ 33, 34, 35, 36, 37, 38, 39, 40, } //col 5
};
```

4.2.5. Das Modul Encoder

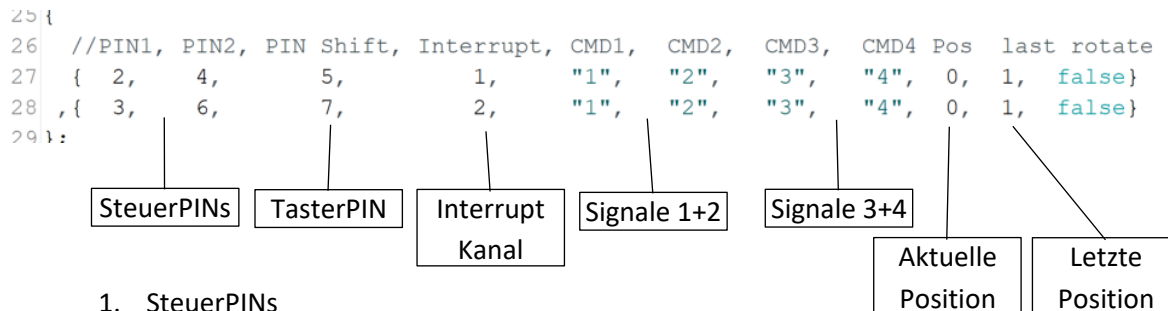
Dieses Modul ermöglicht das Ansteuern von einem oder zwei Drehimpulsgebern, deren Bewegungen als Tastatur- oder Joystickssignale weitergegeben werden. Dies kann beispielsweise für die Einstellung des Höhenmessers oder der CRS/HDG Drehknöpfe des HSI benutzt werden.

Das Modul ist zudem in der Lage, einen Druckknopf auszulesen, der bei einigen Drehimpulsgebern mit verbaut ist. Dies kann für eine „shift“ Funktion genutzt werden, um so über einen Drehimpulsgeber vier verschiedene Signale auszulösen.

Der Drehimpulsgeber verfügt in der Regel über drei PINs. Prüft das Datasheet eures Schalters, um die PIN-Belegung festzustellen. Üblicherweise ist der mittlere PIN ist mit GND zu verbinden, die beiden äußeren PIN mit einem DatenPIN des Arduino. Dieses Modul arbeitet mit einer speziellen Routine des Arduino, die ein extrem schnelles Reagieren auf eine Bewegung des Encoders ermöglicht (Interrupt) und damit ein Höchstmaß an Zuverlässigkeit gewährleistet. Die Fähigkeit zur Nutzung von Interrupts ist nur bei wenigen PINs des Arduino vorhanden. Jeder Encoder muss daher an einen dieser PINs angeschlossen werden.

UNO	2, 3
NANO	2, 3
LEONARDO	0, 1, 2, 3, 7
MICRO	0, 1, 2, 3, 7
MEGA	2, 3, 18, 19, 20, 21

Sollte der Drehimpulsgeber über eine Tasterfunktion verfügen, finden sich zwei weitere PINs am Encoder. Wenn die Funktion genutzt werden soll, ist ein Kontakt mit einem DatenPIN des Arduino und der andere mit GND zu verbinden.



1. SteuerPINs

Hier sind die beiden PIN anzugeben, mit denen die Drehbewegung des Drehimpulsgebers abgelesen wird (die beiden äußeren Pin der Dreiergruppe).

2. TasterPIN

Hier ist der PIN anzugeben, an dem die Tasterfunktion des Encoders verbunden ist.

3. Signale 1+2

Hier wird definiert, welches Kommando an die BMSAIT Windows App gesendet wird, wenn der Encoder sich nach links/rechts bewegt und der Taster nicht gedrückt ist.

4. Signale 3+4

Hier wird definiert, welches Kommando an die BMSAIT Windows App gesendet wird, wenn der Encoder sich nach links/rechts bewegt und der Taster gedrückt ist.

4.2.6. Das Modul Analogchse



Dieses Modul ermöglicht es, ein angeschlossenes Potentiometer auszulesen und über die BMSAIT Windows-App und der vJoy Software als Analogachse eines Joysticks abzubilden. Dies kann zur Ansteuerung einer Analogachse in BMS (z.B. TRIM, Lautstärkeregelung usw.) genutzt werden.

Hierzu müssen in dem Modul die Analog-PINs des Arduino benannt werden, die ausgelesen werden sollen:

```

AAchse analogaxis[] = {
// PIN Command Value
{ A0, 2, 0 }
}

```

1. PIN

Hier ist der PIN anzugeben, an dem das Signal des Potentiometers abgenommen wird. Bitte beachtet, dass hier nur die PINs des Arduino genutzt werden können, die über eine analoge Auswertefähigkeit verfügen (A0, A1...Ax).

2. Command

Um eine Analogachse im vJoy anzusteuern, ist in der BMSAIT WinApp ein Kommando für die Analogachse zu definieren. Hier ist die dabei vergebene KommandoID einzugeben, damit die Analogdaten mit der richtigen vJoy Achse verbunden wird.

3. Value

Wird zur Laufzeit benötigt, um den letzten gelesenen Wert zu speichern. Keine Änderung erforderlich.

Das analoge Lesen eines PINs ist relativ genau, eine 100%ige Präzision ist aber nicht gewährleistet. Der Arduino unterteilt die Spannbreite eines angeschlossenen Potentiometers in 1024 Stellungen. Es ist möglich, dass das Signal an einem analogen PIN durch Umwelteinflüsse leicht schwankt. Um nicht bei jeder Schwankung ein (unnötiges) Signal auszulösen, wird hier zudem ein Pufferwert definiert. Ein Signal wird erst dann ausgelöst, wenn sich das Signal den Pufferwert überschreitet. Ich empfehle den Wert zwischen 3 und 5 einzustellen.

```
// This module allows to read analog inputs and send changes to the F4SME App
#define ATH 3 //Analog Threshold. A change of the analog value will only be co
```

Wird eine Bewegung des Potentiometers registriert, wird der gelesene Wert zur weiteren Verarbeitung an die Windows-App gesendet und dort verarbeitet (siehe 3.2.5).

4.2.7. Das Modul LED



In diesem Modul befindet sich die Funktion zum Ein- und Ausschalten von LED. Die Funktionen können für einfache LED Lösungen eingesetzt werden, bei denen ein Arduino eine überschaubare Anzahl an LEDs steuern soll (1 LED je PIN; Der Strom fließt über einen Output-PIN durch die LED zur Masse).

Das Modul verarbeitet Datenvariablen der Typen 10 und 11.

Der Typ 10 ist zu nutzen, wenn der PIN als Spannungsversorgung für die LED dient (PIN → LED → Masse).

Der Typ 11 ist zu nutzen, wenn die PIN so angeschlossen wurde, dass der PIN als Masse arbeitet. (Spannungsversorgung → LED → PIN).

Einfache Ansteuerung

In der einfachsten Form der Ansteuerung wird lediglich geprüft, ob der der LED zugeordnete Datenwert den Wert „T“(rue) enthält. Wenn ja, wird die LED aktiviert. In allen anderen Fällen wird die LED abgeschaltet. Weitere Einstellungen sind nicht erforderlich.

Led mit angepasster Leuchtstärke

Es ist möglich die Helligkeit einer LED in der Programmierung des Arduino festzulegen. Eine Änderung der Leuchtstärke über die BMSAIT App ist derzeit nicht möglich.

Arduino UNO:	3, 5, 6, 9 – 11
Arduino MEGA:	2 – 13, 44 – 46
Arduino Leonardo, Micro	3, 5, 6, 9, 10, 11, 13

Um die Leuchtstärke festzulegen, ist es erforderlich die LED an einen der pwm-PINs des Arduino anzuschließen.

Die Leuchtkraft der LED wird in dem Modul UserConfig im Datenfeld vorgegeben. In der Zeile mit der Datenvariable, die zu der LED gehört, ist in Spalte Ref2 die Leuchtkraft in einem Bereich von 0 (aus) bis 255 (volle Leuchtkraft) anzugeben.

```

70 Datenfeld datenfeld[] =
71 {
72   //Description ID   DT   OT   target   Ref2 Ref3 Ref4 Ref5 RQ   IV
73   {"RGear", "1599", 'b', 10, 2, 0, , , 0, 0, "", "False"} //Example Variable 0 - Right Gear Safe (brightness 0%)
74   {"NGear", "1597", 'b', 10, 3, 128, , , 0, 0, "", "False"} //Example Variable 1 - Nose Gear Safe (brightness 50%)
75   {"LGear", "1598", 'b', 10, 4, 255, , , 0, 0, "", "False"} //Example Variable 2 - Left Gear Safe (brightness 100%)
76   {"UGear", "1571", 'b', 10, 5, 178, , , 0, 0, "", "False"} //Example Variable 3 - Gear Unsafe (brightness 75%)
77 };
78 const byte VARIABLENANZAHL = sizeof(datenfeld)/sizeof(datenfeld[0]);

```

Blinkende LED

Im Simulator sind einige LED vorhanden, die in bestimmten Situation blinken (bspw. das JetFuelStarter Licht). Die Information, ob eine LED an/aus ist und ob sie blinkt, wird in der SharedMem in unterschiedlichen Bereichen gespeichert. Die BMSAIT App fügt diese Informationen bereits zusammen und überträgt einen kombinierten Status. Das ist auch der Grund dafür, dass einige LED als Boolean Wert und einige als Byte Wert in der Variablentabelle zu finden sind.

Blinkende LED werden von der BMSAIT App als Byte Wert übertragen, damit der Arduino erkennt, ob die LED aus (0) oder an (1) ist bzw. ob sie langsam (3) oder schnell (4) blinkt. Der Arduino wird dezentral ein Blinken der LED veranlassen. Wenn das Blinken zu schnell oder zu langsam ist, kann dies in der Programmierung des Arduino über die Definition BLINKSPEED angepasst werden.

```

7#define BLINKSPEED 500 //pause (in ms) between on/off for fast blinking. Slow blinking will be 50%

```

4.2.8. Das Modul LED-Matrix



In diesem Modul befindet sich die Funktion zur Ansteuerung einer so großen Zahl von LED, bei der es aufgrund der begrenzten Zahl der PINs des Arduino nicht mehr möglich ist jede LED an einen eigenen Steuerungs-PIN anzuschließen.

Die Lösung besteht darin, die LED als Matrix anzuordnen. Hierfür empfehle ich einen Controller zu verwenden, der billig zu bekommen ist und die Ansteuerung stark vereinfacht. Gängig ist hier der MAX7219 Chip. Der MAX7219 Controller wird über die SPI Schnittstelle mit dem Arduino verbunden. Die LEDs werden mit den Ausgangsanschlüssen des MAX7219 verbunden.

Im Modul selbst müsst ihr nur die drei PINs angeben, mit denen ihr den Max7219 verbindet.

```
#define LEDM_CLK 8 //PIN "Clock" for the SPI connection of the LED-Matr
#define LEDM_CS 9 //PIN "Cable Select" for the SPI connection of the I
#define LEDM_DIN 10 //PIN "Data In" for the SPI connection of the LED-Ma
#define LEDM_BRIGHTNESS 5 //sets the intensity of the LED-Matrix
```

Mit der Einstellung LEDM_Brightness könnt ihr die Helligkeit beeinflussen. Der Wert muss zwischen 0 und 15 liegen

Die Defintion der LED erfolgt im Modul User-Konfiguration bei der Angabe der entsprechenden Variablen.

Ref1 – Nummer des MAX7219 chip (falls ihr mehrere verwendet)

Ref2 – Spalte, in der eine LED beleuchtet werden soll

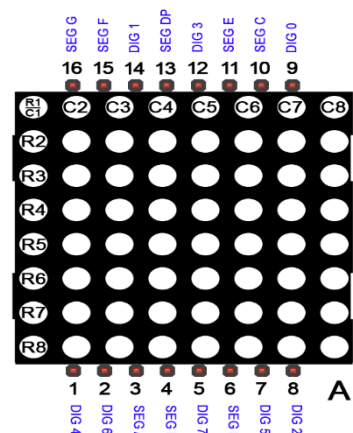
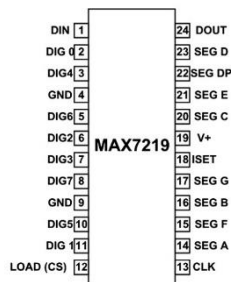
Ref3 – Zeile, in der eine LED beleuchtet werden soll

Beispiel Variablendefinition für eine LED in einer LEDMatrix:

```
{"ENGFI", "1506", 'b', 12, 1, 2, 4, 0, 0, "F"}
```

Diese Variablendefinition würde das Engine Fire Warnlicht (ID 1506) auf dem ersten Max7219 Modul auf der in der 2. Spalte und 4. Zeile angeschlossenen LED ausgeben.

Es ist durchaus möglich den MAX7219 Chip einzeln zu kaufen und anzuschließen. Es gibt aber auch fertig verlötete Platinen mit einem MAX7219 Chip, über den ein 8x8 LED Matrix angesteuert wird. Wenn ihr darauf achtet, dass sich das Modul mit den 8x8 LED einfach von der Platine mit dem Controllerchip abziehen lässt, könnt ihr eure LED ohne große Lötarbeit gleich an den freigewordenen Anschlüssen des Chips anschließen.



Die Bezeichnung der Grafik ist dabei so zu verstehen, dass die PINs der Segmente (SEG) die Reihen ansteuern und die PINs der Digits (DIG) die Spalten. Um also beispielsweise die 4. LED der 2. Reihe zum Leuchten zu bringen, muss an der 8x8 Matrix eine Spannung zwischen PIN 4 (SegB=2.Reihe) und PIN 12 (DIG3=4.Spalte) angelegt werden.

Die LED sind dabei so auszurichten, dass die Anode (+) an die Ausgangsanschlüsse für die Reihe (Seg) und die Kathode (-) an die Spalten (Dig) angeschlossen werden.

Auch im Modul LEDMatrix können LEDs zum Blinken gebracht werden. Die Ansteuerung funktioniert wie auch beim Modul LED über einen Eintrag im Datenfeld des Moduls UserConfig.

4.2.9. Das Modul LCD



Mit Modul kann ein alphanumerischer Wert einer Datenvariablen an ein LCD-Display ausgegeben werden. In dem vorliegenden Coding wird hier ein 16x2 Display (HD44780 Standard) und Ansteuerung über ein I²C -Modul erwartet.

Im Modul kann die Zahl der Zeilen und Zeichen je Zeile verändert werden, falls man einen andere Displaygröße verwendet (z.B. 20x4).

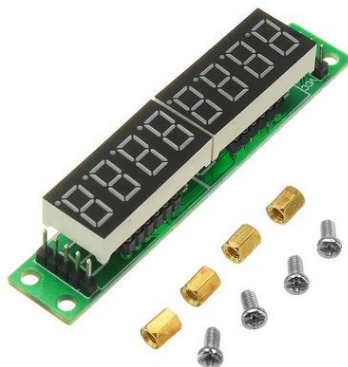
Die Verkabelung des Displays muss an den I²C PINs des jeweiligen Arduino Boards erfolgen. Da diese PINs in der Hardware des Arduinos vorgegeben sind, ist eine Einstellung der PINs in dem Arduino-Sketch nicht erforderlich.

Board	SDA	SCL
Uno	A4	A5
Micro	2	3
Nano	A4	A5
Mega	20	21
Leonardo	2	3
Due	20	21

Es ist durchaus möglich, dass mehrere Datenvariablen gleichzeitig auf einem LCD-Display angezeigt werden. Hierzu sind die Formatierungsmöglichkeiten im Datencontainer zu nutzen, um bspw. zwei Datenvariablen auf dem gleichen Display in unterschiedlichen Zeilen anzuzeigen (siehe 4.2.2 / Ziel).

Es ist möglich mehrere Displays an einem Arduino zu betreiben. Hierfür ist bei den Codezeilen mit dem Wert „lcd[1]“ die Kommentierung zu entfernen (löschen der „//“ am Anfang). Beim Betrieb mehrerer I²C Geräte ist zudem die Hardwareadressierung der Geräte von Bedeutung, die am LCD Display über eine Lötbrücke festgelegt werden muss.

4.2.10. Das Modul SSegMAX7219



Mit diesem Modul kann ein numerischer Wert einer Datenvariablen auf einer 8-Bit 7-Segment-Display Tube mit SPI Anschluss (MAX7219 Chip) ausgegeben werden.

In dem Modul muss die PIN-Belegung festgelegt werden. Das MAX7219-Display wird über 5 Kabel mit dem Arduino verbunden. Die beiden Kabel für die Spannungsversorgung sind selbsterklärend. Für die Datenverbindung sind drei Leitungen erforderlich. Hier ist anzugeben, an welchen PINs des Arduino die Leitungen für „Clock“, „Cable Select“ und „Data In“ des Displays verbunden wurden.

```
#define MAX_CLK 8 //PIN "Clock" for the SPI connection of the 7-Segment Tube
#define MAX_CS 9 //PIN "Cable Select" for the SPI connection of the 7-Segment Tube
#define MAX_DIN 10 //PIN "Data In" for the SPI connection of the 7-Segment Tube
```


Die Helligkeit der Anzeige der 7-Segment Anzeige kann bei der Programmierung des Arduino über die Option MAX_BRIGHTNESS festgelegt werden. Der Wert muss zwischen 0 (aus) und 15 (volle Leuchtstärke) liegen.

Es ist durchaus möglich, dass mehrere Datenvariablen gleichzeitig auf einer Tube angezeigt werden. Hierzu sind die Formatierungsmöglichkeiten im Datencontainer zu nutzen, um bspw. einen Wert auf den ersten vier Zeichen und eine andere Variable auf den letzten vier Zeichen anzuzeigen (siehe 4.2.2 / Zeichenzahl und Startposition).

Hinweis: Um den Dezimalpunkt zu setzen (bspw. Bei der Anzeige einer Funkfrequenz) ist die entsprechende Stelle bei der Datenvariable anzugeben (siehe 4.2.2 / Dezimalpunkt).

4.2.11. Das Modul SSegTM1367



Mit diesem Modul kann ein numerischer Wert einer Datenvariablen auf einem 3 bis 6-stelligen 7-Segment-Display mit TM1637 Steuerungschip ausgegeben werden.

Auch wenn die Ansteuerung des TM1637 über nur zwei PINs (DIO CLK) erfolgt, handelt es sich nicht über eine I²C Verbindung. Das bedeutet einerseits, dass der Anschluss nicht zwingend an den hardwareseitig vorgegebenen I²C

PINs des Arduino erfolgen muss. Andererseits ist es nicht möglich dieses Display mit weiteren Geräten gemeinsam über die I2C PINs zu betreiben. Es muss angegeben werden, mit welchen PINs des Arduino die beiden Datenleitungen des Displays verbunden wurden.

```
// Call 7-Segment-Display with TM1637 controller
#define TM1637_CLK 2
#define TM1637_DIO 3
```

Es ist durchaus möglich, dass mehrere Datenvariablen gleichzeitig auf einem Display angezeigt werden. Hierzu sind die Formatierungsmöglichkeiten im Datencontainer zu nutzen, um bspw. einen Wert auf den vorderen zwei Zeichen und eine andere Variable auf den hinteren zwei Zeichen anzuzeigen (siehe 4.2.2 / Zeichenzahl und Startposition).

Hinweis: Um den Dezimalpunkt zu setzen (bspw. Anzeige Funkfrequenz) ist die entsprechende Stelle bei der Datenvariable anzugeben (siehe 4.2.2 / Dezimalpunkt).

Beispiel Steuerung eines 6-Digit Displays über die Variablendeklaration:

Wert	Target	Ref3 (Anzahl)	Ref4(Offset)	Ref5(Punkt)	Resultat
Var1: 1234	0	4	0	99	1234__
Var1: 1234	0	4	2	99	__1234
Var1: 1234	0	2	3	1	__1.2__
Var1: 12	0	2	0	1	1.2__34.
Var2: 34	0	2	4	2	

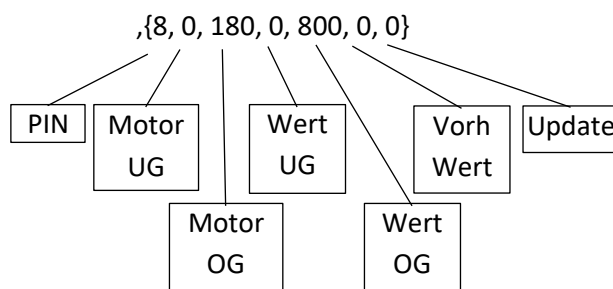
4.2.12. Das Modul Servo



Dieses Modul dient zur Ansteuerung von einem oder mehreren Servo-Motoren, die direkt mit dem Arduino verbunden sind.

Wichtig: Wenn eine Datenvariable an einem Motor ausgegeben werden soll, ist im Datencontainer des Moduls UserConfig nicht der PIN anzugeben, an dem sich der Motor befindet, sondern die Zeile in dem Datencontainer des Moduls Servo. In dem Servo-Datencontainer finden sich neben dem PIN, an dem der Motor angeschlossen ist, auch weitere Daten, die für die Funktion des Motors erforderlich sind.

Um einen Servomotor anzusteuern, sind mehrere Steuerungsinformationen festzulegen. Diese sind in dem Datencontainer im Modul Servo zu hinterlegen.



5. PIN

Der PIN des Arduino, an dem der Servo-Motor angeschlossen wurde.

6. Motor UG

Der Minimalwert, den der Servo annehmen soll (0-180°).

7. Motor OG

Der Maximalwert, den der Servo annehmen soll (0-180°).

8. Wert UG

Der niedrigste Wert, den die Datenvariable annehmen kann (Zahl).

9. Wert OG

Der höchste Wert, den die Datenvariable annehmen kann (Zahl).

10. Vorh. Wert

Platzhalter für den letzten gemessener Wert (Standard: 0). Wird zur Laufzeit benötigt. Keine Definition erforderlich.

11. Update

Platzhalter für den Zeitpunkt des letzten Steuersignals an den Servo (Standard: 0). Wird zur Laufzeit benötigt. Keine Definition erforderlich.

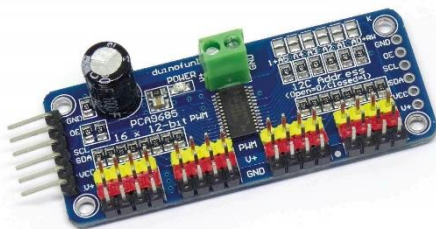
Jeder Motor ist unterschiedlich und es gibt selbst bei baugleichen Servos Fertigungstoleranzen, weshalb eine Kalibrierung erforderlich ist. Bei der Kalibrierung des Arduino (siehe 3.2.1) wird der

Motor seine Minimalposition und Maximalposition anfahren und dort jeweils eine Sekunde verbleiben. Prüft in dieser Zeit, ob der Arduino dort fühlbar oder hörbar vibriert. Wenn ja, dann versucht der Servo eine Position einzunehmen, die aufgrund der Hardwarebeschränkungen nicht erreicht werden kann. Dies wird den Servo wohl nicht beschädigen, sollte aber grundsätzlich vermieden werden. Die Werte Motor_UG und Motor_OG sind daher bei jedem angeschlossenen Motor zu prüfen und ggf. anzupassen.

Einen Motor über Arduino anzusteuern ist grundsätzlich kein Problem. Die Bewegungen mit einem Fluginstrument aus dem Flugsimulator zu synchronisieren, ist aber eine größere Aufgabe. Servomotoren haben den Nachteil, dass der Motor sich nicht frei bewegen kann, sondern nur über einen geringen Bewegungsspielraum für einen Arm/Zeiger verfügt (i.d.R. 180°). Der gewaltige Vorteil des Servos liegt aber darin, dass der Arm/Zeiger absolut angesteuert werden kann, d.h. man muss die vorherige Position des Zeigers nicht kennen, um ihn in eine gewünschte Position zu bringen (→bewege dich auf Position 90°).

Die Funktionsweise des Moduls Servo ist ein Umrechnen des aktuellen Werts der Datenvariablen in die Position, die der Motor ansteuern soll. Dazu wird die Datenvariable mit den Werten „Wert UG“ und „Wert OG“ verglichen. Ist der Wert der Datenvariablen gleich dem definierten Minimalwert „Wert UG“, bedeutet das, dass der Motor ebenfalls seine Minimalposition (Winkel „Motor UG“) anfährt. Befindet sich der Wert der Datenvariablen genau in der Mitte zwischen Minimal- und Maximalwert, wird auch der Motor die Mitte zwischen Minimal- und Maximalwert ansteuern.

4.2.13. Das Modul ServoPWMShield

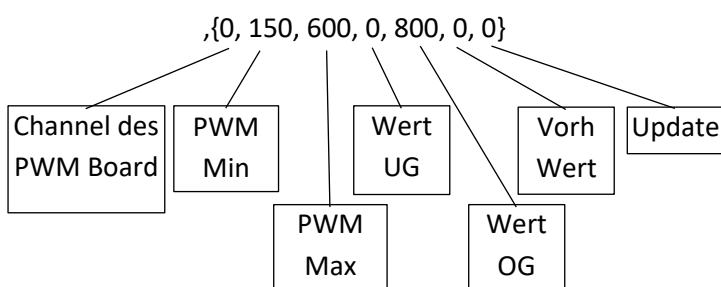


Dieses Modul dient zur Ansteuerung von einem oder mehreren Servo-Motoren, die über ein Motor Shield (z.B. PCA9684, siehe links) mit dem Arduino verbunden sind.

Um einen Servomotor anzusteuern, sind mehrere Steuerungsinformationen festzulegen. Diese sind in dem Datencontainer im Modul Servo zu hinterlegen.

Wichtig: Wenn eine Datenvariable an einem Motor ausgegeben werden soll, ist im Datencontainer des Moduls UserConfig nicht der PIN anzugeben, an dem sich der Motor befindet, sondern die Zeile in dem Datencontainer des Moduls Servo. In dem Servo-Datencontainer finden sich neben dem PIN, an dem der Motor angeschlossen ist, auch weitere Daten, die für die Funktion des Motors erforderlich sind.

Um einen Servomotor anzusteuern, sind mehrere Steuerungsinformationen festzulegen. Diese sind in dem Datencontainer im Modul ServoPWMShield zu hinterlegen.



1. Channel
Der Channel auf dem Motor Shield, auf dem der Servo-Motor angeschlossen wurde.
2. PWM Min
Die Pulsdauer, die den Motor an die untere Grenze fährt (liegt bei ca. 150).
3. PWM Max
Die Pulsdauer, die den Motor an die obere Grenze fährt (liegt bei ca. 600).
4. Wert UG
Der niedrigste Wert, den die Datenvariable annehmen kann (Zahl).
5. Wert OG
Der höchste Wert, den die Datenvariable annehmen kann (Zahl).
6. Vorh. Wert
Platzhalter für den letzten gemessener Wert (Standard: 0). Wird zur Laufzeit benötigt.
Keine Definition erforderlich.
7. Update
Platzhalter für den Zeitpunkt des letzten Steuersignals an den Servo (Standard: 0). Wird zur Laufzeit benötigt. Keine Definition erforderlich.

Jeder Motor ist unterschiedlich und es gibt selbst bei baugleichen Servos Fertigungstoleranzen, weshalb eine Kalibrierung erforderlich ist. Bei der Initialisierung des Arduino wird der Motor seine Minimalposition und Maximalposition anfahren und dort jeweils eine Sekunde verbleiben. Prüft in dieser Zeit, ob der Arduino dort fühlbar oder hörbar vibriert. Wenn ja, dann versucht der Servo eine Position einzunehmen, die aufgrund der Hardwarebeschränkungen nicht erreicht werden kann. Dies wird den Servo wohl nicht beschädigen, sollte aber grundsätzlich vermieden werden. Die Werte PWM Min und PWM Max sind daher bei jedem angeschlossenen Motor zu prüfen und ggf. anzupassen.

Servomotoren haben den Nachteil, dass der Motor sich nicht frei bewegen kann, sondern nur über einen geringen Bewegungsspielraum für einen Arm/Zeiger verfügt (i.d.R. 180°). Der gewaltige Vorteil des Servos liegt aber darin, dass der Arm/Zeiger absolut angesteuert werden kann, d.h. man muss die vorherige Position des Zeigers nicht kennen, um ihn in eine gewünschte Position zu bringen (→ bewege dich auf Position 90°).

Die Funktionsweise des Moduls Servo ist ein Umrechnen des aktuellen Werts der Datenvariablen in die Position, die der Motor ansteuern soll. Dazu wird die Datenvariable mit den Werten „Wert UG“ und „Wert OG“ verglichen. Ist der Wert der Datenvariablen gleich dem definierten Minimalwert „Wert UG“, bedeutet das, dass der Motor ebenfalls seine Minimalposition (Pulsdauer PWM Min) anfährt. Befindet sich der Wert der Datenvariablen genau in der Mitte zwischen Minimal- und Maximalwert, wird auch der Motor die Mitte zwischen Minimal- und Maximalwert ansteuern.

Der Motor kann beschädigt werden, wenn man versucht diesen in Positionen zu fahren, die außerhalb des zulässigen Winkelbereiches (i.d.R. 180°) liegt. Daher ist bitte darauf zu achten, dass die hinterlegten Steuerungsinformationen so gewählt sind, dass es dazu nicht kommen kann.

4.2.14. Das Modul Stepper



Dieses Modul dient zur Ansteuerung von einem oder mehreren Stepper-Motoren (z.B. 28BYJ-48), die jeweils über ein Motor Control Shield (beim 28BYJ-48 das ULN2003) mit dem Arduino verbunden sind. Steppermotoren haben den Vorteil, dass der Motor frei bewegt werden kann ohne an physische Grenzen zu stoßen. Der Nachteil des Steppers liegt aber darin, dass der Motor/die Software nicht weiß, wo sich der Zeiger beim Start des Programms befindet. Da die Ansteuerung relativ erfolgt (→ bewege dich um 20 Schritte nach rechts), muss die vorherige

Position des Zeigers bekannt sein, um ihn in eine gewünschte Position zu bringen. Dies bedingt, dass ein Stepper beim Programmstart manuell oder automatisch geeicht werden muss, um den Arm/Zeiger anschließend in die richtige Position zu bringen.

Wird der Stepper für eine Anzeige verwendet, die über Begrenzungen verfügt (z.B. Speed Indicator) kann die Position bestimmt werden, indem der Zeiger an den Begrenzungen Mikroschalter auslöst. Sobald der Schalter ausgelöst wird, ist die Position des Zeigers bekannt und kann von dort aus zuverlässig berechnet werden.

Bei Stepper-Motoren mit geringem Drehmoment besteht eine weitere Möglichkeit darin, die Bewegung einfach zu begrenzen (Anzeigenadel durch eine Sperre an der Untergrenze blockieren) und den Motor eine volle Drehung vollziehen zu lassen. Bestimmte Stepper-Motoren (z.B. x27-168) haben diese Sperren bereits eingebaut.

Bei der Realisierung von Anzeigen, bei denen sich eine Nadel frei drehen können muss (z.B. Altimeter, Kompass) muss die Anzeige entweder manuell eingestellt werden oder man arbeitet mit einer Lichtschranke, die nur bei einer bestimmten Position der Anzeige ausgelöst wird.

Derzeit gibt es in der Auslieferung des BMSAIT kein funktionierendes Beispiel, die die Funktionsweise eines Stepper-Motors im Zusammenspiel mit Fluginformationen aus BMS über BMSAIT demonstriert. Dies liefere ich in einer späteren Version nach.

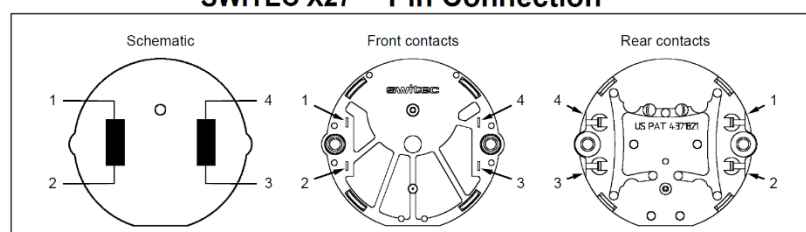
4.2.15. Das Modul StepperX27



Dieses Modul dient zur Ansteuerung des Stepper Motor X27.168. Der besondere Vorteil dieser Motoren liegt darin, dass diese sich nicht frei drehen können (360°), sondern nur einen Winkel von 315° abdecken. Dies stellt aber keinen Nachteil dar, da diese Hardwarebeschränkungen, kombiniert mit dem geringen Drehmoment dieser Motoren, es erlauben den Motor durch das bewusste

„Überdrehen“ an den Anschlägen in einen definierten Zustand zu bringen. So kann die Software die Position der Anzeigenadel in Erfahrung bringen und

SWITEC X27 Pin Connection



anschließend richtig positionieren. Der X27 kann daher fast wie ein Servomotor betrachtet werden, hat aber den Vorteil der größeren Winkelreichweite gegenüber den üblichen Servos, die nur 180° zuverlässig abdecken können.

Das Modul X27 erwartet von der BMSAIT App einen Datensatz, der die aktuelle Position des Motors in Form einer Zahl von 0 (vollausschlag links) bis 65535 (Vollausschlag rechts) repräsentiert. Die Rohdaten aus FalconBMS müssen dafür von der WindowsApp transponiert werden. Ebenso erfolgt eine nicht-lineare Umrechnung (z.B. Anzeige Skala FTIT oder RPM) durch die App (siehe 3.5).

Einstellungen im Modul StepperdataX27:

```
StepperdataX27 stepperdataX27[] =
{
  // {PIN1 PIN2 PIN3 PIN4}   arc      last
  { { 2, 3, 4, 5 }, 315*3,    0 } // example: FTIT
};
```

SteuerPINs

Arc

Letzter Wert

1. SteuerPINs

Hier sind die vier PINs anzugeben, an denen der Motor angeschlossen ist. Achtet auf die richtige Reihenfolge der Kabel mit den Anschlüssen am Motor. Gibt es hier Fehler, wird der Motor sich nicht wie gewünscht verhalten.

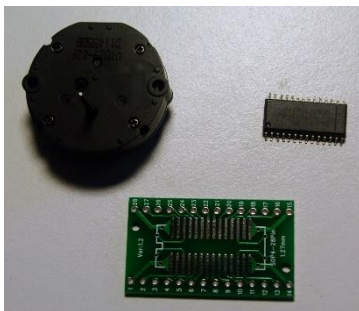
2. Arc

Hier ist die Zahl der Schritte anzugeben, die der Motor zwischen den Begrenzungen einnehmen kann. Die Genauigkeit beim X27.168 liegt bei 3 Schritten pro Grad und einem Winkel von 315°. Soll der Motor eine Anzeige mit einem geringeren Winkel als 315° abbilden, kann dies durch Verkleinerung der Zahl durchgeführt werden.

3. Letzter Wert

Hier wird der zuletzt bekannte Rohdatenwert gespeichert. Dies wird nur zur Laufzeit benötigt, der Wert muss nicht verändert werden.

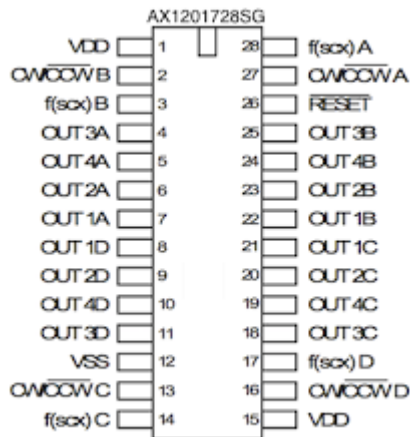
4.2.16. Das Modul StepperVID



Dieses Modul dient zur Ansteuerung mehrerer Stepper Motoren vom Typ X27.168 oder baugleichen Varianten (z.B. X40 mit zwei konzentrischen Achsen). Der besondere Vorteil dieser Motoren liegt darin, dass diese sich nicht frei drehen können (360°), sondern nur einen Winkel von 315° abdecken. Dies stellt aber keinen Nachteil dar, da diese Hardwarebeschränkungen, kombiniert mit dem geringen Drehmoment dieser Motoren, es erlauben den Motor dank der Begrenzungen in einen definierten Zustand zu bringen (die Software muss die Position des Motors kennen, um eine Anzeige in die richtige Position zu fahren). Der X27 kann daher fast wie ein Servomotor betrachtet werden, hat aber den

Vorteil der größeren Winkelreichweite gegenüber den üblichen Servos, die nur 180° zuverlässig abdecken können.

Jeder X27.168 benötigt vier PINs zur Ansteuerung, was die Kapazität eines Arduino ggf. übersteigen kann. Zudem reicht die Stromversorgung eines Arduino nicht aus, um mehrere Motoren zuverlässig ansteuern zu können.



Hier kommt der Controller Chip VID6606 ins Spiel. Ein Chip kann bis zu vier Motoren ansteuern und entlastet den Arduino durch geringere PIN Bedarfe (nur 2 je Motor) und übernimmt die Spannungsversorgung der Motoren. Nebenstehend ist die PIN Belegung eines VID6606 abgebildet.

Das Modul StepperVID erwartet von der BMSAIT WinApp einen Datensatz, der die aktuelle Position des Motors in Form einer Zahl von 0 (vollausschlag links) bis 65535 (Vollausschlag rechts) repräsentiert. Die Rohdaten aus FalconBMS müssen dafür von der WindowsApp transponiert werden. Ebenso erfolgt eine nicht-lineare Umrechnung (z.B. Anzeige Skala FTIT oder RPM) durch die WinApp.

Im Arduino-Coding des Moduls sind Einstellungen vorzunehmen.

```
{
  // {PIN Step PIN Dir}      arc      last
  { { 8, 9 }, 315*12, 0 }, // exan
  { { 6, 7 }, 315*12, 0 }, // exan
  { { 4, 5 }, 225*12, 0 }, // exan
  { { 2, 3 }, 315*12, 0 }  // exan
}
```

SteuerPINs

Arc

Letzter Wert

1. SteuerPINs

Hier sind die zwei PINs anzugeben, über die die Steuersignale für einen Motor an das VID6606 weitergegeben werden. Der erste PIN steuert die gewünschte Drehrichtung des Motors an und der zweite PIN gibt den Befehl zur Bewegung.

2. Arc

Hier ist die Zahl der Schritte anzugeben, die der Motor zwischen den Begrenzungen einnehmen kann. Das VID6606 ermöglicht hier eine Genauigkeit von 12 Schritten pro Grad bei einer Winkelabdeckung von 315° des Motors. Soll der Motor eine Anzeige mit einem geringeren Winkel als 315° abbilden, kann dies durch Verkleinerung der Zahl durchgeführt werden.

3. Letzter Wert

Hier wird der zuletzt bekannte Rohdatenwert gespeichert. Dies wird nur zur Laufzeit benötigt, der Wert muss nicht verändert werden.

4.2.17. Das Modul MotorPoti



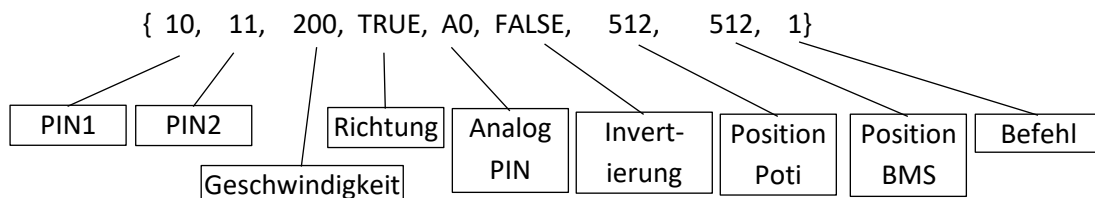
Dieses Modul dient zur Ansteuerung von einem oder mehreren Potentiometern, bei denen einerseits ein Analogwert ausgelesen und weitergegeben werden kann, andererseits aber auch die Position durch einen angeschlossenen Motor verändert werden kann (Beispiel: Regler auf dem Trim-Panel).

Die Software vergleicht dabei kontinuierlich den Wert des Potentiometers, der Position einer Analogachse in BMS und der Auswirkungen in Falcon BMS (Trimwert). Wird eine Abweichung festgestellt, wird ermittelt, ob der Wert in der Simulation verändert wurde oder der Potentiometer gedreht wurde. Der jeweils andere Wert wird anschließend angeglichen.



Der Motor des MotorPoti muss über ein MotorShield (H-bridge) mit dem Arduino verbinden werden. Ich empfehle hier ein Controller Board wie das HG7881 (siehe links).

In dem Modul MotorPoti muss eine Konfiguration des Motorpotentiometers vorgenommen werden. Für jeden Motorpotentiometer muss eine Zeile im Block „Struc_MotorPoti motorPoti[]={}“ ergänzt werden.



1. PIN1
Der PIN, über den die Drehrichtung des Motors angezeigt wird (A1, B1, C1 oder D1 des HG7881 Board)
2. Pin2
Der PIN, über den ein Drehbefehl für den Motor ausgelöst wird (A2, B2, C2 oder D2 des HG7881 Board)
3. Geschwindigkeit
Hier wird eine Pulsdauer angegeben (0..255), mit der die Geschwindigkeit gesteuert wird, mit der sich der Motor bewegen soll. Das funktioniert nur, wenn der Motorpoti an die hierfür geeigneten PWM-Pins des Arduino angeschlossen sind (Beispiel Uno/Nano: 3, 5, 6, 9, 10, 11). Sollen hier nicht-PWM Pins verwendet werden, muss bei der Geschwindigkeit ein Wert von mindestens 124 angegeben werden.
4. Richtung
Hier wird die letzte Drehrichtung des Motors gespeichert.
5. Analog PIN

Hier ist der PIN anzugeben, an dem der SignalPIN des Potentiometers angeschlossen ist.

6. Invertierung

Standard ist FALSE. Wenn die Analogachse in BMS invertiert wurde, muss hier TRUE eingetragen werden.

7. Position Poti

Hier wird der Wert gespeichert, den der Potentiometer zuletzt angenommen hat (0..1024).

8. Position BMS

Hier wird der Wert gespeichert, der gemäß Simulation anzunehmen ist (0..1024).

9. Befehl

Hier ist der Kommandobefehl anzugeben, über den die WinApp eine Verknüpfung des Potentiometers mit einer Analogachse des vJoy herstellt (siehe 3.2.5, Bereich „Joystickachse“).

Das Modul Motorpoti versucht automatisch eine Synchronisierung der Bewegungsrichtung des Motors, der daraus folgenden Bewegungsrichtung des Potentiometers, der Analogachse des vJoy und der Achsenzuordnung in Falcon BMS durchzuführen. Sollte es Probleme mit diesem Modul geben, kann es aber erforderlich sein, die Verkabelung des Motorpotentiometers (+/- am Motor oder die +/- am Potentiometer) zu tauschen.

4.2.18. Das Modul OLED

Das Modul OLED ist dazu gedacht, einfache Informationen auf einem OLED zur Anzeige zu bringen.

Die größte Schwierigkeit besteht darin, die richtige Ansteuerung des Displays über den Arduino zu erreichen. Da es sehr viele verschiedene OLED Größen, Kommunikationschips, Anbieter und verschiedene Anschlussmöglichkeiten gibt, muss hier eine Vorkonfiguration erfolgen. Dies erfolgt am Anfang des Sketches durch Definition eines Konstruktors, der alle relevanten Informationen enthält, die für die korrekte Kommunikation erforderlich ist.

Auswahlblock Speicherverwaltung:

```
/// Declare screen Object
#if defined(DUE) || defined(DUE_NATIVE) || defined(MEGA)
//arduino board with enough memory will use the unbuffered mode
U8G2_SSD1322_NHD_256X64_F_4W_SW_SPI displayDED(U8G2_R0, /* c)
#else
//arduino board with low memory will have to use the buffered
U8G2_SSD1322_NHD_256X64_1_4W_SW_SPI displayDED(U8G2_R0, /* c)
#endif
```

Die Ansteuerung eines OLED kann zudem viele Ressourcen des Arduino in Anspruch nehmen.

Aus diesem Grund ist im Code ein Block hinterlegt, der abhängig von dem in der User-

Konfiguration gewählten Arduino-Typ festlegt, wie der Arduino das Display ansteuert (dies betrifft den im folgenden Absatz beschriebenen Wert „Speicher“). Wenn ihr ein anderes Display als in meinem Beispiel verwendet und die Konstruktorzeilen ersetzen müsst, dann achtet darauf, dass im ersten Block die Speicherverwaltung auf „F“ und im zweiten Block auf „1“ gesetzt ist.

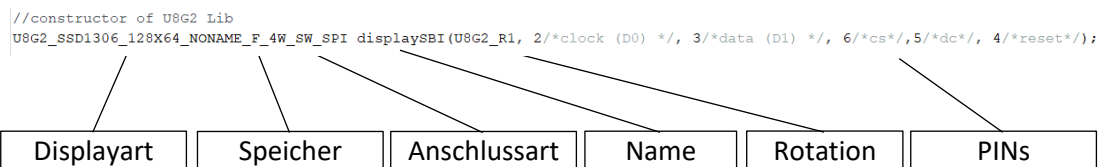
Der U8G2Lib Konstruktor:

Für dieses und die folgenden Module SpeedbrakeIndicator, FuelFlowIndicator und das DED/PFL ist als wichtigste Einstellung die Einbindung des OLED vorzunehmen.

Die Ansteuerung des OLED Displays erfolgt über die Arduino Bibliothek U8G2. Da es viele verschiedene Arten und Größen von OLED gibt, muss der Bibliothek die Information des verwendeten Displays mitgegeben werden. Dies erfolgt über den Konstruktor, der spezifisch für eine bestimmte Art von Display vorgesehen ist. Die vollständige Liste der Konstrukteure findet ihr hier:

<https://github.com/olikraus/u8g2/wiki/u8g2setupcpp>

Der Konstruktor setzt sich dabei aus einer Displayart, einer Einstellung der Speicherverwaltung und einer Anschlussart zusammen.



1. Displayart

Sucht aus der Referenzliste der Konstrukteure den Eintrag mit eurem Controllerchip der Displaygröße und kopiert diesen an die entsprechende Stelle im Kopf des ArduinoCodes des Moduls SBI.

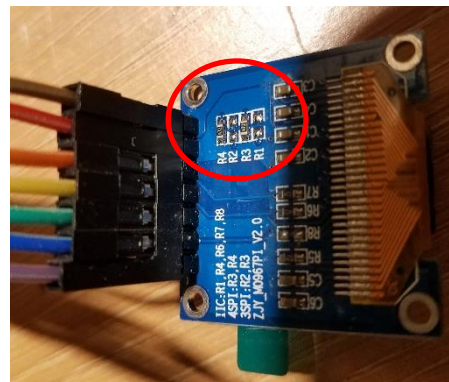
2. Die Speicherverwaltung

Hier wird eingestellt, wie die Daten des OLED gepuffert werden. Bei ausreichenden Systemressourcen sollte die volle Pufferung (F) aktiviert werden (z.B. mit einem Mega oder Due). Wenn die Arduino IDE euch sagt, dass das Modul den Speicher sprengt, dann wählt hier eine Teilpufferung (1 oder 2).

3. Die Anschlussart

Hier könnt ihr festlegen, wie ihr das Display mit dem Arduino verbinden wollt. Hier gibt es – abhängig vom Controllerchip- die Möglichkeit einer Ansteuerung über ein 2-Kabel I²C über ein 3- oder 4-kabeliges SPI bis zu einer 8-Kabel Parallelansteuerung. Ich empfehle eine 4-Kabel SPI Verbindung, wenn dies angeboten wird (in diesem Beispiel 4W_SW_SPI).

Beachtet, dass die Anschlussart auf der Platine des OLED Displays über eine Lötbrücke eingestellt werden muss!



4. Der Name

Hier ist der Variablenname anzugeben, unter dem das OLED im Arduino Quellcode angesprochen wird.

Modul	Name im Konstruktor
OLED	displayOLED
SBI	displaySBI
FFI	displayFFI
DED/PFL	displayDED

5. Die Rotation

Hier wird durch eine Option festgelegt, ob das Display gedreht dargestellt wird (U8G2_R0 = nicht gedreht; U8G2_R1 = 90° CW ; U8G2_R2 = 180° ; U8G2_R3 = 90° CCW).

6. Die PINs

Gebt hier die PINs an, an denen das Display mit dem Arduino verbunden wurde.

Das Modul OLED verfügt zudem über Möglichkeiten der Layoutanpassung. Über die folgenden Einstellungen kann die Position des anzuzeigenden Textes beeinflusst werden:

```
//Layout settings

//Rotation: U8G2_R0  U8G2_R1  U8G2_R2  U8G2_R3
//           (0°)    (90°CW)  (180°)  (270°CW)
#define OFFSETX 0 //increase this to move Text    right  down   left   up
                  //decrease this to move Text    left   up    right  down
#define OFFSETY 0 //increase this to move Text    down   right  up     left
                  //decrease this to move Text    up    left   down   right
```

Normalerweise wird mit einer Erhöhung des Parameters OFFSETX der Text nach rechts verschoben. Dies gilt aber nur, wenn das Display nicht gedreht wurde (siehe Konstrukturoption zur Drehung des OLED). Die rechts von der Option dargestellte Tabelle gibt eine Hilfestellung, welcher Wert erhöht/gesenkt werden muss, um in Abhängigkeit von der Drehung des Displays den Text in die gewünschte Richtung zu bewegen.

4.2.19. Das Modul Speedbrake Indicator



Das Modul Speedbrake Indicator dient zur Anzeige des SBI auf einem OLED display.

Dieses Modul basiert weitestgehend auf DEDuino (<https://pit.uriba.org/tag/deduino/>) und wurde lediglich für eine Datenkommunikation mit BMSAIT und ein paar Anzeigesteuern angepasst.

Abhängig von der Stellung der Speedbrake soll der Text „CLOSE“, die Grafik für die geöffnete Speedbrake (3x3 Kreise) oder die OFF-Anzeige erscheinen.

Für die Abbildung kommen kleine Displays (ca. 1 Zoll) mit 128x64 Pixel in Frage. Im Coding gehe ich davon aus, dass das Display über den sehr häufig anzutreffenden Controllerchip SSD1306 angesteuert wird. Andere Displays gehen auch, erfordern aber eine Anpassung (siehe Einstellungen zum Konstruktur).

A) Der U8G2 Konstrukt

Die notwendigen Einstellungen sind im Kapitel 4.2.18 beschrieben.

Gebt zudem bei den Optionen „SB_SCREEN_W“ und „SB_SCREEN_H“ die Anzahl der Pixel eures Displays (Standard: 128 Breite und 64 Höhe) an.

B) Layouteinstellungen

Grundsätzlich sollte die Einstellung des richtigen Konstruktors reichen, um die SBI zur Anzeige zu bringen. Um das Display aber präzise steuern zu können, habe ich in dem Modul einen Wertebereich hinterlegt, der für die Feinpositionierung der Anzeige auf dem OLED genutzt werden kann.

Wenn das SBI bei euch verschoben ist, könnt ihr die Anzeige durch Anpassung dieser Werte auf dem OLED individuell anpassen.

```
//Layout settings
//Rotation: U8G2_R0 U8G2_R1 U8G2_R2 U8G2_R3
//           (0°)   (90°CW) (180°) (270°CW)
#define OFFSETX 0 //increase this to move Text right down left up
                  //decrease this to move Text left up right down
#define OFFSETY 0 //increase this to move Text down right up left
                  //decrease this to move Text up left down right
```

Mit den Optionen OFFSETX und OFFSETY kann das Display bei Bedarf verschoben werden, damit es genau in die Panelrahmen passt.

```
#define SBIDELAY 250
//#define ANIMATION
#define OFFSET_FRAMES 6
```

Die Option SBIDELAY gibt die Zeit in Millisekunden an, wie lange die OFF Anzeige nach Bewegung der Speedbrake eingeblendet wird (Standard: 250).

Mit der Option ANIMATION wird festgelegt, ob die Anzeige des Speedbrake Indicator bei Wechsel des Status animiert ist.

Ist die Option deaktiviert, wird beim Wechsel zwischen OPEN/CLOSED immer das OFF-Flag als Standbild gezeigt. Ist die Option aktiviert, scrollt die Anzeige von einem Modus zum anderen. Mit der Option OFFSET_FRAMES wird dabei festgelegt, wie viele Zwischenschritte angezeigt werden. Umso mehr Schritte man hier einstellt, umso flüssiger wird die Anzeige theoretisch erfolgen. In der Praxis kommt das Anzeigergebnis auf die Leistungsfähigkeit des Arduino und des Displays an.

Verhalten des Moduls:

- Das Display ist grundsätzlich deaktiviert. Ist BMSAIT auf dem PC nicht aktiv und werden daher keine Daten vom Arduino empfangen, wird das Display nach 10 Sekunden in einen Ruhemodus wechseln.
- Ist BMSAIT aktiv, aber BMS/DCS nicht in der 3D Welt, wird die „OFF“-Anzeige eingeblendet.
- Ist die Stromversorgung der Maschine nicht aktiv, wird die „OFF“ Anzeige eingeblendet.
- Ist die Animation deaktiviert, wird bei jeglicher Bewegung der Speedbrake kurzzeitig die „OFF“-Anzeige eingeblendet.
- Ist die Speedbrake geschlossen (unter 1°) wird die „CLOSED“ Anzeige eingeblendet
- Ist die Speedbrake geöffnet (über 1°) wird die „OPEN“ Anzeige eingeblendet

4.2.20. Das Modul FFI



Dieses Modul ermöglicht die Anzeige des FuelFlowIndicator auf einem OLED Display.

Um das OLED Display korrekt ansteuern zu können, sind Einstellungen erforderlich (Konstruktor). Diese sind in Kapitel 4.2.18 beschrieben worden.

Für die Anzeige ist eine spezielle Datenvariable erforderlich, in der der anzuzeigende Text von der BMSAIT WinApp in der korrekten Formatierung bereitgestellt wird (ID 0511/0521).

Dieses Modul basiert weitestgehend auf DEDuino (<https://pit.uriba.org/tag/deduino/>) und wurde lediglich für eine Datenkommunikation mit BMSAIT und ein paar Anzeigesteuern angepasst.

A) Der U8G2 Konstruktor

Die notwendigen Einstellungen sind im Kapitel 4.2.18 beschrieben.

Gibt zudem bei den Optionen „SB_SCREEN_W“ und „SB_SCREEN_H“ die Anzahl der Pixel eures Displays (Standard: 128 Breite und 64 Höhe) an.

B) Schriftarteinstellungen

```
// FONT DEFINITIONS - Main
#define ffFont FalconFFI /
#define FF_CHAR_W 20 // -
#define FF_CHAR_H 30 // -
```

Hier erfolgt die Einstellung der auf dem Display anzuzeigenden Schriftart. Diese ist spezifisch für das FFI eingestellt worden und sollte nicht verändert werden. Zu beachten ist, dass diese Schriftart nur die zwingend erforderlichen Buchstaben enthält (Zahlen, Buchstaben für „FuelFlow“ und „PPH“). Andere Inhalte können auf dem Display daher nicht angezeigt werden.

Mit den Optionen FF_CHAR_W und FF_CHAR_H wird der Software die Größe der Schriftzeichen mitgeteilt. Diese sollten nicht verändert werden.

C) Layouteinstellungen

```
#define FF_OFFSETX 0
```

Mit diesen Optionen kann die Position des FuelFlow Wertes auf der Anzeige beeinflusst werden, falls dies für eine Feinausrichtung des Textes in einem Panel/Bezel erforderlich ist.

```
#define FF_OFFSETY 0
```

D) Optionen

```
#if defined(MEGA) || defined(DUE) || defined(DUE_NATIVE)
```

Die FFI Anzeige von Uri_ba hat zwei zusätzliche Optionen zur Anzeige des FFI.

```
#define REALFFI
```

Diese Optionen sind standardmäßig

```
#define BEZEL
```

aktiviert, wenn Arduinos mit guter Leistung verwendet werden. Ist dies nicht gewünscht, kann man diese Features durch Auskommentieren („//“ davorsetzen) deaktivieren.

Mit der Option REALFFI erfolgen weitere Animationen der Ziffern, wenn sich der FuelFlow ändert.

Mit der Option BEZEL wird auf dem Display ein Rahmen mit Anzeige der feststehenden Text „Fuel Flow“ und „PPH“ gezeichnet.

Verhalten des Moduls:

- Das Display ist grundsätzlich ausgeschaltet. Ohne Verbindung zu BMSAIT ist das Display nicht aktiv.
- Ist eine Datenverbindung mit BMSAIT aktiv, aber FalconBMS nicht in der 3D Welt, wird ein Nullwert angezeigt.
- Solange sich der PC in der 3D-Welt befindet, wird der zuletzt gültige FuelFlow angezeigt.
- Verlässt der PC die 3D-Welt, wird wieder ein Nullwert angezeigt.
- Die Anzeige erlischt, sobald 10 Sekunden keine Daten mehr von BMSAIT empfangen wurden.

4.2.21. Das Modul DED/PFL



Mit BMSAIT können auch OLED Displays für verschiedene Zwecke angesteuert werden. OLED Displays gibt es in vielen verschiedenen Größen und unterschiedlichen Controllerchips, so dass es nicht möglich ist, mit diesem Modul eine umfängliche Lösung für alle Typen von OLEDs abzubilden.

Dieses Modul basiert auf DEDuino (<https://pit.uriba.org/tag/deduino/>), wurde aber für die Nutzung mit BMSAIT an einigen Stellen angepasst.

A) Der U8G2 Konstruktor

Die Ansteuerung des OLED Displays erfolgt über die Arduino Bibliothek U8G2. Um die Bibliothek aufzurufen und das OLED richtig nutzen zu können, ist im Kopf des Moduls ein Konstruktor zu definieren. Die dafür erforderlichen Einstellungen sind in Kapitel 4.2.18 beschrieben.

B) Optionen

```
#define PRE_BOOT_PAUSE 1000  
#define POST_BOOT_PAUSE 1000
```

Die Art und Weise der Ansteuerung des DED lässt keine individuellen Layouteinstellungen zu. Einzige Optionen sind daher die Angaben PRE_BOOT_PAUSE und POST_BOOT_PAUSE, mit der die Zeit in Millisekunden angegeben werden kann, die das DED beim Hochfahren das Testbild anzeigt.

Funktionsweise:

Dieses Modul ermöglicht die Abbildung des DED oder des PFL auf einem 254x64 OLED Display. Für die Anzeige des DED und des PFL sollen 5 Zeilen mit jeweils 24 Zeichen abgebildet werden. Bei der Displaygröße von 254x64 Zeichen ergibt sich somit eine Zeichengröße von 12x10 Pixel. Hierbei ist zu beachten, dass für die Ausgabe von Text auf dem OLED eine Schriftart installiert werden muss. Die Schriftart hat eine feste Größe für jedes Zeichen. Daher muss man bereits vorab wissen, welche Information wie groß auf dem Display zu sehen sein soll, um die entsprechende Schriftart laden zu können. Aufgrund der begrenzten Ressourcen eines Arduinos hat man nur sehr begrenzte Möglichkeiten mehrere Schriftarten zu hinterlegen. Dem Modul DED liegt eine Schriftart bei, die in der Datei FalconDEDFont.h als C Code gespeichert ist (Siehe Anlage 5.2.5). Die Schriftart enthält dabei nur die für die Anzeige des DED erforderlichen Zeichen (Kleinbuchstaben und einige Sonderzeichen fehlen) und ist für andere Darstellungen nicht geeignet. Jedem Schriftzeichen ist dabei ein Code zugeordnet, der in der Regel mit dem ASCII Code übereinstimmt, bei bestimmten DED-spezifischen Zeichen aber abweicht.

Die Daten des DED finden sich in zwei Variablen der SharedMem (230 „DED“ und 245 „INV“). Diese Daten sind aber nicht ohne weiteres lesbar, da das DED über Sonderzeichen verfügt, die im normalen Zeichenumfang nicht enthalten sind (z.B. invertierter Text, Cursorpfeile). Die BMS WinApp liest daher die Daten aus der SharedMem und verändert den Bytecode vor dem Versenden an den Arduino. Auf dem Arduino wird damit bereits die richtige Zuordnung zu einem anzuzeigenden Zeichen gemäß dem hinterlegten Wertevorrat empfangen und kann direkt ausgegeben werden.

Verhalten des Moduls:

- Das Display ist grundsätzlich ausgeschaltet. Ohne Verbindung zu BMSAIT ist das Display nicht aktiv.
- Ist eine Datenverbindung mit BMSAIT aktiv, aber FalconBMS nicht in der 3D Welt, wird ein Testwert angezeigt.
- Solange sich der PC in der 3D-Welt befindet, wird der aktuelle DED/PFL Inhalt angezeigt
- Verlässt der PC die 3D-Welt, wird wieder ein Testwert angezeigt.
- Die Anzeige erlischt, sobald 10 Sekunden keine Daten mehr von BMSAIT empfangen wurden.

5. Anlagen

5.1.Quellen / Verweise

„F4SharedMem.dll“ von LightningViper (<https://github.com/lightningviper/lightningtools>).

Windows Input Simulator Plus von Michael Noonan, Theodoros Chatzigiannakis
(<https://github.com/TChatzigiannakis/InputSimulatorPlus>).

Virtual Joystick Emulator vJoy von Shaul Eizikovich (<http://vjoystick.sourceforge.net/site/>)

Das BMSAIT Icon wurde von Ahmad Taufik gezeichnet und von der Seite thenounproject.com heruntergeladen.

Codebausteine für die Module DED, FuelFlow wurde dem DEDunino Projekt von Uriba übernommen
(<https://pit.uriba.org/tag/deduino/>).

Der Code für die X27 Stepper Motoren sowie die Ansteuerung des VID6606 Chip stammen von Guy Carpenter und wurden von der Seite <https://guy.carpenter.id.au/gaugette/2017/04/29/switecx25-quad-driver-tests/> übernommen.

5.2.Datenfelddescriptions

5.2.1. Datenvariablen (BMSAIT-Variablen.csv, Windows-App)

ID

Die ID ist eine eindeutige Kennzeichnung einer Datenvariablen. Die ID wird im Programmcode einem Bereich der SharedMem des BMS zugeordnet. Eine Veränderung oder Ergänzung der IDs ist daher nicht ohne Eingriff des Entwicklers möglich.

Gruppe

Die Gruppe ist eine selbstdefinierte Einteilung der Datenvariablen in verschiedene Kategorien. Diese dient nur zur Sortierung.

Format

Das Format (anderer Begriff: Datentyp) legt fest, welchen Wert eine Variable annehmen kann und welche Operationen damit möglich sind. Daten der SharedMem liegen in unterschiedlichen Formaten vor. BMSAIT muss je nach Datentyp unterschiedlich vorgehen, weshalb die Angabe des Formats in der Windows-App und im Arduino wichtig ist. Das Format/der Datentyp einer BMSAIT-Datenvariable ist in der BMSAIT-Variablen.csv fest vorgegeben. Die Angabe eines falschen Datentyps im Feld Format kann dazu führen, dass es zu Problemen kommt.

<u>Datentyp</u>	<u>Begriff lang</u>	<u>Abkürzung</u>	<u>Wertebereich</u>
<u>Datenbyte</u>	<u>Byte</u>	<u>y</u>	<u>0x00 – 0xFF (dezimal: 0-255)</u>
<u>Ganzzahl</u>	<u>Integer</u>	<u>i</u>	<u>-32,768 bis 32,767</u>
<u>Nachkommazahl</u>	<u>Float</u>	<u>f</u>	<u>$-3.4 \cdot 10^{38}$ bis $+3.4 \cdot 10^{38}$</u>
<u>Text</u>	<u>String</u>	<u>s</u>	<u>Beliebige Anzahl an Zeichen</u>
<u>Wahrheitswert</u>	<u>Bool</u>	<u>b</u>	<u>T(rue) – F(alse) (dezimal 0-1)</u>
<u>Datenbytegruppe</u>	<u>Byte[]</u>	<u>1</u>	<u>Beliebige Anzahl von Bytes</u>
<u>Ganzzahlgruppe</u>	<u>Int[]</u>	<u>2</u>	<u>Beliebige Anzahl an Ganzzahlen</u>
<u>Textgruppe</u>	<u>String[]</u>	<u>3</u>	<u>Beliebige Anzahl von Texten</u>
<u>Kommazahlgruppe</u>	<u>Float[]</u>	<u>4</u>	<u>Beliebige Anzahl an Nachkommazahlen</u>

Die Datei „BMSAIT-Variablen.csv“ enthält zudem Variablen mit den Datentypen ushort, uint und uint[]. Diese werden derzeit durch BMSAIT nicht unterstützt. Wenn (wider Erwarten) Bedarf zum Übertragen dieser Variablen besteht, dann spricht mich bitte an.

Typ

Der Typ ist eine Angabe, in welchem Datenformat die Information dieser Datenvariable gespeichert wird. Die Angabe wird als Verarbeitungshilfe vom BMSAIT genutzt und mit jedem Datenpaket an die Arduino übertragen.

Bezeichnung

Eine Kurzbeschreibung, die i.d.R. aus der F4SharedMem-Bibliothek übernommen wurde.

Beschreibung

Eine etwas ausführlichere Beschreibung, welche Information in der Datenvariable zu finden ist. Dies wurde i.d.R. aus der F4SharedMem-Bibliothek übernommen.

5.2.2. Variablenzuordnung (Windows-App)

Formatierungsstring

Der Formatierungsstring ermöglicht es, den Aufbau des Datenpaketes, das das Programm an den Arduino sendet, zu beeinflussen. Es ist zwingend erforderlich, dass der Formatierungsstring das Zeichen ‚@‘ enthält. Dieses Zeichen repräsentiert als Platzhalter den zu übertragenden Datenwert. Alle Zeichen, die diesem Zeichen vorangestellt werden, werden bei der Übertragung der Datenvariable vorangestellt. Analog gilt dies für alle Zeichen, die nach dem ‚@‘ in den Formatierungsstring geschrieben werden.

Standardmäßig ist hier der Formatierungsstring „<@>“ zu verwenden, wenn mit der Standard-Arduino-Software des BMSAIT gearbeitet wird. Anpassungen sind nur bei speziellen Anforderungen der Nutzer oder bei der Kommunikation mit anderen Arduino-Programmierungen erforderlich.

Testdaten

Der hier eingetragene Wert wird genutzt, um bei Auswahl des Testmodus auch ohne Verbindung mit einem laufenden Falcon BMS Daten an angeschlossene Arduino versenden zu können.

Position

Der hier eingetragene Wert wird mit jedem Datenpaket an den Arduino versendet. Der Wert steuert, in welche Zeile des Datencontainers des Arduino der übertragene Wert geschrieben wird. Diesem Wert kommt damit eine wichtige Bedeutung zu. Eine Programmlogik gewährleistet, dass die Positionsnummer nur einmalig vergeben werden kann. Die Positionsnummer darf nicht 99 betragen. Werte über 100 sind nur im Spezialfall notwendig (Abbildung Backup Radio).

Aktualisierungshäufigkeit

Bei einer Änderung des Datensatzes in der SharedMem wird der aktuelle Wert immer sofort an den Arduino übertragen. Gibt es aber keine Veränderung, werden Datenvariablen zur Reduzierung von Fehlern zwar dennoch übertragen, aber mit einer reduzierten Häufigkeit. Der hier eingegebene Wert steuert dies, indem der Zeitraum angegeben wird, nach dem ein nicht veränderter Wert erneut übertragen wird.

5.2.3. Kommandos (Windows-App)

ID

Eindeutige Kennzeichnung des Kommandos. Wenn ein Arduino diese ID als Kommando überträgt, wird die zu dieser ID gespeicherte Tastenkombination ausgelöst.

Typ

Klassifiziert, ob das Kommando (1) einen Tastendruck oder (2) Joystickknopf auslöst oder (3) eine Joystickachse beeinflusst werden soll.

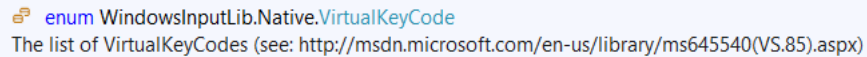
Joystick

Die vJoy Software kann mehrere Joysticks simulieren. Hier wird der Joystick gespeichert, über den ein Signal ausgelöst werden soll.

Taste

Bei Typ Tastatur:

Es wird eine Definition eines Tastendrucks im Format eines virtuellen Tastenkommandos der VirtualInput Library gespeichert.

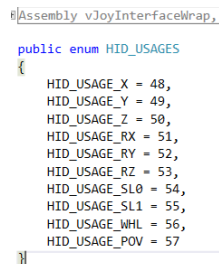


```
enum WindowsInputLib.Native.VirtualKeyCode
The list of VirtualKeyCodes (see: http://msdn.microsoft.com/en-us/library/ms645540\(VS.85\).aspx)
```

Bei Typ Joystickbutton:

Es wird die Nummer des Joystickbuttons gespeichert.

Bei Typ Joystickachse:



```
Assembly vJoyInterfaceWrap,
public enum HID_USAGEES
{
    HID_USAGE_X = 48,
    HID_USAGE_Y = 49,
    HID_USAGE_Z = 50,
    HID_USAGE_RX = 51,
    HID_USAGE_RY = 52,
    HID_USAGE_RZ = 53,
    HID_USAGE_SL0 = 54,
    HID_USAGE_SL1 = 55,
    HID_USAGE_WHL = 56,
    HID_USAGE_POV = 57
}
```

Es wird der Index der gewählten Joystickachse (0..9) gemäß Auflistung der vJoy HID_Usages Liste gespeichert.

Strg

Angabe, ob die Control-Taste zusammen mit der Taste betätigt wird.

Alt

Angabe, ob die Alt-Taste zusammen mit der Taste betätigt wird.

Shift

Angabe, ob die Shift-Taste zusammen mit der Taste betätigt wird.

Modus

Bestimmt, ob eine Taste nur gedrückt, gedrückt und losgelassen oder nur losgelassen wird.

5.2.4. Globalvariablen (Arduino)

BAUDRATE

Die hier definierte Baudrate muss mit der Baudrate der Windows-App übereinstimmen, um eine Kommunikation zu ermöglichen. Standard ist 57600.

DATENLAENGE

Hier wird festgelegt, wie viele Zeichen der Arduino zur Aufnahme eines Datenwertes bereithält. Ist der Wert zu hoch, blockiert das unnötigerweise die knappen Ressourcen des

Arduino. Ist der Wert zu gering, können Daten nicht vollständig übernommen und weitergegeben werden. Standardmäßig ist der Wert 8 vorgesehen (7 verwendbare Zeichen plus ein Zeichen für das im Arduino notwendige Wortabschlusszeichen).

MESSAGEBEGIN

Hier wird ein Byte definiert, anhand dessen der Arduino den Beginn einer neuen Nachricht identifizieren kann. Es muss sich daher um ein Byte handeln, dass im normalen Datenaustausch nie vorkommt. Standardmäßig ist hier 255 vorgesehen.

HANDSHAKE

Wenn der Arduino das hier definierte Byte als InfoByte erhält, wird eine Antwortnachricht mit Angabe der ID dieses Arduino an den PC zurückgesendet. Standard ist 128.

SWITCHPOSITION

Wenn der Arduino das hier definierte InfoByte erhält, wird ein Eingabekommando für alle angeschlossenen Inputs (Kippschalter, Drehschalter) gesendet (siehe 2.2.3). Standard ist 150.

CALIBRATE

Wenn der Arduino das hier definierte Byte als InfoByte erhält, werden Motoren neu kalibriert. Dies ist bei der Nutzung von Stepper-Motoren erforderlich, um diese vor dem Programmstart in eine definierte Position zu bekommen (siehe 2.2.1). Standard ist 160.

ZEROIZE

Wenn der Arduino das hier definierte Byte als InfoByte erhält, werden Motoren auf den Nullwert zurückgesetzt. Dies ist bei der Nutzung von Stepper-Motoren erforderlich, um diese vor dem Programmstart in eine definierte Position zu bekommen (siehe 2.2.2). Standard ist 161.

STARTPULL

Wenn der Arduino das hier definierte Byte als InfoByte erhält, wird die PULL-Verarbeitung gestartet (siehe 2.1.4.2). Standard ist 170.

ENDPULL

Wenn der Arduino das hier definierte Byte als InfoByte erhält, wird die PULL-Verarbeitung beendet (siehe 2.1.4.2). Standard ist 180.

TESTON

Wenn der Arduino das hier definierte Byte als InfoByte erhält, wird der interne Testmodus aktiviert, über den debug-Informationen an die Windows App zurückgemeldet werden. Standard ist 190.

TESTOFF

Wenn der Arduino das hier definierte Byte als InfoByte erhält, wird der interne Testmodus beendet. Standard ist 200.

VAR_BEGIN

Hier wird ein Zeichen definiert, über das der Arduino erkennen kann, dass in einer Übertragung direkt im Anschluss ein Datenwert beginnt. Standard ist ,<‘.

VAR_ENDE

Hier wird ein Zeichen definiert, über das der Arduino erkennen kann, dass der gerade zuvor eingelesene Datenwert nun vollständig ist. Standard ist ,>‘.

TYP_ANFANG

Hier wird ein Zeichen definiert, über das der Arduino erkennen kann, dass es sich beim folgenden Zeichen einer Übertragung um die Angabe eines Datentyps handelt. Standard ist ,{‘.

TYP_ENDE

Hier wird ein Zeichen definiert, über das der Arduino erkennen kann, dass es sich beim vorherigen Zeichen einer Übertragung um die Angabe eines Datentyps handelt. Standard ist ,}‘.

5.2.5. Schriftart DED/PFL

C	A	Z	F		C	A	Z	F		C	A	Z	F		C	A	Z	F		C	A	Z	F	
1					48	0	0	0		69	E	E	E		90	Z	Z	Z		111	o		K	
2					49	1	1	1		70	F	F	F		91	[0		112	p		L	
3-28																								
29					50	2	2	2		71	G	G	G		92	\		1		113	q		M	
30					51	3	3	3		72	H	H	H		93]		2		114	r		N	
31					52	4	4	4		73	I	I	I		94	^		3		115	s		O	
32					53	5	5	5		74	J	J	J		95	_		4		116	t		P	
33	!				54	6	6	6		75	K	K	K		96	`		5		117	u		Q	
34	"				55	7	7	7		76	L	L	L		97	a		6		118	v		R	
35	#				56	8	8	8		77	M	M	M		98	b		7		119	w		S	
36	\$				57	9	9	9		78	N	N	N		99	c		8		120	x		T	
37	%				58	:	:	:		79	O	O	O		100	d		9		121	y		U	
38	&				59	;	;	;		80	P	P	P		101	e		A		122	z		V	
39	'				60	<	*	*		81	Q	Q	Q		102	f		B		123	{		W	
40	(61	=	-	-		82	R	R	R		103	g		C		124			X	
41)				62	>	>	>		83	S	S	S		104	h		D		125	}		Y	
42	*				63	?	°	°		84	T	T	T		105	i		E		126	~		Z	
43	+				64	@				85	U	U	U		106	j		F		127				

44	,	,	,		65	A	A	A		86	V	U	U		107	k		G		128			
45	-	-	-		66	B	B	B		87	W	W	W		108	l		H					
46	.	.	.		67	C	C	C		88	X	X	X		109	m		I					
47	/	/	/		68	D	D	D		89	Y	Y	Y		110	n		J					

C – Bytewert

A – ASCII Zeichen

Z – Code in der FalconBMS SharedMem

F – Zeichen FalconDED Schriftart Arduino