

Anleitung zu „Udo's Suite“



Inhaltsverzeichnis

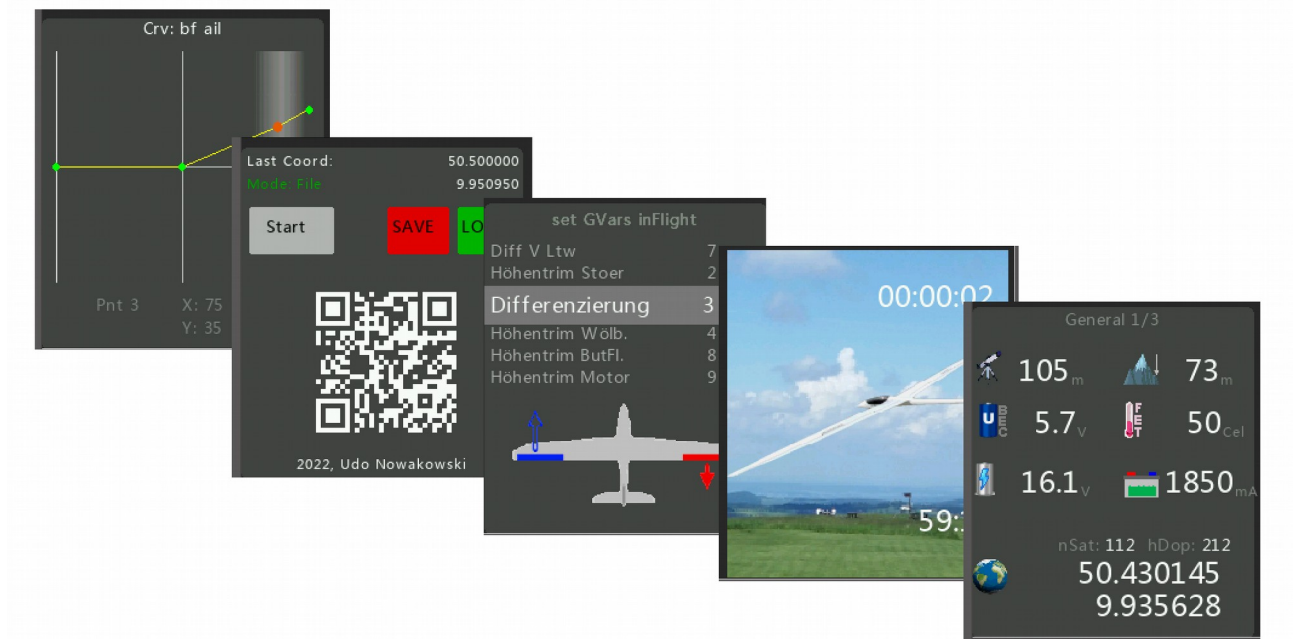
Einleitung.....	3
Allgemeines.....	3
Was ist das besondere ?.....	3
Grundidee:.....	4
Funktionsweise:.....	6
Die einzelnen Apps:.....	7
Telemetrie.....	7
Modell Image.....	8
Kurveneditor.....	9
QR Modelfinder.....	9
TopBar „Full“.....	10
TopBar „Standard“.....	10
TopBar „light“.....	10
(Future: GVAR editor).....	11
(Future: Ansagen).....	12
Installation:.....	13
(1) Dateien & Ordner kopieren.....	13
(2) Screenlayout auswählen.....	14
(3) widget allgemein einrichten.....	15
(4) Konfiguration der einzelnen Apps.....	16
Telemetrie.....	16
Picture.....	17
Kurveneditor.....	18
TopFull.....	19
Anhang.....	20
Modellspezifische Konfiguration.....	20
Standardsettings der Apps.....	20
Der „Telemetrikatalog“.....	20
Landesspezifische Texte / Sprachdateien.....	20
Beispieltemplate „Alpina“.....	21
Sonstige Optionen.....	22
Sprachdateien.....	22
Sensorverzeichnis.....	22
Modellspezifische Parameter.....	22
Internals.....	23

Einleitung

Allgemeines

Udos Suite ist eine Sammlung von Ethos widgets für Tandem Sender von Frsky.

Es existieren z.B. Widgets für umfangreiche Telemetrie Anzeigen, QR Code basierte Google Maps „Modellfinder“, ein Kurveneditor der im Flug mittels Akustik feedback genutzt werden kann, openTx ähnliche Topbars etc...



Die Tandem Sender unterstützen grundsätzlich die Touch Bedienung, was auch in den widgets genutzt wird

Was ist das besondere ?

Herkömmliche Widgets unter Ethos stellen EIN lua „Programm“ dar, welches sich in einen vom Anwender zuvor definierten „Rahmen“ konfigurieren lassen.

Das Layout der Rahmen gibt die Anzahl von widgets per screen vor.

Möchte man mehr widgets implementieren, muss man diese auf verschiedene Seiten platzieren und die Seiten immer kompoett umschalten.

Zur Zeit des Verfassens dieser Anleitung werden auch noch keine TopBars wie bei oTx üblich unterstützt.

Diese Einschränkungen werden durch die Suite aufgehoben.

Bereits in der einfachsten Form können in einem Rahmen 3 Apps abgerufen werden.

Die „maximale“ Konfiguration lässt zwei Rahmen mit jeweils 3 Apps sowie eine (ggf individuell gestaltbare) TopBar zu.

Die einzelnen Apps besitzen eigene „Sprachfiles“, so dass sich beliebige Sprachdarstellungen umsetzen lassen (zur Zeit deutsch und englisch)

Grundidee:

Ich komme von oTx.

Dieses OS konnte ich extrem gut auf meine persönlichen Bedürfnisse zuschneiden.

Eine besondere Rolle spielten dabei die widgets.

Auf Basis von lua wurde man in die Lage versetzt selbst kleine „Apps“ zu schreiben, die genau das machten was man von einer guten Senderanlage erwarten würde.

Also entwickelte ich auf mich zugeschnittene Apps

Sie dienten mir z.B.

- zum einen als umfangreicher Informationsbildschirm (Darstellung etlicher Sensorwerte innerhalb eines Frames),
- zum Feintrim des Modells im Flug (Selektion einer gewünschten Globalen Variable per Schalter, Einstellen per Poti, Selektion der nächsten Gvar, alles mit Ansageunterstützung um nicht auf den screen schauen zu müssen)
- zur erhöhten Sicherheit durch Signalisierung von Zuständen einiger LSW's die Mischer bedienen, oder dem Zustand des Motor „Kill switches“ bzw eines möglichen Motoranlaufs in der Topbar..

Standardmässig nutzte ich auf einer X12 das Layout zur Darstellung mit 4 Widgets pro Seite und TopBar. Durch den Wechsel auf eine X18 unter Ethos ergaben sich einige Änderungen

- Ethos selbst unterstützt derzeit keine eigene „Topbar“
- der X18 Bildschirm ist kleiner, wodurch das „4 widget layout“ erheblich an Präsentationsfläche verliert
- Ethos unterstützt unter lua aber auch touch Ereignisse auf den Tandem Sendern.

Der Wechsel auf die X18 war eine bewusste Entscheidung, die Eigenschaften bzw. Pros und Cons waren vorher klar, nun galt es daraus eine Umsetzung zu entwickeln, die der oTx Lösung um nichts Nachstand sondern sie „übertrumpfte“

Daraus entstand folgendes Grob-Konzept / Anforderungsprofil:

- Anstelle von vier zuvor „kleinen“ widgets auf der X12 werden nur noch zwei in dafür „grösseren“ Frames auf der X18 dargestellt
- per Touch lassen sich innerhalb eines Frames mehrere Apps aufrufen
- per Touch lassen sich innerhalb einer App mehrere Seiten aufrufen
- da keine generische Ethos „lua TopBar“ möglich ist, wird eine eigene programmiert.
- einfache Adaption auf andere Bildschirmauflösungen wie X20 und zukünftige Hardware
- unterstützung mehrerer Sprachen durch selbst editierbare Sprachdateien
- mehrere „Themes“, individuell anpassbar

Letztendlich bedeutete das auch eine eigene widget Verwaltung, im Ergebnis aber deutliche Vorteile gegenüber der oTx Lösung:

- pro widget selbst auf der X18 eine grössere Präsentationsfläche
- weniger HW Schalter und pods notwendig, da einige Funktionalität per touch abrufbar ist
- ein höheres Mass an Individualisierung
- eine deutlich verbesserte Konfigurationsmöglichkeit der Widgets unter lua durch den „Config Handler“

Das ganze sollte dann im Fullscreen Layout laufen

Während der Entwicklung kam es noch zur Idee, das ganze auch in einem einzelnen Frame (Suite1) bzw ohne eigene Topbar in zwei Frames (Suite2) zu adaptieren.

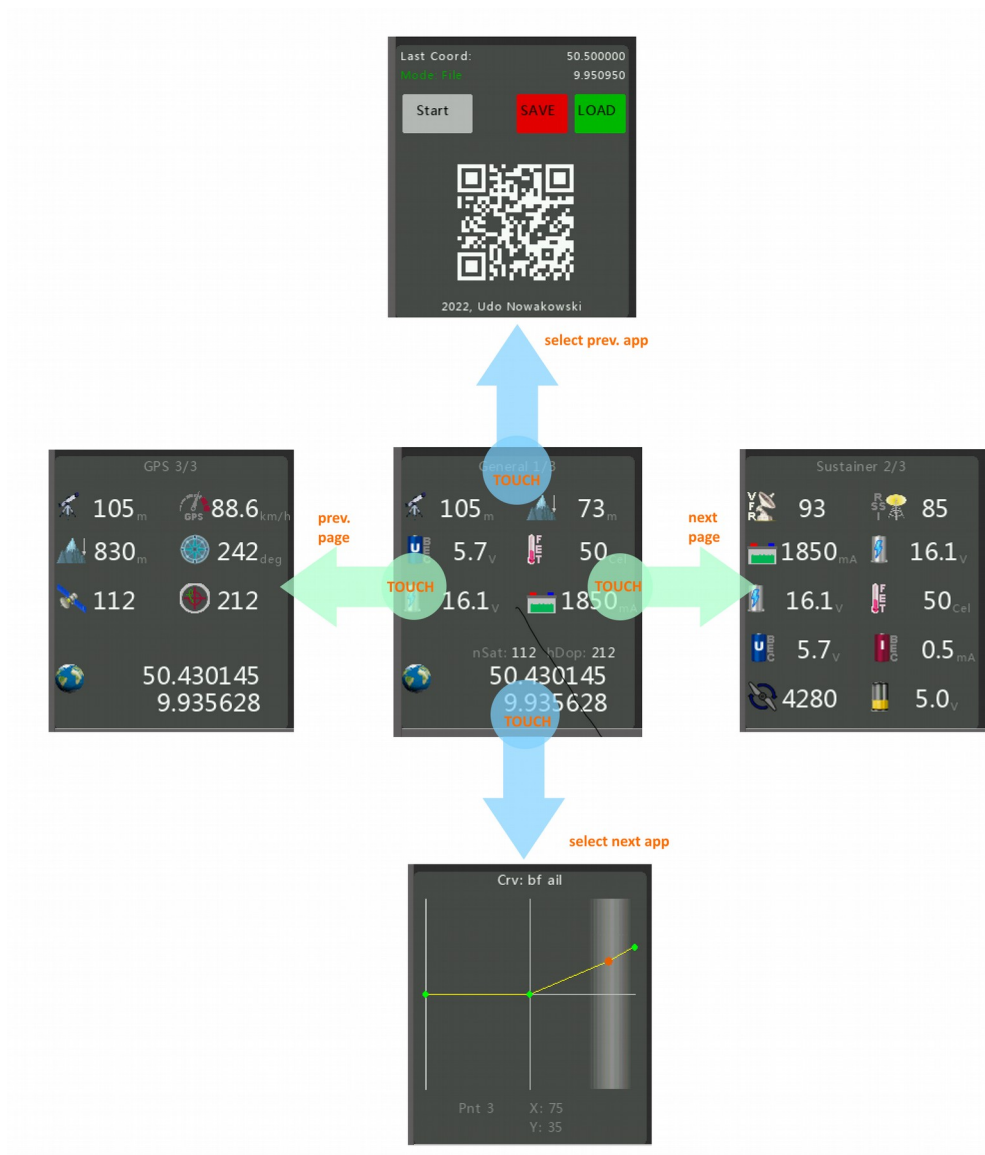
Das Ergebnis ist nun diese Sammlung von Apps, die ich „Udo's Suite“ titliert habe und durch ein übergeordnetes Programm (wrapper) ansteuere

Innerhalb eines Rahmens/Frames lassen sich bis zu drei Apps konfigurieren.
Jede App kann bis zu drei Seiten Umfang besitzen.

In den Frames existieren „reservierte“ Touch Bereiche, um zwischen den Apps umschalten zu können, bzw innerhalb einer App blättern zu können.

Im u.a. Bild ist im Zentrum der Hauptbildschirm der ersten (Telemetrie) App zu sehen.
Um auf eine andere Telemetrie-seite zu gelangen drückt man in die grün markierten Bereiche.
Links wird zurück geblättert, rechts wird vorwärts geblättert.

Die Blau markierten Bereiche ermöglichen es in eine andere App zu springen.
Oben wird die vorherige App angewählt (Google Maps Modelfinder), unten die nächste App (Kurveneditor)

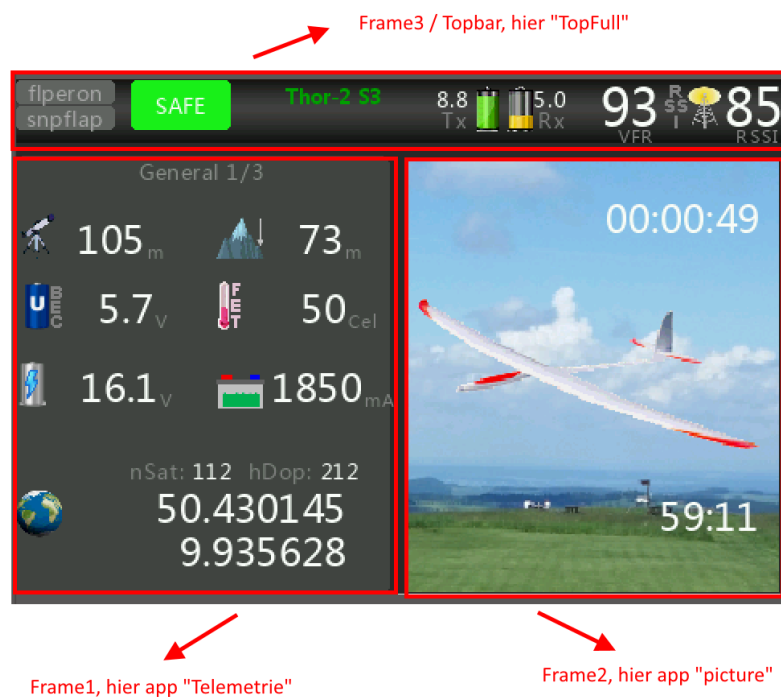


Funktionsweise:

Das „Hauptprogramm“ dient dazu die aktuell ausgewählte app darzustellen, gibt dieser app die „Arbeitsfläche“ (den Frame) bekannt, überprüft touch Eingaben und wertet diese aus, sowie startet Hintergrundaufgaben von apps, falls diese es erfordern.

Das Hauptprogramm „umschliesst“ logisch die eigentlichen apps und organisiert ihren Ablauf, stellt quasi eine eigene widget-Verwaltung dar.
Der Fachausdruck dieses Types von Programm ist „wrapper“

Werden mehrere Apps/Frames bedient, werden die dazugehörigen Apps nacheinander abgearbeitet.



Bsp: Suite3 im „Fullframe“ modus mit drei apps/frames

Die Apps selbst werden in der widget Konfiguration parametrisiert.

Da zu Anfang der Wrapper gar nicht weiss, welche Apps der Anwender wählen wird, sieht man bei der Grundinstallation lediglich eine Auswahl für das Farbschema („thema“) und die Auswahlfelder für die Apps (maximal drei per frame).

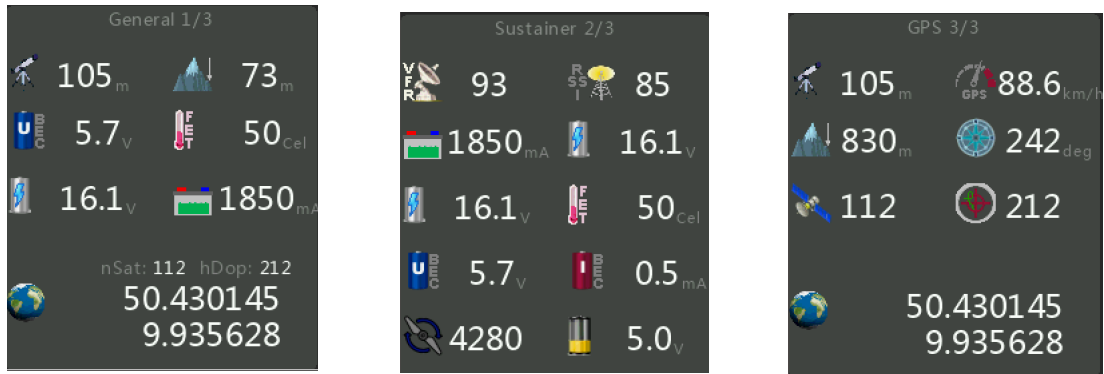
Sobald eine App in der Konfig angewählt wurde, erscheinen die dazugehörigen Felder zur Parametrisierung.

Ist eine App durchkonfiguriert, kann das Konfigmenu verlassen werden und die entsprechende App startet im Frame.

Die Apps werden also dynamisch in die Oberfläche via wrapper eingebunden.

Die einzelnen Apps:

Telemetrie



Die Telemetrie App visualisiert etliche Sensorwerte auf kleinem Raum um sich schnell einen Gesamtüberblick verschaffen zu können.

Auf einer Darstellungsseite können z.B. maximal 10 Werte in 2 Spalten und 5 Zeilen präsentiert werden.

Natürlich lassen sich die Sensoren auch auf weniger Spalten/Zeilen darstellen (-;

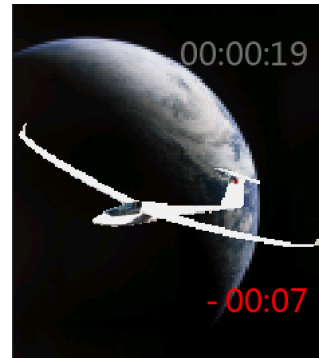
Bis zu 3 Seiten mit Sensoren lassen sich abrufen, so dass man auch „logische Gruppierungen“ pro Seite vornehmen kann.

Es wäre viel zu umständlich alle diese Sensorwerte einzelnen in das widget hineinzukonfigurieren, daher wird eine solche Darstellung von dutzenden von Werten über mehrere Seiten als ein „Set“ / Datensatz in einer Datei definiert und ist so schnell abrufbar.

Wie dies geschieht wird im Anhang erläutert.

Dadurch lassen sich jedenfalls für die individuellen Anforderungen Sets schnell darstellen.

Modell Image



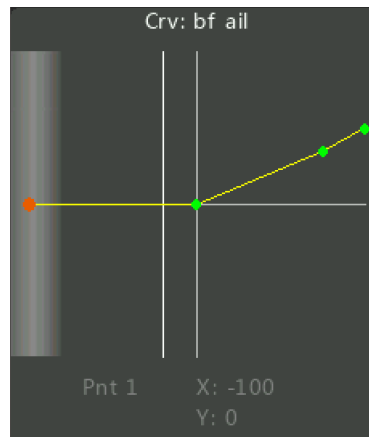
Die App zur Darstellung eines Modellfotos unterscheidet sich im wesentlichen durch zwei Punkte vom Standardwidget:

- es lassen sich Modellfoto (als freigestelltes png Format) und der Hintergrund unabhängig voneinander wählen
- Da das Bild „alleine“ viel Fläche des Displays nutzt können optional zwei Timer zusätzlich eingeblendet werden (ich nutze meistens den FlugzeitTimer sowie den MotorlaufzeitTimer)

Bildgrößen werden automatisch in den Frame skaliert

Kurveneditor

Der Kurveneditor dient dazu eine Kurve im Flug anpassen zu können.



Dazu wird der gewünschte Kurvenpunkt z.B. durch einen Trimmer angewählt, mittel des „zur Kurve gehörigen Inputs“ wird der Punkt angesteuert (akustik feedback, so dass man nicht auf das Display schauen muss), ein analog Geber ermöglicht den Feintrim auf die gewünschte Position.

Per Schalter lässt sich der Urzustand wiederherstellen (reset).

Ideal um z.B. die HR-Kompensationskurve im Butterfly anzupassen u.a.

QR Modelfinder

Ideal wenn ein GPS im Modell verbaut ist.



Die app „merkt“ sich die letzte empfangene GPS Position (diese Routine arbeitet auch, wenn die App nicht im Vordergrund läuft).

Per Touch lässt sich der Wert in einen QR Code darstellen

(Achtung lange rechenzeit, z.B. 15 Sekunden)

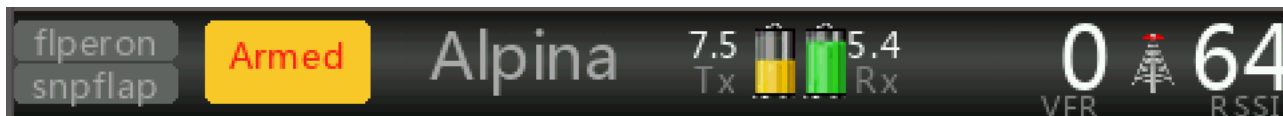
Der QR Code kann per neuerem Handy abfotografiert werden, Google maps öffnet sich dann automatisch mit der Positionsangabe auf der dazugehörigen Kartenregion

Abspeichern der Koordinaten & Wiederaufruf sind möglich

Die App ist eine Portierung des „standalone“ widgets.

TopBar „Full“

Diese Topbar enthält die komplette Funktionalität, wie ich sie unter oTx genutzt hatte.



Zwei schmale Schaltflächen zeigen, ob ich snapflap bzw Flaperon Mischer (Wölkappen als Querruder) aktiviert habe

Eine grössere Schaltfläche signalisiert, ob mein Sicherheitsschalter den versehentlichen Anlauf des E-Motors unterbindet, wenn er aktiv (Motor „unscharf“) ist und der Motor-Input auf aus steht, ist die Anzeige grün. Sollte man auch bei unscharfen Sicherheitsschalter einen Motoranlauf gesetzt haben, blinkt die Fläche rot und gibt Warntöne aus (Sicherheitsrisiko, da bei deaktivieren des Schalters der Motor SOFORT anläuft)

Ist der Schalter deaktiviert, also der Motor „scharf“, wird eine Orange Darstellung gewählt.

Relativ Mittig steht der Modellname

Dann folgt eine Darstellung der Sender und Empfängerspannung. Die Symbole ändern sich je nach Spannungslage Grün auf Gelb nach Rot. Die Empfängerspannungsgrenzen sind modellspezifisch in einer Datei hinterlegbar.

Ganz rechts davon werden die VFR sowie RSSI Empfangsgüten dargestellt.

Das Symbol ändert die Farbe / den Status auf den „niedrigeren“ Status beider Sensoren.

Bsp: RSSI ist 55 (gelb), aber VFR 98 (grün) >> Status gelb

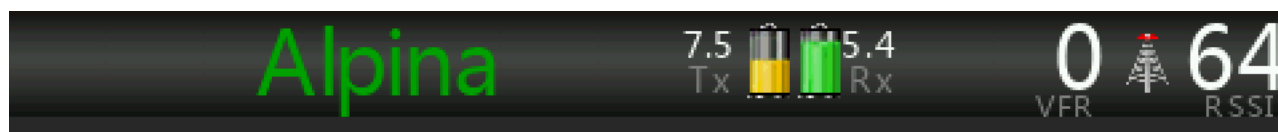
TopBar „Standard“

Wie TopBar „full“, jedoch ohne Mixer-Signalflächen bzw Schutzschalter Display, dafür mit zwei Timern



TopBar „light“

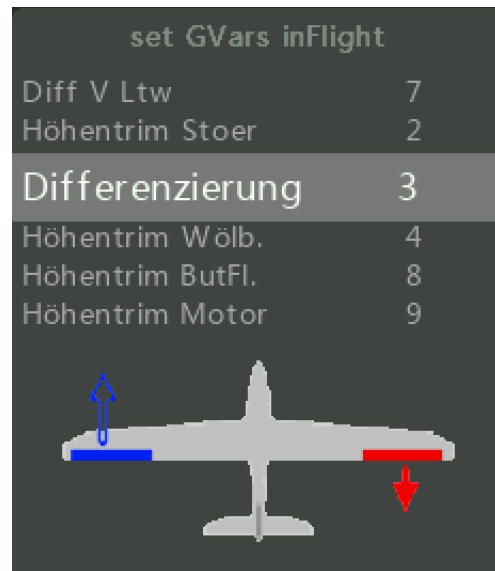
Eine sehr einfache Variante, lediglich mit Modellname, Akkuspannungen und Empfangsgüte



(Future: GVAR editor)

Dies ist ein Vorentwurf zur Anpassung von Globalen Variablen im Flug

(aufgrund Akustik feedback wird keine Notwendigkeit existieren dabei auf den Sender zu schauen).



Über einen Selektionsschalter (z.B. Trimmer / Tastschalter) wird die gewünschte Gvar angewählt, es erfolgt eine Durchsage der Selektion.

Dann kann via Poti der Wert innerhalb gewisser Grenzen (z.B. +/-20) angepasst werden, die Anpassungen werden sofort auf das Modell angewendet.

Ist der richtige Wert gesetzt, kann mit erneuter Wahl des Selektionsschalters der Wert final „gespeichert“ werden, die Anpassung via Poti wird dadurch geblockt.

Das Poti MUSS nun wieder in die Neutralposition zurückgesetzt werden (Piepston & Ansage „Go“ wenn erreicht)

Nun kann die nächste gewünschte Gvar selektiert werden.

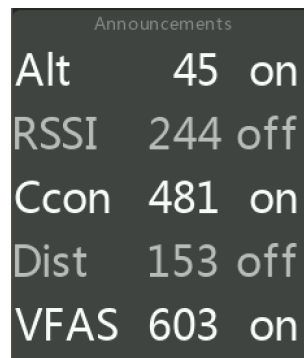
Ein „Resettaster“ ermöglicht das Rücksetzen auf die (zuvor intern gespeicherten) Ursprungswerte.

Dieses widget ist zunächst ein Preview und wird erst im Laufe von Ethos 1.5 fertiggestellt.

Noch sind nicht alle notwendigen Methoden unter lua Implementiert (Zugriff auf GVars)

(Future: Ansagen)

Dieser Vorentwurf generiert zyklische Ansagen von Sensorwerten.



Alt	45	on
RSSI	244	off
Ccon	481	on
Dist	153	off
VFAS	603	on

Die z.B. Spannungs- / Kapazitäts- / Höhen- / Entfernungsansagen etc.. in regelmässigen zeitlichen Abständen durchgeführt werden, ohne Logische Schalter bzw Spezialfunktionen konfigurieren zu müssen.

Welche Sensoren angesagt werden sollen bzw welche Zykluszeiten gewünscht sind lassen sich in der Konfig bzw per touch einstellen

Im oben gezeigten Beispiel steht der Sensor in der 1. Spalte, die Zykluszeit (Sekunden) in der 2.Spalte, ob die Ansage derzeit aktiv ist in der 3.Spalte

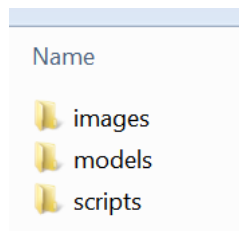
Idee ist per touch in das Statusfeld bzw das Zeitfeld den zustand / die Zykluszeit anpassen zu können.

Aus technischen Gründen kann die produktive Umsetzung erst in einem der 1.5er Ethos Versionen erfolgen.

Installation:

(1) Dateien & Ordner kopieren

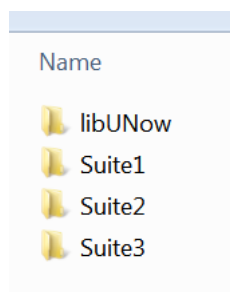
Der Download besteht (im git-Unterordner „src“) aus einer Reihe von Ordnern (sowie Unterordnern), die auf das entsprechende „Laufwerk“ des Senders kopiert werden müssen.



Im Folgenden wird das Haupt- bzw Wurzelverzeichnis des Senders mit „/“ gekennzeichnet

Bitte die Ordner wie durch die Downloadstruktur vorgegeben auf den Sender kopieren:

- „images“ in das Hauptverzeichnis
- alle Ordner unterhalb von „scripts“ in das „/scripts“ Verzeichnis des Senders
das sind die Ordner „libUNow“, „Suite1“, „Suite2“ sowie „Suite3“



- wer möchte kann sich aus dem Ordner „models“ die Modellvorlage „scale“ in das /models Verzeichnis kopieren, das template enthält eine durchkonfigurierte Beispieldvorlage
- Sender durchstarten, damit die scripte „bekannt“ gemacht werden

(2) Screenlayout auswählen

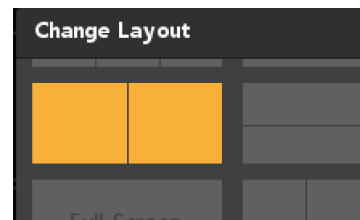
Wie üblich wird ein widget einem Frame / Rahmen unter Ethos zugeordnet.
Ethos bietet bekanntermassen eine Vielzahl verschiedener Layouts an.

Udo's Suite unterstützt folgende Layouts / Rahmen:

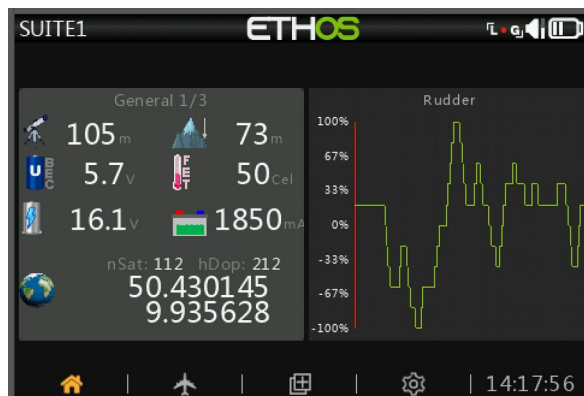
- **ein einzelner Frame**, in dem max 3 apps ausgewählt werden können

Hier wird das widget / der wrapper „SUITE1“ genutzt
Als Layout folgende Variante nutzen:

nur einer der beiden Frames darf durch den wrapper
genutzt werden !

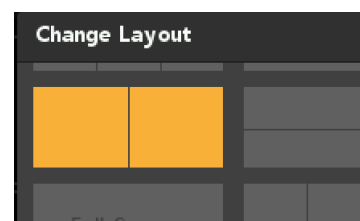


Das Ergebnis nach Konfiguration könnte dann so aussehen:



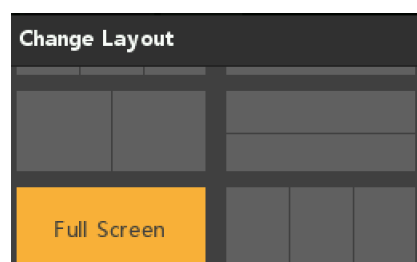
- **zwei Frames**, jeweils maximal 3 apps, Ethos original TopBar wird weiter genutzt

Hier wird das widget / der wrapper „SUITE2“ genutzt
Als Layout folgende Variante nutzen:



- **zwei Frames und Suite eigene TopBar**

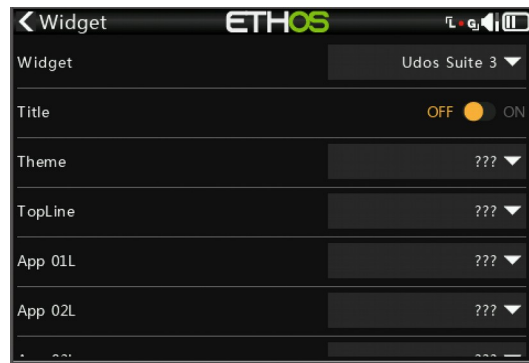
Hier wird das widget / der wrapper „SUITE3“ genutzt
Als Layout die VollBild Variante nutzen:



(3) widget allgemein einrichten

Über die Widgetkonfiguration lassen sich zwei Allgemeine Parameter sowie die einzelnen Apps konfigurieren
Um das widget zu aktivieren wird eine der drei Suite widgets ausgewählt (Udos Suite 1..3)

Es erscheint folgende Darstellung:



(1) nun bitte, wie im Bild gezeigt, den Titel abschalten

(2) danach wird das Farbschema / Thema ausgewählt; derzeit stehen „hell“ oder „dunkel“ zur Auswahl

(3) nun werden die einzelnen Apps ausgewählt

„TopLine“ stellt die Auswahl der TopBar Variante dar falls suite3 gewählt wurde

„App 01L..App03L“ entspricht der Platzierung im linken bzw Haupt-Frame

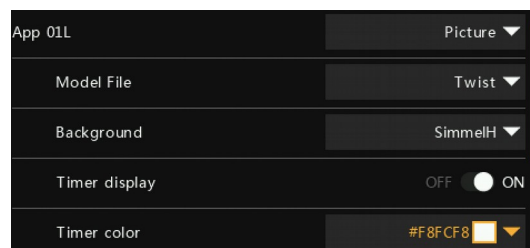
„App 01R..App03R“ sind die Apps 1-3 im rechten Frame (nur suite2 & suite 3)

Durch Klick auf die Auswahlbox erscheint die Auswahlliste der Apps



Eine App darf immer nur einmal pro Modell verwendet werden.

Sobald eine App ausgewählt wurde, erscheinen die dazu nötigen Konfigurationsparameter



Ist dies nicht der Fall, bitte kurz den Sender zur Initialisierung neu starten

(4) Konfiguration der einzelnen Apps

Telemetrie

App 01L	Tele01 ▼
DemoMode:	OFF <input type="radio"/> ON <input checked="" type="radio"/>
SensorSet:	Neuron & oXs ▼

Es existieren lediglich zwei Parameter

(1) Demomode

Aktiviert man den Demo Mode werden nicht die wirklichen Telemetrie/Sensor- Werte gelesen, sondern vordefinierte „Platzhalter Werte“. Dadurch erhält man einen Eindruck wie die Anzeige im Betrieb wirklich aussehen würde, wenn das Modell eingeschaltet ist. Der DemoMode wird häufig genutzt wenn man sich ein neues Set an TSensorwerten zusammenstellt

(2) Sensorset

Hier wählt man eines der vordefinierten Sensorsets aus.

Diese können für den jeweiligen Modeltyp passend individuell zusammengestellt werden (z.B. Segler, Elektrosegler, Motormodell, je mit/ohne GPS..). Durch die Vordefinition der Sets muss man nicht jedesmal die zur Darstellung gewünschten Sensoren pro Modell neu konfigurieren. Ein Sensorset kann Werte in 2..5 Zeilen besitzen und bis zu drei Seiten Umfang haben. Wie ein Set definiert wird ist im Anhang beschrieben

Picture

App 01R	Picture ▼
Model File	alpina2 ▼
Background	SimmelH ▼
Timer display	OFF <input type="radio"/> ON <input checked="" type="radio"/>
Timer color	#C8CCC8 <input type="color"/>

(1) Model File

Hier wird das Modell-Image (png) ausgewählt.

Das Image wird automatisch an die Framegrösse angepasst, man sollte es jedoch mit der (Bild-/Speichergrösse) nicht übertreiben.

(2) Background / Hintergrund

Ist das obige Modell Image freigestellt, kann ein dedizierter Hintergrund verwendet werden.

Das Hintergrundbild sollte so gewählt werden, dass es dem Höhen/Breite verhältnis des Frames entspricht, ansonsten wird auf einer Seite die Fläche nicht nicht ausgenutzt

(Ethos Issue zur Skalierung von images ist noch offen)

(3) Timer Display

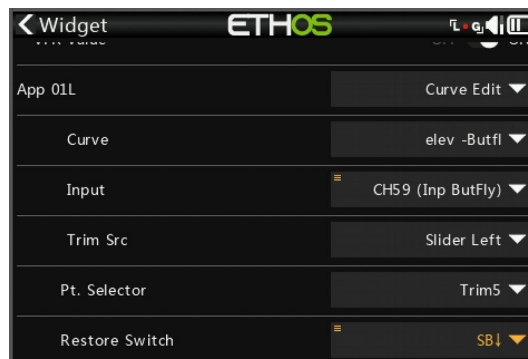
Wird dieser Parameter konfiguriert werden Timer1 und Timer2 in das Bild projiziert

(4) Timer Color

Eine pauschal „schwarze“ oder „weiße“ Schriftfarbe für die Timerdarstellung war nicht zielführend, da je nach Hintergrundfarbe kaum Kontrast vorhanden war.

Daher kann der Anwender hier selbst bestimmen, welche Farbe für die Timer genutzt werden soll

Kurveneditor



(1) Kurve/curve

wie der Name bereits sagt, wird hiermit die Kurve ausgewählt, die man im Flug nachtrimmen möchte

(2) Input

Die App hat grundsätzlich keine Kenntnis darüber, wo und wie die Kurve eingesetzt wird.

Daher muss man manuell den passenden Input suchen.

Bei einem HR Mischer, der den HR Knüppel als Input nutzt und lediglich im Mischer eine Kurve nutzt ist das trivialerweise der Input „Höhenruder“ der aus der „Analog“ Kategorie

So einfach ist das jedoch nicht immer.

Wird ein Input über mehrere Mischer „geleitet“ und am „letzten“ Mischer eine Kurve hinterlegt, kann man nicht einfach so den „ungefilterten“ analogen Eingang nutzen.

Aus diesem (und anderen) Gründen lege ich für die Hauptfunktionen weitestgehend dedizierte Mischerzeilen an, die den Input darstellen. Oben z.B. nutze ich den Mischer namens „Inp Butfly“ als „Geber“ für Butterfly.

Dieser Mischer (bei mit Ausgang auf Kanal 59) steuert dann die Flügelklappen in einem beispielsweise freien Mischer an, aber auch einen dedizierten Mischer für die Höhenruderbeimischung mit „eigener“ Kurve.

Diese Kurve (elev -ButFl) wird nun editiert, als Input dient also auch der „reale“ Geber

(3) Trim Src / Trim Quelle

Hier wird der (analoge) Geber gewählt, um die Y Koordinate eines Kurvenpunktes im Flug einzustellen

(4) Pt Selektor

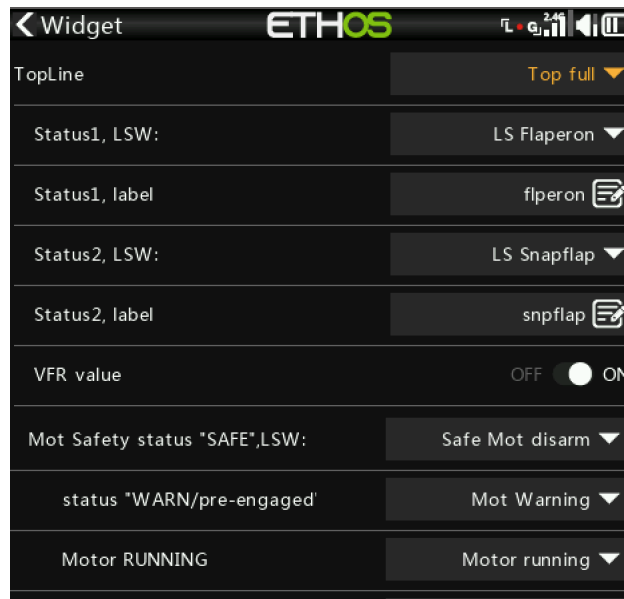
Damit wird der Geber gewählt, der die einzelnen Kurvenpunkte „anfährt“/selektiert.

Ich empfehle einen Trimmer zu verwenden

(5) restore Schalter / switch

Die dort enigestellte Schalterposition (oder LSW) triggert den reset, der zu Programmstart zwischengespeicherte Kurvenverlauf wird wiederhergestellt.

TopFull



Das „TopFull“ widget ist in der Lage den Zustand zweier logischer Schalter darzustellen, sowie besitzt es ein kleines Panel um vor unbeabsichtigten Motoranlauf zu schützen.

Dies alles wird hier konfiguriert.

(1) Status1, LSW

hier wird der erste logische Schalter gewählt, der dargestellt werden soll

(2) Status1, label

ein Textfeld, um der Schaltfläche einen individuellen „Namen“ zu vergeben

(3) Status2, LSW

siehe oben, für LSW2

(4) Status2, label

siehe oben, für LSW2

(5) VFR value

Nun zum „Sicherheits“ Panel (Motoranlauf)

Bei E-Segelfliegern ist es empfehlenswert und üblich einen „Motorschutzschalter“ bzw „Kill switch“ zu konfigurieren.

Dieser Kill switch überschreibt die Motor-Geber wie z.B. Gasknüppel oder -Schalter und „hält“ den Kanal auf den „ausgeschalteten“ Wert, so dass z.B. beim Einschalten des Empfängers nicht versehentlich der Motor anläuft, weil man an den Knüppel gekommen ist, oder aus anderen Gründen.

Es kann aber sein, dass der Kill switch auf „disarmed“ bzw „gesichert“ steht, das Gas bereits auf Vollgas, nur der Schutzschalter verhindert den Anlauf.

Sinn des Panels ist es, dann einen Alarm auszugeben, damit man die „Gasvorwahl“ in einen sicheren „Leerlauf“ Zustand bringt, bevor man den Schutz „abschaltet“

Als Voraussetzung müssen drei LSW definiert sein, die folgende Situationen darstellen

- Kill Switch ist aktiv und daher kein Motoranlauf möglich
- Geber / Input für Motor steht auf „Motor auf Leistung“ / Gas aktiviert, nur Kill switch verhindert Anlauf (also ohne aktivem kill switch dreht der Motor)
- Motor definitiv unter Last (z.B. Motorausgang nicht auf Leerlauf)

(6) Motor Safety Status „Safe“

Hier wird der LSW gewählt, der aktiv wird sobald der Kill Switch die Gaswahl abschaltet

(7) .. status „Warnung“

LSW der erkennt, das Gas gegeben wurde, aber Kill switch dies „zum Glück“ verhindert

(8) .. Motor Running

als Ergänzung, signalisiert, das der Motor anlaufen würde, falls Empfänger eingeschaltet wird

Anhang

Modellspezifische Konfiguration

Einige Darstellungen benötigen genauere modellspezifische Konfigurationen.

Zum Beispiel ist das am TopBar-Icon für die Empfängerspannung zu sehen.

Das Batterie-Icon ändert seine Farbgebung (Grün, Gelb, Rot) je nach Spannung.

Die Schwellwerte könnte man individuell in der widget Konfiguration vornehmen, wenn man alle derartigen Parameter jedoch dort erfassen möchte, kann dies zu einer „grösseren Aktion“ ausarten.

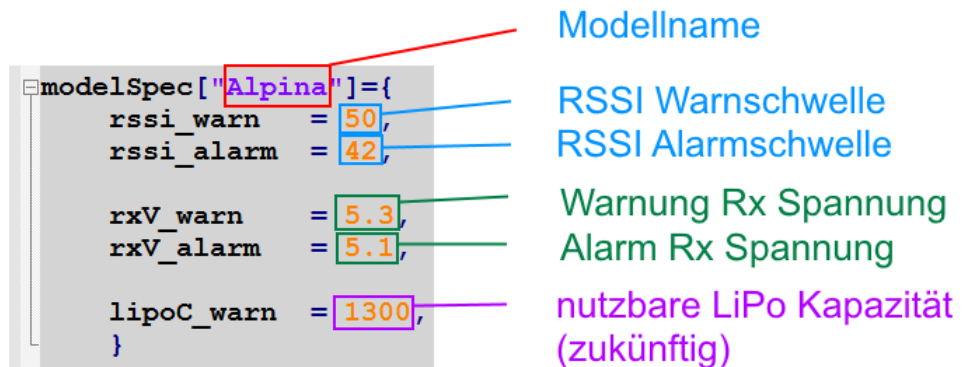
Daher wird die Möglichkeit unterstützt vom „Standard“ abweichende Werte in Form einer individuellen Datei zu hinterlegen.

Diese Datei ist die „**lib_modelSpecs**“ im Verzeichnis **\\scripts\\libUNow**

Dort können für „spezielle“ Modelle die individuellen Parameter hinterlegt werden

Derzeit stehen fünf Pflegeparameter zur Verfügung

- rssi_warn Schwelle, ab der das icon für reduzierten Empfang signalisiert wird (wenn nicht access)
- rssi_alarm Schwelle, ab der das icon für sehr schwachen Empfang signalisiert wird
- rxV_warn Warnschwelle, „gelbes“ Akku-icon
- rxV_alarm Warnschwelle, „rotes“ Akku-icon
- lipoC_warn (für zukünftige Nutzung): wieviel tatsächlich nutzbare Kapazität im Antrieb



Sollten bestimmte Parameter nicht nötig sein (z.B. es sollen nur die RSSI Schwellen individualisiert werden) ist das „Wort“ nil als Wert einzutragen, Beispiele dazu in der Datei

Standardsettings der Apps

Wer die suite häufiger nutzt wird feststellen, dass viele Modelle im Grunde einheitliche SetUps mit nur kleinen Varianten benötigen.

Idealerweise würde eine Vorbelegung der Apps den Aufwand für die eigentliche Parametrisierung deutlich reduzieren.

Eine Standard-Vorbelegung der Felder kann für jede App spezifisch in der Datei

\\scripts\\libUNow\\suite\\suite_conf.lua

vorgenommen werden.

Es sollten grundlegende lua Kenntnisse vorhanden sein.

Die Funktion „**getSubForm(index,txt,lang)**“ enthält die Beschreibung aller App-Konfigurationsparameter (der jeweiligen „lua Form“)

Man sucht die „if Bedingung“ der entsprechende App (hier setcurve) und kann im default Parameter einer Zeile die entsprechenden Standard Feldeinträge vornehmen

Mit ein wenig lua (oder anderer Programmier-) Kenntnissen sollte die sich die Logik aus den vorhandenen Beispielen ergeben.

```
elseif index == w_SETCURVE then
    subFrm= {
        {txt.conf[1][lang], "createCurveChoice", nil, 1, {{dummy.1},{dummy.2}}, default=1 },
        {txt.conf[2][lang], "createSourceField", nil, 1, {{dummy.1},{dummy.2}}, default= system.getSource({name="Throttle", category = CATEGORY_ANALOG}) },
        {txt.conf[3][lang], "createChoiceField", nil, 1, {{Pot1},{Pot2},{Pot3},{Pot4},{Slider Left},{Slider Right}}, default= },
        {txt.conf[4][lang], "createChoiceField", nil, 1, {{SH},{SI},{S3},{Trim5},{Trim6}}, default= },
        {txt.conf[5][lang], "createSwitchField", nil, 1, default= system.getSource({member = 5, category = CATEGORY_SWITCH_POSITION}) }
```

Der „Telemetrikatalog“

Unter Ethos-lua funktioniert das Auslesen eines Telemetriesensors über zwei Schritte

Zunächst wird ein „Quellobjekt“ mittels Namen angelegt (jeder Sensor hat einen eindeutigen Namen).

Diesem Namen muss noch die Kategorie „Telemetrie“ mitgeteilt werden, da ja ggf auch ein LSW o.a. gleichen Namens existieren könnte.

Sollte das Objekt mehrere Werte liefern können, müssen noch Optionen angegeben werden (z.B. Latitude oder Longitude Werte bei GPS)

Um das widget / die suite möglichst Universal zu halten, werden Telemetriewerte nicht „direkt“ abgefragt, sondern die einzelnen Quellen werden anhand eines Katalogs ermittelt.

Dieser Katalog enthält dann noch für die suite notwendige, weitere Parameter.

Dies ermöglicht auch später eine einfache Konfiguration von settings (siehe telemetry widget)

Der Katalog befindet in der Datei

`\scripts\libUNow\widgets\teleglobal\sensordlist.lua`

```
function defineSensors(widget)
    local midi = -35
    local midi = -20
    local bmpPath = "/scripts/libUNow/bmp/"
    local frpPath = "/scripts/libUNow/freepic/"

    local sensors = {
        -- Tx
        [{"TxRt"} = {name = "TxRt", icon = ("Batt3.png"), path=bmpPath, options = nil, f_lenght=5, dec=1, alignV=-20, alignB=-5, zh=true, mkSens=true, testVal = 8.8, testUnit = "V"},
        -- Rx
        [{"rssi"} = {name = "RSSI", icon = ("rssi.png"), path=bmpPath, options = nil, f_lenght=5, dec=0, alignV= midi, alignB=0, zh=false, mkSens=true, testVal = 85, testUnit = "" },
        [{"vvr"} = {name = "VVR", icon = ("vvr.png"), path=frpPath, options = nil, f_lenght=5, dec=0, alignV= midi, alignB=0, zh=false, mkSens=true, testVal = 98, testUnit = "" },
        [{"smr"} = {name = "SMR", icon = ("Ant_alarm.png"), path=bmpPath, options = nil, f_lenght=5, dec=0, alignV= midi, alignB=0, zh=false, mkSens=true, testVal = 98, testUnit = "" },
        [{"RxBt"} = {name = "RxBt", icon = ("Batt3.png"), path=bmpPath, options = nil, f_lenght=5, dec=1, alignV= midi, alignB=0, zh=false, mkSens=true, testVal = 5.05, testUnit = "V" },
        [{"RxBt"} = {name = "RxBt", icon = ("Ant_ok.png"), path=bmpPath, options = nil, f_lenght=5, dec=1, alignV= midi, alignB=0, zh=false, mkSens=true, testVal = 0, testUnit = "" },
        -- GPS
        [{"gpsLat"} = {name = "GPS", icon = ("earth.png"), path=frpPath, options = OPTION_LATITUDE, f_lenght=5, dec=6, alignV= 0, alignB=0, zh=false, mkSens=true, testVal = 50.430145, testUnit = "" },
        [{"gpslon"} = {name = "GPS", icon = ("earth.png"), path=frpPath, options = OPTION_LONGITUDE, f_lenght=5, dec=6, alignV= 0, alignB=0, zh=false, mkSens=true, testVal = 9.935628, testUnit = "" },
        [{"GPS"} = {name = "GPS", icon = ("earth.png"), path=frpPath, options = OPTION_LATITUDE, f_lenght=5, dec=6, alignV= 0, alignB=0, zh=true, mkSens=true, testVal = 50.430145, testUnit = "" },
        [{"GPS"} = {name = "GPS", icon = ("earth.png"), path=frpPath, options = OPTION_LONGITUDE, f_lenght=5, dec=6, alignV= 0, alignB=0, zh=true, mkSens=true, testVal = 9.935628, testUnit = "" },
        [{"GAlt"} = {name = "GPS Alt", icon = ("alti.png"), path=bmpPath, options = nil, f_lenght=5, dec=0, alignV= midi, alignB=0, zh=false, mkSens=true, testVal = 830, testUnit = "m" },
    }
```

Das widget kann nur Werte darstellen, die auch im Katalog „bekannt“ gemacht wurden und dadurch eine interne Bezeichnung erhielten.

Selbst Sensoren „gleicher“ Gattung können je nach Hersteller unterschiedliche Namen in der Sensorsuche erhalten und müssen daher einzeln aufgeführt werden (z.B. ESC's FrSky <> YGE etc..)

Es folgt eine Liste der einzelnen Parameter

name	der Ethos Sensorname
icon	Filename des dazugehörigen icons
path	Pfad des Iconfiles
options	Options Parameter
f_lenght	Feldlänge
dec	Nachkommastellen
alignV	vertikaler Versatz, mit dem das Icon dargestellt wird
alignB	horizontaler Versatz, mit dem das Icon dargestellt wird
xh	exception handling in telemetry library
mkSens	aktiviere Ethos Source definition (einmal bei erstem Aufruf)
testval	Testwert im Simulationsmodus
testUnit	Testeinheit im Simulationsmodus

Landesspezifische Texte / Sprachdateien

Die suite ermittelt die Spracheinstellung des Senders und kann, falls gepflegt, Texte landesspezifisch ausgeben.

Dezeit ist Deutsch und englisch gepflegt.

Nicht gepflegte Sprachen werden automatisch mit den englischen Texten ausgegeben.

Das suite Hauptprogramm (main.lua) sowie die einzelnen Apps besitzen jeweils eigene Sprachdateien.

Diese Sprachdateien enthalten die Texte der jeweiligen Applikation sowie die des Konfig-Formulars.

Das Hauptprogramm wird durch die Datei „\scripts\libUNow\suite\suite_lang.lua“ bedient.

Einzelne Apps laden die Sprachdateien aus dem Ordner „\scripts\libUNow\widgets\appname“

Filename lautet „lang.lua“

Beispiel „Modelfinder“, Datei „\scripts\libUNow\Modelfind\lang.lua“

```
local txt_Fields = {
    --
    --          de          en
    modefile   = {"Modus: File",          "Mode: File"          }, -- 1
    refresh    = {"refresh in:",          "refresh in:"         }, -- 2
    noloack    = {"kein GPS lock",         " no Lock, sorry"     }, -- 3
    lastcoord   = {"letzte Koord.:",       "Last Coord:"         }, -- 4
    waiting    = {"Kalkuliere QR !",       "Processing!"         }, -- 5

    -- config Form
    conf       = {
        {
            "          Test Modus",          "          Test mode"
        }
    }
}
```

Im oberen Textblock sind in Spalte 1 die deutschsprachigen, in Spalte 2 die englischen Texte gepflegt.

Da diese App nur eine Konfig Zeile besitzt, ist hier auch nur eine Eintrag zu sehen (Test Modus)

Dem Anwender wird dadurch eine einfache Möglichkeit gegeben Texte individuell anzupassen.

Soll eine weitere Sprache bedient werden, müssten alle Sprachdateien um die entsprechende Spalte ergänzt werden.

Im Hauptprogramm müsste in der Funktion getLang für diese (dritte) Spalte eine weitere else Unterscheidung mit Rückgabewert „return 3“ eingebunden werden

```
311
312 local function getLang()
313     if system.getLocale() == "de" then
314         return 1
315     else
316         return 2
317     end
318 end
319
```

Beispieltemplate „Alpina“

Internals

Für diejenigen, die etwas mehr Interesse an Lua Programmierung besitzen, ein paar Erklärungen:

Rev 0.8
unow, Aug 2023