

# **DCS Export Script**

Version 1.0.0

Copyright by Michael aka McMicha 2016

Die Software unterliegt der LGPL Version 3

# Inhaltsverzeichnis

Hinweis.....	3
Installation.....	4
Konfigurieren.....	5
ExportsModules.....	8
A-10C.lua.....	8
Su-25T.lua.....	9
Generisches Funkgerät für die DCS Module.....	11
Hilfsfunktionen.....	13
Fehlermeldung.....	14
Informationen.....	15
Erstellen einer Exportdatei für ein DCS Modul am Beispiel der A-10C.....	16
mainpanel_init.lua.....	16
devices.lua.....	16
clickabledata.lua.....	17
Welche Daten kommen nun wohin?.....	19
Spezial Funktionen der einzelnen Devices.....	21

## Hinweis

Dies ist ein universell einsetzbares Export Script für DCS. Es wird der gleichzeitige Export von Daten an Glasscockpitsoftware und I/O Hardware ermöglicht.

Zur Zeit werden folgende Export-Formate unterstützt.

- D.A.C. (DCS Arcaze Connector) ([DAC in GitHub](https://github.com/DAC-in-GitHub)) um die Arcaze USB Controller anzusprechen (<http://wiki.simple-solutions.de/de/products/Arcaze/Arcaze-USB>)
- Ikarus von [H-J-P](#), unsere neue Glasscockpitsoftware

und andere Software/Hardware die ihre Daten über eine UDP Netzwerkverbindung bekommen und die Daten in einem Key=Value Format verarbeiten.

Das ExportScript Paket ist zwei grobe Bereiche unterteilt, die Programmlogik und die Modul spezifischen Logik. Die Skripte „Config.lua“ und „ExportScript.lua“ unter dem Pfad „%USERPROFILE%\Saved Games\DCS\Scripts\DCS-ExportScript\“

`C:\Users\<USER>\Saved Games\DCS\Scripts\DCS-ExportScript\`

und die Skripte „Tools.lua“ und „genericRadio.lua“ in dem dortigen Unterordner „lib“ sind für die ganze Steuerung und Logik des Datenexportes zuständig. Die Modul spezifischen Skripte befinden sich in dem Ordner „ExportsModules“ im Pfad „%USERPROFILE%\Saved Games\DCS\Scripts\DCS-ExportScript\ExportsModules\“

`C:\Users\<USER>\Saved Games\DCS\Scripts\DCS-ExportScript\ExportsModules\`

In der Datei „Config.lua“ werden die Grundsätzlichen Einstellungen getätigt.

Die zu exportierenden Module lassen sich recht einfach erweitern. Dazu müssen nur alle Informationen über die zu exportierenden Daten in einer neuen Datei geschrieben werden. Diese Datei muss unter dem Namen des entsprechenden Moduls im Ordner „ExportsModules“ gespeichert werden. Wie die Dateien aufgebaut sein müssen steht weiter unten, oder ist in den bereits vorhandenen Dateien dokumentiert.

Das Export Script unterstützt den Einzel- und Mehrspieler-Modus, sowie das Wechseln der Flugzeuge während die Simulation läuft (RAlt + J).

# Installation

**WICHTIG:** Von eventuell vorhandenen Dateien bitte vorher eine Sicherheitskopie machen.

Temporäres Entpacken des Zip-Archiv auf dem Desktop.

Kopieren des enthaltenen Ordners „Scripts“ in den Dateipfad „%USERPROFILE%\Saved Games\DCS\“

C:\Users\<USER>\Saved Games\DCS\

Falls bereits eine „Export.lua“ Datei einer anderen Software vorhanden ist, müssen am Ende dieser Datei die folgenden Zeilen angefügt werden.

```
-- load the DCS ExportScript for DAC and Ikarus  
dofile(lfs.writedir()..[[Scripts\DCS-ExportScript\ExportScript.lua]])
```

# Konfigurieren

**WICHTIG:** Zum Editieren der LUA Dateien bitte einen vernünftigen Editor benutzen, z.B. [Notepad++](#)

In der Datei „Config.lua“ befinden sich die Grundlegenden Einstellungen für den Export.

Default Einstellungen:

```
ExportScript.Config          = {}

-- Ikarus a Glass Cockpit Software
ExportScript.Config.IkarusExport    = true -- false for not use
ExportScript.Config.IkarusHost      = "127.0.0.1" -- IP for Ikarus
ExportScript.Config.IkarusPort      = 1625 -- Port Ikarus (1625)
ExportScript.Config.IkarusSeparator = ":"

-- D.A.C. (DCS Arcaze Connector)
ExportScript.Config.DACExport      = true -- false for not use
ExportScript.Config.DAC            = {}
-- first hardware
ExportScript.Config.DAC[1]         = {}
ExportScript.Config.DAC[1].Host    = "127.0.0.1" -- IP for hardware 1
ExportScript.Config.DAC[1].SendPort = 26026 -- Port for hardware 1
ExportScript.Config.DAC[1].Separator = ":"
-- second to n hardware
--ExportScript.Config.DAC[2]        = {}
--ExportScript.Config.DAC[2].Host    = "127.0.0.1" -- IP for hardware 2
--ExportScript.Config.DAC[2].SendPort = 9092 -- Port for hardware 2
--ExportScript.Config.DAC[2].Separator = ":"

-- Ikarus and D.A.C. can data send
ExportScript.Config.Listener        = true          -- false for not use
ExportScript.Config.ListenerPort    = 26027         -- Listener Port for D.A.C.

-- Other
ExportScript.Config.ExportInterval  = 0.05 -- export evry 0.05 seconds
ExportScript.Config.ExportLowTickInterval = 0.5 -- export evry 0.5 seconds
ExportScript.Config.LogPath         = lfs.writedir()..[[Logs\Export.log]]
ExportScript.Config.ExportModulePath = lfs.writedir()..[[Scripts\DCS-
ExportScript\ExportsModules\]]
ExportScript.Config.Debug            = false
ExportScript.Config.FirstNewDataSend = true
ExportScript.Config.FirstNewDataSendCount = 5
```

Der Konfigurationsblock ist in 4 Bereiche unterteilt.

Der erste Bereich ist für die Software „Ikarus“ vorgesehen.

Die Variable „ExportScript.Config.IkarusExport“ gibt an, ob Daten an Ikarus exportiert werden sollen. Hier kann „true“ oder „false“ angegeben werden.

Die Variable „ExportScript.Config.IkarusHost“ gibt an, an welche IP Adresse die Daten geschickt werden sollen. Hier kann die IP Adresse „127.0.0.1“ (für localhost) oder eine andere gültige IP Adresse des Zielrechners angegeben werden.

Die Variable „ExportScript.Config.IkarusPort“ gibt an, an welchen Port die Daten geschickt werden sollen. Der Port „1625“ ist der Default-Port von Ikarus und brauch nicht geändert werden. Wird der Port trotzdem geändert, muss dies auch innerhalb der Ikarus Konfiguration getan werden.

Der zweite Bereich ist für die Software D.A.C. vorgesehen, an die Daten exportiert werden sollen. Es ist mögliche mehrer IP Adressen und Ports für unterschiedliche Installationen der Software anzugeben.

Die Variable „ExportScript.Config.DACExport“ gibt an ob Daten an (alle) D.A.C. Installationen exportiert werden sollen. Hier kann „true“ oder „false“ angegeben werden.

Die Zeilen

```
ExportScript.Config.DAC[1]          = {}  
ExportScript.Config.DAC[1].Host      = "127.0.0.1" -- IP for hardware 1  
ExportScript.Config.DAC[1].SendPort  = 26026 -- Port for hardware 1  
ExportScript.Config.DAC[1].Separator = ":"
```

enthalten die Daten für den ersten Hardware Export.

Die Zeile „ExportScript.Config.DAC[1] = {}“ darf nicht verändert werden.

Die Variable „ExportScript.Config.DAC[1].Host“ gibt an, an welche IP Adresse die Daten geschickt werden. Hier kann die IP Adresse „127.0.0.1“ (für localhost) oder eine andere gültige IP Adresse des Zielrechners angegeben werden.

Die Variable „ExportScript.Config.DAC[1].SendPort“ gibt an, an welchen Port die Daten geschickt werden sollen. Der Port „26026“ ist der Default-Port von D.A.C. und braucht nicht geändert werden. Wird der Port trotzdem geändert, muss dies auch innerhalb der D.A.C. Konfiguration getan werden.

Die Variable „ExportScript.Config.DAC[1].Separator“ gibt das Trennzeichen zwischen den einzelnen Key=Value Paaren an. Für D.A.C. ist das Default-Trennzeichen der Doppelpunkt „:“, ansonsten sind alle Zeichen außer das Gleichzeichen „=“ erlaubt.

Für einen weiteren Hardware Export an einem anderen Computer werden diese 4 Zeilen noch einmal unter den bereits vorhanden eingetragen und um die jeweiligen Daten ergänzt.

**WICHTIG:** Wichtig dabei ist, dass die Zahl in den Eckigen Klammern durch die nächst höhere ersetzt wird.

Beispiel:

```
ExportScript.Config.DAC[2]          = {}  
ExportScript.Config.DAC[2].Host      = "192.168.0.14" -- IP for hardware 2  
ExportScript.Config.DAC[2].SendPort  = 9090 -- Port for hardware 2  
ExportScript.Config.DAC[2].Separator = ":"
```

Auf dieser Art und Weise können fast unbegrenzt viele verschiedene Hardware/Software Kombinationen angesprochen werden.

Der dritte Bereich betrifft den Daten Empfang von Ikarus und D.A.C.

Die Variabel „ExportScript.Config.Listener“ gibt an ob Daten empfangen und verarbeitet werden sollen. Mögliche Werte sind „true“ und „false“.

Die Variable „ExportScript.Config.ListenerPort“ gibt den Port an, auf dem das Export Script die Daten von Ikarus/D.A.C. empfängt. Der Port „26027“ ist der Default-Port von Ikarus/D.A.C. und braucht nicht geändert werden.

Der vierte Bereich gibt allgemeine Werte vor und braucht in der Regel nicht geändert werden.

Die Default-Werte sind wie folgt:

```
ExportScript.Config.ExportInterval    = 0.05    -- export evry 0.05 seconds
```

```
ExportScript.Config.ExportLowTickInterval = 0.5    -- export evry 0.5 seconds
ExportScript.Config.LogPath               = lfs.writedir()..[[Logs\Export.log]]
ExportScript.Config.ExportModulePath      = lfs.writedir()..[[Scripts\DCS-
ExportScript\ExportsModules\]]
ExportScript.Config.Debug                 = false
ExportScript.Config.FirstNewDataSend      = true
ExportScript.Config.FirstNewDataSendCount = 5
```

Und die einzelnen Werte bedeuten folgendes:

ExportScript.Config.ExportInterval: gibt das Zeitintervall in Sekunden an, in der die Zeitkritischen Daten aktualisiert werden (Default 0.05)

ExportScript.Config.ExportLowTickInterval: gibt das Zeitintervall in Sekunden an, in der die anderen Daten aktualisiert werden (Default 0.5).

ExportScript.Config.LogPath: gibt den Pfad für die Log-Datei an

ExportScript.Config.ExportModulePath: gibt den Pfad zu den Modul-Definitionen an.

ExportScript.Config.Debug: gibt an ob Daten geloggt werden sollen, Daten werden durch die Funktion „WriteToLog()“ in die Log-Datei geschrieben.

ExportScript.Config.FirstNewDataSend: gibt an ob alle Export-Daten nach dem Start der Simulation noch einmal neu versendet werden sollen, dies ist wichtig falls nach dem Start keine Initialdaten vorliegen.

ExportScript.Config.FirstNewDataSendCount: gibt an ab wann die Export-Daten noch einmal gesendet werden sollen, der Wert ist keine Zeitangabe sondern ein Zähler der zählt wie oft das Export-Script Daten geliefert hat.

**WICHTIG:** Alle Angaben bei der Konfiguration müssen in dem Vorgegeben Format angegeben werden.

# ExportsModules

Der Ordner „ExportsModules“ enthält die einzelnen Modul spezifischen Dateien, z.B. A-10C.lua oder Su-25T.lua .

In diesen Dateien sind alle Exportierbaren Daten der jeweiligen Module aufgeführt.

Hier ein paar exemplarische Beispiele für ein DCS und ein Flaming Cliffs Modul.

## A-10C.lua

Die A-10C ist ein voll durch simuliertes DCS Modul. Diese Module beinhalten sehr genau simulierte einzelne Systeme und ein anklickbares Cockpit. Daraus resultiert, dass bei diesen Modulen sämtliche Informationen einzelner Geräte oder Anzeigen ausgegeben werden können.

Grundsätzlich sind die Dateien für die DCS Module in mehrer Blöcken unterteilt.

```
ExportScript.ConfigEveryFrameArguments = {}  
ExportScript.ConfigArguments = {}  
ExportScript.ProcessIkarusDCSConfigHighImportance()  
ExportScript.ProcessIkarusDCSConfigLowImportance()  
ExportScript.ProcessDACConfigHighImportance()  
ExportScript.ProcessDACConfigLowImportance()
```

Die ersten beiden Blöcke sind Variablen und enthalten die Informationen welche Daten generell exportiert werden sollen.

Der Inhalt der Variablen ist wie folgt aufgebaut:

```
[4] = "%.4f",          -- AOA
```

DCS ID: 4 Ausgabe Format: “%.4f” bedeutet eine Fließkommazahl mit 4 Nachkommastellen Als Kommentar: AOA (als Beschreibung was der Wert für Daten angibt)

Unter der ID 4 ist bei der A-10C der AOA Wert gelistet. Dieser Wert wird als Fließkommazahl mit 4 Nachkommastellen ausgegeben. Bei der Software kommt das ganze also als „4=0.5486“ an.

Die Variable „ExportScript.ConfigEveryFrameArguments“ enthält die Informationen zu den Zeitkritischen Werten, z.B. die Werte der Instrumente und Statuslampen.

Die Variable „ExportScript.ConfigArguments“ enthält die Informationen zu den sonstigen Werten, z.B. Schalterstellungen.

Die beiden Funktionen „ExportScript.ProcessIkarusDCSConfigHighImportance()“ und „ExportScript.ProcessIkarusDCSConfigLowImportance()“ können Informationen zu Werten enthalten die erst errechnet oder mit speziellen Funktionen ermittelt werden. Die hier ermittelten Werte werden an Ikarus ausgegeben.

Zum Beispiel werden hier die Funkfrequenzen ermittelt.

```
local lUHFRadio = GetDevice(54)  
ExportScript.Tools.SendData(2002, string.format("%.3f",  
lUHFRadio:get_frequency()/1000000))
```

Die erste Zeile erzeugt eine lokale Variable mit den Informationen des Gerätes mit der ID 54, dies ist in



der A-10C das UHF Radio. Die Device ID stehen in der „devices.lua“ Datei, im Pfad „...\\Eagle Dynamics\\DCS World\\Mods\\aircraft\\A-10C\\Cockpit\\Scripts\\“

In der zweiten Zeile werden die Daten ausgelesen, umgerechnet, formatiert und mit der Funktion `ExportScript.Tools.SendData()` an Ikarus übertragen.

`ExportScript.Tools.SendDaten()`: ist die Funktion, mit der die Daten an Ikarus übertragen werden.

2002: ist der Key unter dem Ikarus die Daten empfängt/erwartet

`string.format()`: ist eine LUA Funktion mit der Daten Formatiert werden können (mehr dazu in der offiziellen [LUA Dokumentation](#) )

„%7.3f“: der angegebene Wert wird als Fließkommazahl mit 7 Stellen vor und drei Stellen nach dem Komma ausgegeben.

`IUHFRadio:get_frequency()`: es wird die Spezial Funktion `get_frequency()` des UHF Radio Devices aufgerufen, diese Funktion gibt die aktuell eingestellte Frequenz des Funkgerätes als Dezimalzahl in Hz zurück

`/1000000`: die Frequenz des Funkgerätes wird durch 1000000 geteilt um die Frequenz in MHz zu ermitteln

Die Funktion „`ExportScript.ProcessIkarusDCSConfigHighImportance()`“ wird öfters aufgerufen als die Funktion „`ExportScript.ProcessIkarusDCSConfigLowImportance()`“ und eignet sich daher eher für Zeitkritische Werte.

Die beiden Funktionen „`ExportScript.ProcessDACConfigHighImportance()`“ und „`ExportScript.ProcessDACConfigLowImportance()`“ entsprechen den beiden oben genannten Funktionen und sind für die Ausgabe an die Hardware zuständig.

Hier werden die entsprechenden Daten aus dem Simulator ausgelesen, gegebenenfalls noch einmal aufgearbeitet und dann an die Hardware ausgegeben.

Alle hier bereits eingetragenen Datenabfragen/Ausgaben werden an alle, in der „Config.lua“ Datei eingetragene DAC Installationen gesendet. Hier werden nur Daten ausgegeben, die mit einem Arcaze USB Controller darstellbar sind. Dies sind LEDs und 7 Segment-Displays.

Das verarbeiten der Daten erfolgt wie oben beschrieben. Die Daten werden aber über die Funktion „`ExportScript.Tools.SendDataDAC()`“ exportiert. Alle in der „Config.lua“ Datei eingetragenen Ziel IP/Ports bekommen die selben Daten geliefert.

## Su-25T.lua

Die Su-25T entspricht einem Flaming Cliffs Modul und wird hier stellvertretend für alle bereits verfügbaren Flaming Cliffs Flugzeuge beschrieben.

Die Flaming Cliffs Flugzeuge unterscheiden sich nicht nur in der Simulationstiefe und dem Handling von den vollwertigen DCS Modulen, sondern auch in dem Umfang der Daten die von diesen Flugzeugen exportiert werden können.

Aus diesem Grund sind die Dateien für die Flaming Cliffs Flugzeuge anders aufgebaut als z.B. die Datei der A-10C.

Die Su-25T Datei besteht aus 4 großen Blöcken und diversen Hilfsfunktionen.

```
ExportScript.ProcessIkarusFCHighImportanceConfig()  
ExportScript.ProcessDACConfigHighImportance()  
ExportScript.ProcessIkarusFCLowImportanceConfig()  
ExportScript.ProcessDACConfigLowImportance()
```

Die ersten beiden Funktionen enthalten Informationen zu den Daten die an Ikarus geschickt werden.

Diese Werte werden über DCS eigene Funktionen ermittelt, gegebenenfalls noch umgerechnet und dann über Ikarus ausgegeben.

Die beiden Funktionen „ExportScript.ProcessIkarusFCLowImportanceConfig()“ und „ExportScript.ProcessDACConfigLowImportance()“ enthalten die Informationen zu den Daten die über D.A.C. ausgegeben werden.

Die Daten werden von den DCS eigenen Funktionen in Funktionsgruppen oder nach Geräten ausgegeben. Aus diesem Grund wurden einzelne Geräte spezifische Funktionen erzeugt die die jeweiligen Daten der Flugzeuge aufarbeiten und ausgeben.

Alle Funktionen/Geräte die von mehr als einem Flugzeug benutzt werden befinden sich in der Datei FC\_AuxiliaryFuntions.lua.

Zum Beispiel gibt die Funktion „ExportScript.AF.FC\_SPO15RWR()“ die Informationen des SPO-15 Radar Warnsystems zurück. Mit Hilfe dieser Informationen ist es möglich ein authentisches SPO-15 nachzubauen.

Viele Geräte spezifische Funktionen haben einen optionalen Parameter, nämlich den Funktionstyp, der angibt ob die Daten für Ikarus oder D.A.C. bestimmt sind.

Wie auch bei den DCS Modulen sind hier bereits alle Datenabfrage hinterlegt die mit Hilfe von DAC an den Arcaze USB Controller übertragen werden können.

Alle Flaming Cliffs Dateien sind ähnlich aufgebaut und ausreichend Kommentiert.

Leider besitzen die US Flugzeuge aus Flaming Cliffs viele Anzeigen mit Display, sodass nicht so viele Daten wie bei den Russischen Flugzeugen über die Hardware ausgegeben werden können. Als Beispiel sei hier das Radar Warn-System genannt, das in den US Flugzeugen die Informationen auf eine Art Monitor darstellt.

# Generisches Funkgerät für die DCS Module

Für einige der DCS Module existiert ein „Generisches Funkgerät“ (genericRadio) in Form eines speziellen Funktion die innerhalb des Export Scriptes konfiguriert und aufgerufen wird.

Mit Hilfe des Generischen Funkgerätes lassen sich die verschiedenen Funkgeräte eines Flugzeuges über eine Hardware bedienen.

Diese Funktion arbeitet ausschließlich mit D.A.C. zusammen, da hier einige spezielle Funktionen von D.A.C. und des Arcaze USB Controllers benutzt werden.

Folgende Hardware wird vorausgesetzt:

- 1 x Arcaze USB-Controller
- 1 x Display Driver 3 oder 32
- 1 x 8fach Display Board mit 8 Displays (7 Segmentanzeigen) oder 1 x 2fach und 1 x 6fach Display Board mit 8 Displays
- 1 x Stufen-Dreheschalter mit mindestens 3 Positionen
- 4 x Taster
- 4 x Drehencoder

Optional

- 4 x Low current LED

Über den Dreheschalter wird das Funkgerät ausgewählt welches dargestellt und/oder konfiguriert werden soll.

Einer der Taster ist zum aktivieren/deaktivieren des Funkgerätes. In der Regel wird darüber die Stromversorgung eingeschaltet.

Auf dem Display wird mit den ersten beiden Stellen der Preset Channel angezeigt (sofern vorhanden), auf den restlichen 6 Anzeigen erscheint die Frequenz.

Sofern das Funkgerät die Möglichkeit hat zwischen einer Manuellen Frequenzeingabe und der Auswahl eines Preset Channels zu wählen, lässt sich über eine weitere Taste dazwischen umschalten. Wurde die Preset Channel Einstellung aktiviert, erscheint im Display die jeweilige Frequenz des Preset Channels (dies entspricht nicht der Anzeige in der Simulation).

Die dritte Taste ist für die Aktivierung der Rauschunterdrückung (Squelch) da.

Die vierte Taste ist zum laden oder speichern von Preset Channels da (abhängig vom Modul und Funkgerät).

Ein Drehencoder stellt den Preset Channel ein.

Zwei Drehencoder sind für die Frequenz Einstellung zuständig, wobei der eine die Ziffern vor dem Komma und der andere die Ziffern nach dem Komma einstellt.

Der vierte Drehencoder ist für die Lautstärkeregelung zuständig.

Über die vier LEDs kann der Power, der Preset, der Squelch und gegebenenfalls das Laden des Presets Status des Funkgerätes angezeigt werden.

Teilweise unterstützen einige Funkgeräte nicht alle Funktionen des Generischen Funkgerätes, manchmal

ist es auch anders herum.

Bei den Einstellungen innerhalb von DAC ist zu beachten, dass die vier Taster (Power, Preset, Squelch, Load) und der Drehschalter keinen Off Wert senden.

Die vier Drehencoder müssen auf einen Wertebereich von 0.0 bis 2.0 eingestellt werden, die Schrittweite muss 0.1 betragen.

## Hilfsfunktionen

In den einzelnen Modul Exportdateien befinden sich diverse (teils auskommentierte) Beispiele, mit speziellen Funktionen die so nicht von LUA bereitgestellt werden. Diese Funktionen sind innerhalb der „Tools.lua“ Datei definiert und beschrieben.

Eine der wichtigsten Funktionen zum Überprüfen von Daten ist die Funktion „ExportScript.Tools.dump()“. Die Funktion „ExportScript.Tools.dump()“ ist ähnlich der Funktion var\_dump() aus PHP und gibt Informationen zu dem übergeben Wert zurück. „ExportScript.Tools.dump()“ kann mit allen LUA Typen umgehen und gibt detaillierte Informationen zu den einzelnen Werten zurück.

Dies kann wie folgt aussehen, z.B. Ausgabe der Variable „ExportScript.Config.DAC“ ( ExportScript.Tools.dump(ExportScript.Config.DAC) ).

```
{
  [1] = {
    [Host] = string: "127.0.0.1"
    [Separator] = string: ":"
    [SendPort] = number: "26026"
  }
  [2] = {
    [Host] = string: "127.0.0.1"
    [Separator] = string: ";"
    [SendPort] = number: "9092"
  }
}
```

Man kann hier erkennen wie der Inhalt der Variable „ExportScript.Config.DAC“ aufgebaut ist. Bei den einzelnen Werten wird auch immer der Typ der Daten und die Daten selber angegeben.

Die ist wichtig, falls unklar ist wie die Daten weiter zu verarbeiten sind.

Die Funktion „ExportScript.Tools.dump()“ lässt sich am sinnvollsten mit der Funktion „ExportScript.Tools.WriteToLog()“ einsetzen. „ExportScript.Tools.WriteToLog()“ schreibt die übergeben Daten in die Export.log Datei.

Z.B. ExportScript.Tools.WriteToLog(ExportScript.Tools.dump(ExportScript.Config.DAC))

# Fehlermeldung

Gibt es einen Fehler in einer der LUA Dateien, erscheint beim Start des Simulators eine entsprechende Fehlermeldung in der „DCS.log“ Datei.

Beispiel:

```
...
00027.643 ERROR   Lua::Config: Call error LuaExportActivityNextEvent:[string
"C:\Users\...\Saved Games\DCS\Ex..."]:327: attempt to compare number with table
stack traceback:
  [C]: ?
  [string "C:\Users\...\Saved Games\DCS\Ex..."]:327: in function 'StatusLamp'
  [string "C:\Users\...\Saved Games\DCS\Ex..."]:151: in function
'ExportScript.ProcessIkarusDCSConfigLowImportance'
  [string "C:\Users\...\Saved Games\DCS\Sc..."]:251: in function <[string
"C:\Users\...\Saved Games\DCS\Sc..."]:192>.
...
```

der Fehlermeldung wird der (fast vollständige) Pfad zu der betreffenden Datei angegeben. Dann folgt die Zeilennummer in der der Fehler aufgetreten ist. Danach gibt es eine mehr oder weniger genaue Fehlerbeschreibung.

Die Zeilen darunter geben an, welche Funktionen aufgerufen wurden bis der Fehler auftrat.

Anhand dieser Informationen kann der Fehler lokalisiert und behoben werden.

Falls in der Log-Datei keine Fehlermeldung erscheint, aber trotzdem kein Datenexport erfolgt, kann das aktivieren des Debug Modus in der „Config.lua“ mehr Informationen liefern.

## Informationen

Weiterführende Informationen zum Thema LUA gibt es auf der LUA Homepage unter <http://www.lua.org/manual/5.1/>

Informationen zu den Devices oder den IDs der Werte finden sich in den Dateien „device.lua“ und „mainpanel\_init.lua“ im Pfad „...\\Eagle Dynamics\\DCS World\\Mods\\aircrafts<MODUL>\\Cockpit\\Scripts\\“.

Informationen zu den DCS Flaming Cliffs Export Funktionen gibt es unter [http://lockon.co.uk/en/dev\\_journal/lua-export/](http://lockon.co.uk/en/dev_journal/lua-export/)

# Erstellen einer Exportdatei für ein DCS Modul am Beispiel der A-10C

Öffnen des folgenden Ordners: "...Eagle Dynamics\DCS World\Mods\aircraft\A-10C\Cockpit\Scripts"

Dort sind die drei folgenden Dateien von Interesse:

- devices.lua
- mainpanel\_init.lua
- clickabledata.lua

Die Dateien enthalten folgende Inhalte:

## mainpanel\_init.lua

Enthält alle Anzeigen und Lampen, in der Regel sind die auch alle verständlich bezeichnet.

z.B.

```
-- Gauges
-- Standby Attitude Indicator
SAI_Pitch = CreateGauge()
SAI_Pitch.arg_number = 63
SAI_Pitch.input = {-math.pi / 2.0, math.pi / 2.0}
SAI_Pitch.output = {-1.0, 1.0}
SAI_Pitch.controller = controllers.SAI_Pitch
```

Hier sind folgende Informationen von Interesse:

SAI\_Pitch: Als Bezeichnung des Gerätes und des damit angezeigten Wertes zu gebrauchen.

SAI\_Pitch.arg\_number = 63: Die 63 ist die ID unter der die Daten später abgefragt werden.

SAI\_Pitch.output = {-1.0, 1.0}: -1.0, 1.0 der Wertebereich der zurückgegeben Daten.

Gegebenenfalls sind einige Definitionen in extra Dateien ausgelagert und werden dann über einen dofile() Aufruf importiert.

## devices.lua

Enthält die Device Bezeichnungen und die dazugehörigen IDs (es kann vorkommen das die Ids im Kommentar nicht stimmen, dann einfach von oben durch zählen).

z.B.

```
devices["ELEC_INTERFACE"] = counter()--1
...
```

ELEC\_INTERFACE: Elektrisches Interface, hat die ID 1



## clickabledata.lua

Enthält alle Informationen zu den Tastern, Schaltern, Drehschalter und Drehregler. Meistens befindet sich am Anfang der Datei ein Block mit Funktionsbeschreibungen, hier werden die einzelnen Schalter Ausführungen beschrieben. Einträge mit der Bezeichnung "Cover" sind uninteressant, das sind nur die Schalter-Abdeckungen.

z.B.

```
-- Left MFCDI
elements["PNT-BTN-MFD-L-01"] = {class = {class_type.BTN}, hint = _("OSB 1"), device =
devices.MFCD_LEFT, action = {device_commands.Button_1}, stop_action =
{device_commands.Button_1}, arg = {300}, arg_value = {1.0}, arg_lim = {{0.0, 1.0}},
use_release_message = {true} }
```

Hier sind folgende Informationen von Interesse: class\_type.BTN: bedeutet dass es sich um einen Button handelt, diese Information ist nur für das Verständnis interessant.

\_("OSB 1"): Das ist die Englische Bezeichnung die angezeigt wird, wenn man mit der Maus im Cockpit über den Button geht. Diese Bezeichnung sollte als Teil der Beschreibung genutzt werden.

devices.MFCD\_LEFT: Das Device zum dem der Button gehört, ist das Linke MFCD (MFCD\_LEFT). Den Device Namen als Teil der Beschreibung nehmen. Über den Device Namen bekommt man aus der Datei „device.lua“ die Device ID.

arg = {300}: die 300 ist die ID unter der man den aktuellen Wert des Button auslesen kann. Dies ist in einigen Situationen von Interesse.

action = {device\_commands.Button\_1}: Die 1 aus Button\_1 ist von Interesse. Gelegentlich hat der Schalter bei stop\_action eine andere Button Nummer, dann muss sich diese auch noch gemerkt werden.

arg\_value = {1.0}: Das ist der Wert der gesendet wird, wenn der Button gedrückt wird.

arg\_lim = {{0.0, 1.0}}: Das ist der mögliche Wertebereich, also 0.0 wenn der Button nicht gedrückt ist und 1.0 wenn der Button gedrückt ist.

oder

```
elements["PNT-MFCD-L-ADJ-UP"] = {class = {class_type.BTN}, hint = _("Moving Map Scale
Adjust Increase"), device = devices.MFCD_LEFT, action = {MFCD_ADJ_Increase},
stop_action = {MFCD_ADJ_Stop}, arg = {320}, arg_value = {1.0}, arg_lim = {{0.0,
1.0}}, use_release_message = {true} }
```

Dies ist ein Wipptaster am Linken MFCD, im groben ist es ähnlich wie oben, außer folgende Dinge:

action = {MFCD\_ADJ\_Increase}: MFCD\_ADJ\_Increase ist eine Variable, über die Suche in der Datei findet man die passenden Button Nummer dazu, in diesem Fall 21.

stop\_action = {MFCD\_ADJ\_Stop}: MFCD\_ADJ\_Stop ist auch eine Variable, diesmal enthält sie die Nummer 23

oder

```
elements["PNT-LVR-MFD-L"] = {class = {class_type.TUMB, class_type.TUMB}, hint =
_("DAY/NIGHT/OFF"), device = devices.MFCD_LEFT, action = {device_commands.Button_36,
device_commands.Button_36}, arg = {325, 325}, arg_value = {0.1, -0.1}, arg_lim =
{{0.0, 0.2}, {0.0, 0.2}}}
```

class\_type.TUMB: TUMB steht hier für rotieren, also ein Drehschalter.

\_(“DAY/NIGHT/OFF”): Hier sieht man drei Bezeichnungen. Wenn man den Schalter kennt, dann weiß man das der auch nur drei Schaltstellungen hat.

arg = {325, 325}: Zweimal die ID des Drehschalters, da er ja nach links und recht gedreht werden kann.

arg\_value = {0.1, -0.1}: Das sind die Werte die beim betätigen gesendet werden. Nach links drehen 0.1 und nach rechts drehen -0.1

arg\_lim = {{0.0, 0.2}, {0.0, 0.2}}: 0.0 bis 0.2 ist der mögliche Wertebereich des Schalters. 0.0 ist der Startwert, wie schon bei arg\_value gesehen, wird entweder 0.1 aufgerechnet oder abgezogen. Also sind die möglichen Werte 0.0, 0.1 und 0.2, das entspricht auch den drei beschriebenen Schaltstellungen.

oder

```
-- CMSP
elements["PNT-LEV-CMSP-BRT"] = default_axis_limited(_("Adjust Display Brightness"),
devices.CMSP, device_commands.Button_9, 359, 0.1, false, false, {0.15, 0.85})
```

default\_axis\_limited: Ist der Funktionsname der den Schalter, oder in diesem Fall der Drehregler, beschreibt.

Hier muss man sich die Funktion anschauen um zu sehen welcher Wert welchen Zweck hat.

```
function default_axis_limited(hint_, device_, command_, arg_, gain_, updatable_, relative_,
_arg_lim)

    local relative = false
    if relative_ ~= nil then
        relative = relative_
    end

    local gain = gain_ or 0.1
    return {
        class      = {class_type.LEV},
        hint       = hint_,
        device     = device_,
        action     = {command_},
        arg        = {arg_},
        arg_value  = {1},
        arg_lim    = {_arg_lim},
        updatable  = {updatable_},
        use_OBB    = false,
        gain       = {gain},
        relative   = {relative},
    }
end
```

\_(“Adjust Display Brightness”): ist die Beschreibung des Drehreglers.

devices.CMSP: Das Device zu dem der Drehregler gehört, hier das CMSP.

device\_commands.Button\_9: Wieder die Button Nummer.

359: Ist der Wert von arg, also die ID unter der man den aktuellen Wert des Drehreglers abfragen kann.

0.1: Ist der Wert von gain, dies bedeutet bei einem Drehregler dass der Wert je nach Drehrichtung um 0.1 erhöht oder verringert wird.

false, false: Diese Werte sind uninteressant.

{0.15, 0.85}: Sind die Werte von `arg_lim`, also der mögliche Wertebereich des Drehreglers. In diesem Fall von 0.15 bis 0.85, das sind bei einer Genauigkeit von 0.1 70 mögliche Werte.

## Welche Daten kommen nun wohin?

Alle IDs (`arg_number`) aus der `mainpanel_init.lua` Datei kommen in die "ExportScript.ConfigEveryFrameArguments" Variable (vom Typ Table) in der Export Datei (basierend auf `Empty-DCS.lua`).

Das dazugehörige Format ergibt sich aus dem angegebenen Wertebereich, ist aber meistens eine Fließkommazahl.

Als Beschreibung (in Form eines Kommentars) wird die Bezeichnung der Anzeige angegeben.

```
[216] = "%.1f",      -- APU_FIRE
```

Die Selben IDs kommen auch in die dazugehörige D.A.C. XML Datei (basierend auf `Empty-DCS.xml`) in den "DCS\_ID" Bereich.

Den leeren Datenblock als Vorlage vorher ein paar Mal kopieren.

```
<DCS_ID>
  <ExportID>216</ExportID>
  <Description>APU_FIRE</Description>
  <ExportIDWithDescription>216 - APU_FIRE</ExportIDWithDescription>
  <Type>Lamp</Type>
</DCS_ID>
```

Type ist hier „Lamp“ angegeben, da es sich bei „APU FIRE“ um eine Statuslampe handelt. Durch die Angabe des Types erscheint der Parameter innerhalb von D.A.C. auf dem entsprechenden Tab für Lampen (LEDs).

Als weiterer Type ist „Display“ vorgesehen. Dieser Type ist nur für 7-Segment-Anzeigen vorgesehen (z.B. Funk Frequenzen), damit die Einträge auf dem Display Tab erscheinen.

Alle IDs der Analogen Instrumente (Zeiger,...) brauchen nicht in der D.A.C. Datei gespeichert werden, da D.A.C. die Daten nicht darstellen kann.

Alle Schalter/Regler IDs aus der "clickabledata.lua" Datei kommen in die "ExportScript.ConfigArguments" Variable (vom Typ Table) in der Export Datei (basierend auf `Empty-DCS.lua`).

Das dazugehörige Format ergibt sich aus dem angegebenen Wertebereich, ist aber meistens eine Fließkommazahl.

Als Beschreibung wird die Bezeichnung der Anzeige angegeben.

```
[101] = "%.1f",      -- PTR-EXT-STORES-JETT (mergency Jettison External Stores)
```

Die Selben Daten kommen auch in die dazugehörige DAC XML Datei (basierend auf `Empty-DCS.xml`) in den "Clickabledata" Bereich.

Den leeren Datenblock vorher ein paar Mal kopieren.

Hier wird es aber ein wenig komplizierter.

In `<ID></ID>` kommt eine fortlaufende Zahl rein, beginnend bei 1.

In <DeviceID></DeviceID> kommt die zum Gerät passende Device ID aus der “devices.lua” Datei, z.B. 12.

In <ButtonID></ButtonID> kommt die zugehörige Button Nummer, z.B. 1.

In <Discription></Discription> kommt eine passende Beschreibung was der Schalter macht.

In <DcsID></DcsID> kommt die DCS Argument Nummer.

```
<Clickabledata>
  <ID>1</ID>
  <DeviceID>12</DeviceID>
  <ButtonID>1</ButtonID>
  <Discription>Emergency Jettison External Stores</Discription>
  <Type>Switch</Type>
  <DcsID>101</DcsID>
</Clickabledata>
```

Als Type ist hier „Switch“ angegeben, da es sich bei „Emergency Jettison External Stores“ um einen Schalter handelt.

Durch die Angabe des Types erscheint der Parameter innerhalb von DAC auf dem entsprechenden Tab für Schalter (Switch).

Als weiterer Type ist „Rotary“ vorgesehen. Dieser Type ist nur für Achsen vorgesehen (z.B. default\_axis\_limited), damit die Einträge auf dem Encoder oder ADC Tab erscheinen.

Alle Device IDs aus der “devices.lua” Datei kommen in den “Devices” Block, am besten in der Reihenfolge wie sie auch in der Device.lua Datei stehen.

```
<Devices>
  <DeviceID>12</DeviceID>
  <Discription>IFFCC</Discription>
</Devices>
```

**HINWEIS:** Alle Daten können auch innerhalb von D.A.C. im „MasterData“ Tab in den entsprechenden Tabellen eingetragen werden.

## Spezial Funktionen der einzelnen Devices

Einige Devices in dem simulierten Flugzeug haben spezielle Funktionen mit denen bestimmte Werte ausgelesen werden können.

Um diese Funktionen zu finden muss einmalig der folgende Codebereich aktiviert werden (Kommentarzeichen entfernen am Ende der „ExportScript.ProcessDACConfigLowImportance()“ Funktion).

```
local ltmp1 = 0
for ltmp2 = 1, MAX-ID, 1 do
    ltmp1 = GetDevice(ltmp2)
    ExportScript.Tools.WriteToLog(ltmp2..'': '..ExportScript.Tools.dump(ltmp1))
    ExportScript.Tools.WriteToLog(ltmp2..' (metatable):
'..ExportScript.Tools.dump(getmetatable(ltmp1)))
end
```

MAX-ID muss durch die höchste ID aus der device.lua Datei ersetzt werden.

Dann starte man das Spiel mit dem entsprechenden Modul, am besten so dass das Flugzeug mit gestarteter Maschine auf dem Rollfeld steht. Es reicht wenn das Spiel nur ein paar Sekunden läuft, dann kann die Simulation wieder beendet werden.

Nun befindet sich in der Export.log Datei („C:\Users<USER>\Saved Games\DCS\Logs\Export.log“) eine lange Liste an möglichen Funktionen.

Hier als Beispiel ein Auszug des Logs bei der A-10C. Es handelt sich nur um ein paar Beispieldaten zum verdeutlichen des Aufbaues.

```
46: {
    [link] = userdata: 0000000083285608
}

46 (metatable): {
    [__index] = {
        [listen_event] = "function: 000000008EEE7360, C function"
        [listen_command] = "function: 000000008EEE72C0, C function"
        [performClickableAction] = "function: 000000008EEE7310, C function"
        [SetCommand] = "function: 000000008EEE7270, C function"
    }
}

47: {
    [link] = userdata: 0000000083367118
}

47 (metatable): {
    [__index] = {
        [get_sideslip] = "function: 000000008EEEA3C0, C function"
        [performClickableAction] = "function: 000000008EEE9E30, C function"
        [get_bank] = "function: 000000008EEEA320, C function"
        [listen_event] = "function: 000000008EEE9E80, C function"
        [get_pitch] = "function: 000000008EEEA070, C function"
        [listen_command] = "function: 000000008EEE9DE0, C function"
        [SetCommand] = "function: 000000008EEE9D90, C function"
    }
}
```

Als erstes steht immer die Devices ID, des abgefragten Gerätes. Der erste Daten Block gibt mögliche Daten wieder, die man direkt verarbeiten könnte. Meistens enthält der Block nur „[link] = userdata:“, dies ist ein Verweis auf Daten die man nicht direkt verarbeiten kann.

Dann kommt wieder die Devices ID gefolgt von dem Vermerk „metatable“. Dieser Block gibt Funktionen an, mit denen auf die Daten des Gerätes zugegriffen werden kann.

Bei der ID 46 gibt es keine besonderen Funktionen, da das Device 46 das NMSP (Navication Mode Select Panel) ist. Und dieses Panel hat nur ein paar Schalter.

Bei der ID 47 sieht es schon etwas anders aus. Die ID 47 ist das ADI (Attitude Direction Indicator), hier gibt es einige Werte die man sich ausgeben lassen kann. Alle Funktionen die mit „get“ anfangen sind interessant.

Das wäre hier „get\_sideslip“, „get\_bank“ und „get\_pitch“. Die Bezeichnungen geben schon an welche Daten hier zurück geliefert werden.

```
54: {
  [link] = userdata: 0000000083377A68
}

54 (metatable): {
  [__index] = {
    [listen_command] = "function: 000000005DB929A0, C function"
    [set_frequency] = "function: 000000005DBA2540, C function"
    [is_on] = "function: 000000005DB69A10, C function"
    [get_frequency] = "function: 000000005DB830F0, C function"
    [performClickableAction] = "function: 000000005DB90500, C function"
    [set_modulation] = "function: 000000005DB86B90, C function"
    [set_channel] = "function: 000000005DB90640, C function"
    [listen_event] = "function: 000000005DB910A0, C function"
    [SetCommand] = "function: 000000005DB72CC0, C function"
  }
}
```

Die ID 54 ist bei der A-10C das UHF Radio, hier sind die beiden Funktion „get\_frequency“ und „is\_on“ interessant.

Auf dieser Art lassen sich viele spezielle Funktionen ermitteln mit denen viele Daten über das Export Script zurückgegeben werden können.

Wie man diese Funktionen verwendet muss im Einzelfall ausprobiert werden, hilfreich ist es sich das Vorgehen dabei in den bereits existierten Export Scripten ab zuschauen.

Am besten verwendet man die Spezial Funktionen in der Funktion „ExportScript.ProcessDACConfigLowImportance()“ des Export Scriptes. Dann werden die Funktionen nicht so häufig aufgerufen und erzeugen dadurch auch nicht eine höhere Systemlast.