DCS Export Script

Version 0.9.9 BETA

Table of contents

1 Notice

This is an universally insertable export script for DCS. It allows for the simultaneous export of data of different programs and hardware

At present the following export formats are supported: * D.A.C. (DCS Arcaze Connector) (DAC in GitHub) to address the Arcaze USB Controller (http://wiki.simple-solutions.de/en/products/Arcaze/Arcaze-USB) * HawkTouch in version 1.5/1.6 (http://forums.eagle.ru/showthread.php?t=71729) with new GaugePack.dll in the version by H-J-P * HELIOS in the version 1.3.19 (http://www.gadrocsworkshop.com/helios/) (Only supports the DCS Module A-10C, Ka-50 and in a limited capacity the SU-25T and the Flaming Cliffs Aircraft)

and other Software/Hardware that obtain your data via a UDP network connection and then process it into a Key=Value Format.

The export script package is divide into two rudimentary categories. Once in the script „Export.lua“

under the path „%USERPROFILE%\Saved Games\DCS\Scripts\“

```
C:\Users\<USER>\Saved Games\DCS\Scripts\Export.lua
```

And then also the folder „ExportsModules“ with the respective settings for the DCS Module, under the path „%USERPROFILE%\Saved Games\DCS\ExportsModules\“

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

The basic settings are featured in the file „Export.lua“. In the folder „ExportsModules“ the individual files for the DCS Module, e.g. „A-10C.lua“ are found. Featured here are the settings for every individual module.

The exporting modules can quite easily be expanded. In order to do this all the information concerning the export data needs to be written into a new file. This file then needs to be saved under the corresponding module in the folder „ExportsModules“. How the files are to be configured stands below, or has already been documented in the existing files.

This export script supports the single and multiplayer mode, as well as changing between aircraft while the simulaton is running (RAlt + J).

---

2 Installation

> **IMPORTANT**: Firstly please create a backup file of all existing files!

Temporarily unzipping the ZIP archive onto the desktop.

Copy both the obtained folders „Scripts“ and „ExportsModules“ in the file path „%USERPROFILE%\Saved Games\DCS\“ (e.g. „C:\Users\\Saved Games\DCS\“).

---

3 Configuration

> **IMPORTANT**: For editing the LUA files please use a reasonable editor, e.g Notepad++ (http://notepad-plus-plus.org/)

The basic settings for the export are found in the upper section of the installed „Export.lua“ file.

Default settings:

```
-- for GlassCockpit Software`
gES_GlassCockpitExport = true -- false for not use`
gES_GlassCockpitHost = "127.0.0.1" -- IP for HELIOS/HawgTouch`
gES_GlassCockpitPort = 9089        -- Port for HELIOS/HawgTouch `
gES_GlassCockpitSeparator = ":"`
gES_GlassCockpitType = 2           -- 1 HELIOS, 2 HawgTouch`

-- for example D.A.C. or SIOC`
gES_HARDWAREExport = true -- false for not use`
gES_HARDWARE = {}`
-- first hardware`
gES_HARDWARE[1] = {}`
gES_HARDWARE[1].Host = "127.0.0.1" -- IP for hardware 1`
gES_HARDWARE[1].SendPort = 26026 -- Port for hardware 1`
gHARDWARE[1].Separator = ":"`

-- second to n hardware`
--gES_HARDWARE[2] = {}`
--gES_HARDWARE[2].Host = "127.0.0.1" -- IP for hardware 2`
--gES_HARDWARE[2].SendPort = 9092 – Port for hardware 2`
--gES_HARDWARE[2].Separator = ";"`

-- D.A.C. can data send `
gES_HARDWAREListner = true          -- false for not use `
gES_HARDWAREListnerPort = 26027          -- Listener Port for D.A.C. `

gES_ExportInterval = 0.1 `
gES_ExportLowTickInterval = 1 `
gES_ExportModulePath = lfs.writedir().."ExportsModules\\" `
gES_LogPath = lfs.writedir().."Logs\\Export.log" `
gES_Debug = true`
```

The configuration block is subdivided into 4 sections.

The first section is for ‚glas-cockpit-solutions‘ as HELIOS or HawgTouch.

The variable „gES_GlassCockpitExport“ indicates whether data can be exported to a glasscockpit software. Here „true“ or „false“ can be specified.

The variable „gES_GlassCockpitHost“ indicates which IP address the data should be sent to. Here the IP address „127.0.0.1“ (for localhost) or another valid IP adress can be given to destination computer.

The variable „gES_GlassCockpitPort“ indicates to which port the data should be sent to. The specified port (1625) is the HawgTouch default port and should not be changed.

The variable „gES_GlassCockpitType“ is used to define which glascockpit Software is relevant. 1 for HELIOS or 2 for HawgTouch

The second part is for the hardware where the data should be exported to. It is possible for multiple IP addresses and Ports to be specified under different hardwares.

The variable „gES_HARDWAREExport“ indicates if the data should be exported to the (alle) hardware control software. Here „true“ or „false“ can be specified.

The Lines

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

contain the data for the first hardware export.

The line gES_HARDWARE[1] = {} may not be changed.

The variable „gES_HARDWARE[1].Host“ indicates to which IP address the data should be sent to. Here the IP address „127.0.0.1“ (for localhost) can be specified or that of an alternative valid IP address of the destination computer.

The variable „gES_HARDWARE[1].SendPort“ indicates to which Port the data should be sent to. The Port „26026“ is the default-Port of D.A.C. and does not need to changed. If the Port does however get changed, this must be done within the D.A.C. configuration.

The variable „gES_HARDWARE[1].Separator“ indicates the delimiter between the single Key=Value pairs. The default-delimiter for D.A.C. is the colon „:“, otherwise all symbols other than the equal-sign „=“ are allowed.

For a further hardware export the following 4 lines have once again been insert to amend the respective data.

> **IMPORTANT**: Important is here that the Number/Figure in the square brackets needs to be replaced by the next higher one.

For example:

```
gES_HARDWARE[2] = {}
gES_HARDWARE[2].Host = "192.168.0.14" -- IP for hardware 2
gES_HARDWARE[2].SendPort = 9090 -- Port for hardware 2
gES_HARDWARE[2].Separator = ";"
```

In this way many different hardware/software combinations can be addressed virtually unrestricted.

The third section concerns the data reception from D.A.C.

The variable „gES_HARDWAREListner“ indicates if the data should be received and processed. Possible values here are „true“ and „false“.

The variable „gES_HARDWAREListnerPort“ indicates on which Port, D.A.C is listening for Data. By default „26027“ is used for Datatransfere betwen D.A.C. and DCS.

The fourth section specifies the general values and rarely needs to be changed.

The default values are as follows:

```
gES_ExportInterval = 0.1
gES_ExportLowTickInterval = 1
gES_ExportModulePath = lfs.writedir().."ExportsModules\\"
gES_LogPath = lfs.writedir().."Logs\\Export.log"
gES_Debug = false
gES_FirstNewDataSend = true
gES_FirstNewDataSendCount = 5
gES_genericRadioHardwareID = 1
```

And the individual values mean the following:

„gES_ExportInterval“: indicates the interval in which the time critical data needs to be updated (Default 0.1)

„gES_ExportLowTickInterval“: indicates the interval in which the other data needs to be updated. (Default 1). The „gES_ExportLowTickInterval“ is reached when the „gES_ExportInterval“ has been so often downloaded, that the values for „gES_ExportInterval“ and „gES_ExportLowTickInterval“ are the same.

„gES_ExportModulePath“: provides the path to the module definitions.

„gES_LogPath“: provides the path for log-files.

„gES_Debug“: switches logging on or off. With the Funktion „WriteToLog()“ you can send data to the log-file.

„gES_FirstNewDataSend“: specifies if all Export Data should be resent immediately after starting the simulation, in case the initail data was not present.

„gES_FirstNewDataSendCount“: specifies the delay of dataoutput. The value represents the quantity of loops in the export-script.

„gES_genericRadioHardwareID“: specifies the HardwareID for the generic radio in D.A.C. The value has to be the same as that of the index gES_HARDWARE[X].

> **IMPORTANT**: All informations for the configuration have to be specified in the defined format.

The export script can also receive data from HELIOS or D.A.C. and then pass it onto the DCS. This feature is unfortunately only support for the complete DCS module.

Example of the exported data using the A-10C module:

HELIOS:

```
53d7fc73*33=0.0020:35=0.0020:47=0.0009:63=0.0091:70=0.3267:78=0.7042:82=0.6889:281=
0.0599:4=0.4315:83=0.6902:12=-0.0585:14=0.0040:18=0.0164:20=-0.0492:23=-0.8279:24=-
0.0746: 34=0.0457:36=0.0432
:48=0.6310:64=-0.0102:76=0.2628:80=0.7058:84=0.2269:88=0.8235:
73=0.3262:77=0.2635:85=0.2290:89=0.8235:648=0.6138:647=0.9025:2051=0.0000;0.3000;0.
48475:2029=0.00;0.00;0.0102:2090=0.02;0.92;0.91821

53d7fc73*33=0.0025:35=0.0025:47=0.0019:63=0.0101:70=0.3573:78=0.6417:82=0.6345:281=
0.0616:4=0.4446:83=0.6361:12=-0.0573:14=0.0047:18=0.0173:23=-0.7070:24=-0.0247:34=0.
0467: 36=0.0447:48=0.6219:
64=-0.0114:76=0.2243:80=0.6436:84=0.2016:88=0.8245:73=0.3569:77=0.2252:85=0.2041:
89=0.8245:648=0.5650:647=0.7801:2051=0.0000;0.3000;0.43930:2029=0.00;0.00;0.0112:
2090=0.03;0.93;0.93496
```

D.A.C.:

```
53d7fc73*178=1:179=1:181=1:182=1:191=1:480=1:481=1:482=1:483=1:484=1:485=1:486=1:48
7=1:488=1:489=1:490=1:491=1:492=1:493=1:494=1:495=1:496=1:497=1:498=1:499=1:500=1:5
01=1:502=1:503=1:504=1:505=1:506=1:507=1:508=1:509=1:510=1:511=1:512=1:513=1:514=1:
515=1:516=1:517=1:518=1:519=1:520=1:521=1:522=1:523=1:524=1:525=1:526=1:527=1:540=1:
541=1:542=1:606=1:610=1:614=1:616=1:618=1:619=1:620=1:659=1
:660=1:661=1:663=1:664=1:665=1:730=1:731=1:732=1:737=1

53d7fc73*404=1

53d7fc73*404=0

53d7fc73*404=1:2005=10600

53d7fc73*404=0
```

---

4 ExportsModules

The folder „ExportsModules“ contains the individual module specification files, e.g. A-10C.lua or Su-25T.lua .

In these files are all the exportable data of the respective modules are performed.

Here a few exemplary examples for a DCS and a Flaming Cliffs modul.

A-10C.lua

The A-10C is a fully simulated DCS module. This module contains very precise individual simulated systems and a clickable cockpit.

Basically the files for the DCS module have been divided into numerous blocks.

```
gES_GlassCockpitConfigEveryFrameArguments
gES_GlassCockpitConfigArguments
ProcessGlassCockpitDCSConfigHighImportance()
ProcessGlassCockpitDCSConfigLowImportance()
ProcessHARDWAREConfigHighImportance()
ProcessHARDWAREConfigLowImportance()
genericRadio()
```

The first two blocks are variables and contain the information which data is to be sent HELIOS.

The contents of the variable is structed as follows:

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

DCS ID: 4 Output format: "%.4f" means a floating point number with 4 decimal places. As commentary: AOA (A description what the value for data provides)

With ID 4, EagleDynamics provides the AOA value of the A-10C. For the glasscockpit software we use a floating-point number with four digits. The value arriving in the glascockpit software could look like: „4=0.5486“.

The variable „gES_GlassCockpitConfigEveryFrameArguments“ includes all the Informations for time-critical Values, e.g. all the Values for instruments and statuslights.

The variable „gES_GlassCockpitConfigEveryFrameArguments“ includes all other Values. E.g. the status of switches.

The two funktions „ProcessGlassCockpitDCSConfigHighImportance“ and „ProcessGlassCockpitDCSConfigLowImportance“ could contain values you have to calculate or find out with special functions. Also included in this funcions are several blocks for the glas-cockpit-solutions. E.g. frequencies for all the radio devices.

```
local lUHFRadio = GetDevice(54)
SendData(2000, string.format("%7.3f", lUHFRadio:get_frequency()/1000000))
```

The top Line, prepares a variable for the informations from ID 54, this represents the UHF Radio in the case of the A-10C. You can find the ID in the „devices.lua“ file, in „..\Eagle Dynamics\DCS World\Mods\aircraft\A-10C\Cockpit\Scrips\“

In the second Line the data will be read, calculated, formatted and sent per SendData to HELIOS. The function „SendData():“ will do this job. „2000“ is the key HELIOS can receive data. „string.format()“ is a LUA-function to format the data (look to http://www.lua.org/manual/5.1/manual.html#pdf-string.format for further information) „%7.3f“ cause the formatting as a floating-point number with 7-digit ahead and 3 digits after the comma. The function „lUHFRadio:get_frequency():“ provides the actual frequency from the UHF Radio as a decimal value in Hz. „/1000000“ the value will divided with 1.000.000 to get the frequency in MHz.

The function „ProcessGlassCockpitDCSConfigHighImportance“ is executed more often as the function „ProcessGlassCockpitDCSConfigLowImportance“. So the first function is usefull for time-critical Values.

The both functions „ProcessHARDWAREConfigHighImportance“ and „ProcessHARDWAREConfigLowImportance“ entsprechen den beiden oben genannten Funktionen und sind für die Ausgabe an die Hardware zuständig.

These functions responsible for reading datas from the simulation. Upon receipt these datas may or may not be modified and thereafter will be sent to the hardware.

All entries for input or output will be sent to the first hardware entry in the „Export.lua“. Normally thats the Arcaze USB Controller receiving data from D.A.C. For this reason only Data will be sent, which can be processed from the Arcaze USB Controller. E.g. LEDs or 7-segment displays.

An Example for the data inquiry and transfer:

```
SendDataHW("404",  mainPanelDevice:get_argument_value(404)) -- MASTER_WARNING_LAMP
```

SendDataHW():is the function that sends the Data to the Hardware. „404“: is the Key D.A.C. is listening for this information. This key is defined in the configfile for D.A.C.

mainPanelDevice:get_argument_value(): mainPanelDevice contains all Data for the aircraft. The Data can only be prompted via the special function get_argument_value().

404: is the ID used to tell the function get_argument_value()which Data should prompted. The ID can be carried over from „gES_GlassCockpitConfigEveryFrameArguments“ and „gES_GlassCockpitConfigArguments“. More IDs can be found in the „mainpanel_init.lua“ located in „mainpanel_init.lua“ im Pfad „...\Eagle Dynamics\DCS World\Mods\aircraft\A-10C\Cockpit\Scripts\“.

The function „SendDataHW()“ has also a third optional parameter => the ID to identify the Hardware used for processing.

```
SendDataHW("404",  mainPanelDevice:get_argument_value(404), 2)
```

In this example the data will be sent (as explained above), to the second, in the „Export-lua“ file listed Hardware.

Additional functions „ProcessHARDWAREConfigHighImportance“ and „ProcessHARDWAREConfigLowImportance“ will identify, process and send all Data that are not readout by the function „mainPanelDevice:get_argument_value()“. To explain this in detail would over shoot the purpose of these instructions. For more information you can consult the comments in the programmcode.

The function „genericRadio()“ will be explained in detail later in this Readme.

All files for the DCS Modules were built similar and are annotated. Furthermore all the Data relevant for D.A.C. has already been listed. Usually it’s not necessary to edit these files.

SU-25T.lua

All non-DCS-modules are Flaming Cliffs-Modules and are to be handle in a different way. We will only describe the module SU-25T.

In the FC-modules a lot less Data can be exported.

As a result all the Files for FC Modules are constructed in a different way.

The SU-25T files are fielded in four big blocks and various help functions.

```
ProcessGlassCockpitFCHighImportanceConfig()
ProcessGlassCockpitFCLowImportanceConfig()
ProcessHARDWAREConfigHighImportance()
ProcessHARDWAREConfigLowImportance()
```

The first two functions present all the data sent to Helios.

Unfortunately HELIOS can only handle very few Informations from the FC-Modules. Especially the data for the analog gauges. Unlike HawgTouch which supports nearly all Data avaliable in Flaming Cliffs.

The value determined by DCS-functions eventually has to be converted and afterwards sent to the glas-cockpit-software. Convertion is necessary to get the Data with the correct unit. DCS exports metrical Data and Helios or HawgTouch needs imperial values.

Examples can be found in the included configfiles.

Both functions „ProcessHARDWAREConfigHighImportance“ and „ProcessHARDWAREConfigLowImportance“ including all information for data sent to the Hardware.

These Datas will be exportet from DCS in groups per function or device. For that reason special functions were made for some groups of data in different Modules.

For example the function SPO15RWR() includes all Data for the SPO-15 Radar Warning Receiver. With all this Information it‘s now possible to build your own SPO-15.

All this functions have an optional parameter which tells them to which hardware the data should be sent.

For example with the entrie „SPO15RWR(2)“ this data will be sent to the second hardware defined in the „Export.lua“.

As previously explained with the DCS Modules the second hardware are Arcaze Modules.

All Files for the Flaming Cliffs Modules were also built similar and are annotated.

Sadly the US-Modules use many displays, so we loose some infos we would need to manipulate LEDs and Needls.

---

5 Generic Radio for DCS-Modules

For some DCS-Modules we made a special function in the export script for a „generc Radio“.

This function can be usefull to handle different radios with the same hardware.

This function can only be used with D.A.C., because of some special functions in D.A.C. and the Arcaze Hardware.

You need the following Hardware to use the GenericRadio-Functions:

```
1 x „Arcaze USB-Controller“
1 x „Display Driver 3“ or „Display Driver 32“
1 x „8fold display board“ plus 8 displays (seven-segment displays)
or 1 x „2fold display board“ and 1 x „6fold display board“ plus 8 displays
1 x rotary switch (3 positions needed)
3 x pushbuttons
4 x rotary encoder
optional
3 x low current LED
```

The rotary switch is used to select the radio. All displayed data and inputs are now set to the chosen radio.

Button one is required to switch the radio on or off (electricity supply).

The first 2 digits of the display-driver-board now shows the Preset Channel (if existing).

The second Button is used to switch the radio from manual-mode to the preset-channels and vs (if the radio uses presets). If the preset-mode is active the remaining 6 digits will show the corresponding frequenzy. (unlike the simulation)

The third button is to initiate the squelch function.

Button four is to load or save the preset-channel (according to the type of radio).

The first encoder is to change the preset-channel.

Encoders three and four are used to chose the frequenzy (1x before the comma, 1x after the comma).

The Last Encoder is necessary to change the volume.

The optional LEDs are usefull to show the power-status, the squelch-status and the mode of the selected radio.

Not all available Modules can use all the functions of the generic radio, and vice versa.

By the configuration of the Buttons in D.A.C. (power, preset, squelch) you have to use the „S“ option to prevent sending the ‚off-values‘. The values for all the encoders must be set to: min: 0.0, max:2.0 and dent: 0.1

---

6 Help functions

In the different export-files (e.g. A-10C.lua) are many examples included for some special functions. Some of this functions are commented out. All these functions are defined and explained in the „Export.lua“ file.

One of the most important functions for testing is the „dump()“ – function. This function is similar to the PHP-function „var_dump()“ and is used to get detailed information from the different values. „dump()“ can be used with all LUA-typs.

An example of „gES_HARDWARE“ (dump(gES_HARDWARE))

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

You can see the structure of the value from the „gES_HARDWARE“ variable. The function always gives the type and the value back.

Sometimes this information is important for further processing of the data.

Most usefull is the „dump()“ function in combination with the „WriteToLog()“ function. So you can get a Logfile with all the data you need.

e.g. WriteToLog(dump(gES_HARDWARE))

---

7 Errormessages

If there is a Error in one of the LUAs, during start of the simulation an erromessage will be written to the „DCS.log“ file.

```
...
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
'ProcessHARDWARELowImportance'
    [string "C:\Users\...\Saved Games\DCS\Sc..."]:251: in function <[string
"C:\Users\...\Saved Games\DCS\Sc..."]:192>.
...
```

In the above example you can see:

- the path to the LUA-file.
- The Number of the Line where the error was found.
- The errormessage (expressive or not)

Die line numbers below tells you, which functions were accomplished before the error occurred.

So you can find the error and can try to fix it.

---

8 Additional infos

More Infos to LUA can found on the LUA-Homepage http://www.lua.org/manual/5.1/

Informations to the IDs and devices can be found in the „device.lua“ or „mainpanel_init.lua“ in the folder: „...\Eagle Dynamics\DCS World\Mods\aircraft\\Cockpit\Scripts\“.

You can also find informations to DCS Flaming Cliffs Export functions on: http://lockon.co.uk/en/dev_journal/lua-export/

---

9 Create your own export-file for DCS (A-10C as example)

Open the following folder: "...\Eagle Dynamics\DCS World\Mods\aircraft\A-10C\Cockpit\Scripts\"

The listed files are important:

*devices.lua *mainpanel_init.lua *clickabledata.lua

The Files contain the following:

mainpanel_init.lua

All displays and lamps are normally adequately commented.

e.g.

```
-- 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
```

Following informations are important: SAI_Pitch: Needed as Label of the device and the displayed value. SAI_Pitch.arg_number = 63: defines the ID used to pick out the data later. SAI_Pitch.output = {-1.0, 1.0}: -1.0, 1.0 is defining the range of the values.

devices.lua

Conatins the labels of the devices and the appropriate IDs. If the IDs in the annotation doesn’t match, you have to enumerate.

e.g.

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

ELEC_INTERFACE: electrical interface, has the ID 1

clickabledata.lua

Includes all informations to the buttons, switches, rotarys and encoders. Normally the file includes a discription at the top. There the all the switches are explained. Entrys called „Cover“ are only for the switchguards and are not mandatory.

e.g.

```
-- 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} }
```

Following Informations are important: class_type.BTN: just tells ‚it's a button' – thats only for understanding.

_("OSB 1"): is the ‚Mouse-Over Tooltip' in the 3D-Cockpit. We recommend to use this text as part of the description.

devices.MFCD_LEFT: is the Device name for which the button is for. This name should also be a part of the description. You can use this name to find the correct ID in the „device.lua“.

arg = {300}: 300 is the ID to read the value of the button.

action = {device_commands.Button_1}: The 1 is important. Sometimes a switch uses for the stop_action a different number. You have to make sure that you don not overlook this.

arg_value = {1.0}: If the button is pushed, this value that will bes sent.

arg_lim = {{0.0, 1.0}: Defines the possible range of the values. 0.0 => button not pushed, 1.0 button pushed.

other example:

```
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} }
```

This example is for a rocker switch on the left MFCD. The differences are: action = {MFCD_ADJ_Increase}: MFCD_ADJ_Increase is a variable. In the file we can find the button-number for this function => 21. stop_action = {MFCD_ADJ_Stop}: MFCD_ADJ_Stop is also a variable. This one has the Number 23.

other example:

```
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 means turning, so we have a rotary-switch.

_("DAY/NIGHT/OFF"): Here we can see the name of the three possible positions.

arg = {325, 325}: the rotary switch has two ID's for either turning it clockwise or counterclockwise.

arg_value = {0.1, -0.1}: defines the Values sent when the rotary-switch is used. 0.1 for turning left, -0.1 for turning right.

arg_lim = {{0.0, 0.2}, {0.0, 0.2}}: 0.0 to 0.2 are the possible values for the switch. 0.0 is the start-value. In arg_value it has already been defined to add or subtract 0.1. Therefore the possible Values are 0.0, 0.1 and 0.2. That corresponds to the possible positions already described.

other example:

```
-- 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: Is the name of the switch, or rotary function.

other example:

```
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
```

Here we have to check the function to find the purpose of the different values.

(_("Adjust Display Brightness"): is the description used for this rotary-switch. devices.CMSP: Is the device where the rotary-swich is included. device_commands.Button_9: is the button-number again. 359: Is the value of arg / the ID necessary to retrive the actual value of the rotary-switch. 0.1: is the value for the gain. So the value for the rotary-switch will be encreased or decreased by turning it. false, false,: these values are not used for our purpose. {0.15, 0.85}: are the values of arg_lim, and defines the range of values for this rotary-switch. 0.15 to 0.85 gives 70 possible values with a accuracy of 0.1.

10 Which data come now where?

All IDs (arg_number) from the mainpanel_init.lua-file has to be inserted to the "gES_GlassCockpitConfigEveryFrameArguments" variable (table type) in the export-file (copy of your Empty-DCS.lua)

The format comes about from the specified value range. Mostly this will be a floating-point number. From the description (inline comment) comes the name of the display.

```
[216] = "%0.1f",    -- APU_FIRE
```

You can write the IDs to the "ProcessHARDWAREConfigHighImportance" function also. See the examples there.

The same IDs have been inserted in the relevant D.A.C. XML-file (your copy of the Empty-DCS.xml) in the „DCS_ID“ section.

Bear in mind to copy the empty block of data a bunch of times.

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

In the example above the type „Lamp“ is specified, because the „APU FIRE“ ID represents such an indicator. This entry is nessesary to find the ID later in D.A.C. in the correct Tab. So you can use all IDs defined as „Lamp“ to control a LED. Another option is „Display“ for seven-segment displays connected to your Arcaze-Display-Driver. For example: It‘s possible to display frequencies or the quantity of ammo rounds. All IDs for switches or adjusters from the "clickabledata.lua" have to be defined in the "gES_GlassCockpitConfigArguments" variable (table type) in the export file (copy of your Empty-DCS.lua). The format came about from the specified value range. Mostly this will be a floating-point number. From the description (inline comment) comes the Name of the Display.

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

All this entries have also been added into the relevant D.A.C. XML-file (copy of your Empty-DCS.xml) to the " Clickabledata " section.

Again, bear in mind to copy the empty block of data a few times.

Now it starts to come a little more complicated For an ongoing number (beginning with 1) has to be entered. In the proper Device ID (from "devices.lua") has to be enterd (e.g. „12“) In is for the Button Number (e.g. „1“) In is used for a suitable description for this Switch.

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

The type „Switch“ is used because the „Emergency Jettison External Stores“ is a button. This entry is nessesary to find the ID later in D.A.C. in the correct Tab for switches. An other possible type is „Rotary“. This type is used for axes (e.g. default_axis_limited). All this entries can be found on the ADC tab.

All device IDs from the „devices.lua“-file have to the entered into „Devices“-Section. Advisably in the same order as they were listed in the „Device.lua“ file.

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

TIP

Optional there is the possebility to edit the table with all this data via the „MasterData“-Tab in D.A.C.

11 Speciall functions of all the different devices

Some devices in the simulator for the chosen aircraft use special functions to read values.

To find this functions, you have to activate the programmcode at the end of the „ProcessHARDWAREConfigLowImportance“ function (just remove the comment-assignment at the beginning of the line).

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

MAX-ID has to be replaced by the highest ID in the device.lua file.

Now you have to start the simulation with the correct modul. (With active engines on the runway to save time. After few seconds in the cockpit you are able end the simulation.

Now you can open the Export.log file („C:\Users\\Saved Games\DCS\Logs\Export.log“) and check all the entries with the avaliable functions.

Below you can see an extract from the log file from the A-10C. Just as an illustration of the layout.

```
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"
    }
}
```

The first entry is always the ID, for the relevant device. The data in the first block can be used directly.

But very often the first block of data includes just „[link] = userdata:“, and these are only links to data which cannot be directly used.

After this the file shows again the Device ID, followed by „metatable“. This block includes functions to access all the data of this device.

In ID 46 are no special functions included because Device 46 is the NMSP (Navication Mode Select Panel). This Device only uses a few switches.

Next Device (ID 47) is the ADI (Attitude Direction Indicator), so there are some values to deal with. All functions beginning with „get“ are relevant.

In this example „get_sideslip“, „get_bank“ and „get_pitch“. The label already tells us what data are included.

```
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"
    }
}
```

The device with ID 54 in the A-10C is the UHF Radio. The avaliable functions are „get_frequency“ and „is_on“.

With this method we have access to many data which can be written into our export script.

To use these functions correctly you will require a bit trial and error. Check the existing files first to find the right way sooner.

It is recommended to use these functions is the section „ProcessHARDWAREConfigLowImportance“ in the export sript, to get better performence.