DCS Export Script

Version 1.0.0

Table of contents

Notice

This is an universally applicable 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) to address the Arcaze USB Controller (http://wiki.simple-solutions.de/en/products/Arcaze/Arcaze-USB)
- Ikarus by H-J-P, ours new glass cockpit software

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 divided two major areas, the program logic and the module-specific logic. The scripts “Config.lua” and “ExportScript.lua” under the path “%USERPROFILE%\Saved Games\DCS\Scripts\DCS-ExportScript”

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

and the scripts “Tools.lua” and “genericRadio.lua” in the local folder “lib” are responsible for all the control and logic of the data export. The module-specific scripts are in the folder “Export Modules” in the path “%USERPROFILE%\Saved Games\DCS\Scripts\DCS-ExportScript\ExportsModules”

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

In the “Config.lua” file the basic settings are made.

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 name 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 simulation is running (RAlt + J).

Installation

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

Temporarily unzipping the ZIP archive onto the desktop.

Copy the folder contained “scripts” in the file path “%USERPROFILE%\Saved Games\DCS”.

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

If a “Export.lua” file another software already exists, the following lines need to be added at the end of this file.

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

Configuration

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

In the “Config.lua” file are the Basic settings for exporting.

Default settings:

```
-- 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 = ":"
-- secound 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 secounds
ExportScript.Config.ExportLowTickInterval  = 0.5    -- export evry 0.5 secounds
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
```

The configuration block is subdivided into 4 sections.

The first section is for the software “Ikarus”.

The variable “ExportScript.Config.IkarusExport” indicates whether data are to be exported to Ikarus. Here, “true” or “false” to indicate.

The variable “ExportScript.Config.IkarusHost” indicates to which IP address the data should be sent. Here, the IP address “127.0.0.1” will specify (localhost) or another valid IP address of the target computer.

The variable “ExportScript.Config.IkarusPort” indicates the port to which the data is to be sent. The port “1625” is the default port of Ikarus and need not be changed. If the port is changed anyway, this must be done within the Ikarus configuration.

The second area is for the software D.A.C. provided, to be exported to the data. It is possible several IP specify addresses and ports for different installations of the software.

The variable “ExportScript.Config.DACExport” indicates whether data to (all) D.A.C. Installations should be exported. Here, “true” or “false” to indicate.

The Lines

```
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 = ":"
```

contain the data for the first hardware export.

The line “ExportScript.Config.DAC[1] = {}” may not be changed.

The variable “ExportScript.Config.DAC[1].Host” indicates to which IP address the data is sent. Here, the IP address “127.0.0.1” will specify (localhost) or another valid IP address of the target computer.

The variable “ExportScript.Config.DAC[1].SendPort” indicates the port to which the data is to be sent. The port “26026” is the default port of D.A.C. and need not be changed. If the port is changed anyway, this must also be within the D.A.C. Configuration to be done.

The variable “ExportScript.Config.DAC[1].Separator” specifies the delimiter between the Key = Value pairs. For D.A.C. If the default delimiters the colon “:”, otherwise all characters except the equal sign “=” Allowed.

For another hardware export to another computer these 4 lines are again already registered exists among and supplemented by 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:

```
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 = ":"
```

In this way many different hardware/software combinations can be accessed almost indefinitely.

The third area relates to the data receiving D.A.C.

The Variable “ExportScript.Config.DACListener” indicates whether data should be received and processed. Possible values are “true” and “false”.

The variable “ExportScript.Config.DACListenerPort” specifies the port on which the export script data from D.A.C. Receives. The port “26027” is the default port of D.A.C. and need not be changed.

The fourth section sets out general values and need not be changed as a rule.

The default values are as follows:

```
ExportScript.Config.ExportInterval         = 0.05   -- export evry 0.05 secounds
ExportScript.Config.ExportLowTickInterval  = 0.5    -- export evry 0.5 secounds
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
```

And the individual values mean the following:

ExportScript.Config.ExportInterval: specific the time interval in seconds, in which the time-critical data are updated (default 0.05)

ExportScript.Config.ExportLowTickInterval: specifies the time interval in seconds, in which the other data are updated (Default 0.5).

ExportScript.Config.LogPath: specifies the path for the log file to

ExportScript.Config.ExportModulePath: specifies the path to the module definitions.

ExportScript.Config.Debug: indicates whether data should be logged, data is written to the log file through the function “ExportScript.Tools.WriteToLog()”.

ExportScript.Config.FirstNewDataSend: indicates whether all export data to be sent again once more after the start of the simulation, it is important to be present after the start no initial data.

ExportScript.Config.FirstNewDataSendCount: specifies the date from which export data is to be sent again, the value is not a time but a counter which counts how many times the export script has supplied data.

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

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 module.

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.

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

The first two blocks are variable and which contain information which data should be generally exported.

The contents of the variable is structured 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 glass cockpit software we use a floating-point number with four digits. The value arriving in the glass cockpit software could look like: “4=0.5486”.

The variable “ExportScript.ConfigEveryFrameArguments” contains the information about the time-critical values, for example, the values of the instruments and status lights.

The variable “ExportScript.ConfigArguments” contains the information on other values, such as switch positions.

The two functions “ExportScript.ProcessIkarusDCSConfigHighImportance()” and “ExportScript.ProcessIkarusDCSConfigLowImportance()” to review information about contain values are only calculated or determined with special functions. The values determined here are output to Ikarus.

For example, here the radio frequencies are determined.

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

The first line creates a local variable with the information of the device with the ID 54, this is in the A-10C, the UHF radio. The device ID are in the “devices.lua” file in the path “…\Eagle Dynamics\DCS World\mods\aircraft\A-10C\cockpit\Scripts”

In the second line the data is read, converted, formatted and with the ExportScript.Tools.SendData()

function to Ikarus transferred.

ExportScript.Tools.SendDaten(): is the function with which the data is transmitted to Ikarus.

2002 is the key under the Ikarus receiving data / expected

string.format(): is a LUA function with which data can be formatted (for more information see the official documentation LUA http://www.lua.org/manual/5.1/manual.html#pdf-string.format)

“% 7.3f”: the specified value is displayed as a floating point number with 7 points before and three places after the decimal point.

lUHFRadio:get_frequency(): it is called radio devices special function get_frequency() of UHF, this function returns the current frequency of the radio as a decimal in Hz back

/1000000: the frequency of the radio is divided by 1000000 to determine the frequency in MHz

The “ExportScript.ProcessIkarusDCSConfigHighImportance()” function is often called as the “ExportScript.ProcessIkarusDCSConfigLowImportance()” and therefore more suitable for time-critical values.

The two functions “ExportScript.ProcessDACConfigHighImportance()” and “ExportScript.ProcessDACConfigLowImportance()” corresponding to the two above-mentioned functions and are responsible for the output to the hardware.

Here, the corresponding data are read from the simulator, optionally worked up again, and then output to the hardware.

All here already entered data queries/issues are sent to everyone in the “Config.lua” file registered D.A.C. installations. Here only data is output that can be displayed with a Arcaze USB controller. These are LEDs and 7 Seqment displays.

The process of the data is as described above. However, the data on the “ExportScript.Tools.SendDataDAC()” exported. All destination IP/Ports from the “Config.lua” file obtain the same data supplied.

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.

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

The first two functions provide information about the data that are sent to Ikarus.

These values are determined by DCS own features optionally converted and then output via Ikarus.

The two functions “ExportScript.ProcessIkarusFCLowImportanceConfig()” and

“ExportScript.ProcessDACConfigLowImportance()” containing information on the data about D.A.C. be issued.

The data is output from the DCS own functions into functional groups or devices. For this reason, individual devices have specific features creates the work up and output the respective data of the aircraft.

All functions/tools that are used by more than one aircraft in the FC_AuxiliaryFuntions.lua file.

For example, the function “ExportScript.AF.FC_SPO15RWR()” gives the information of the SPO-15 radar warning system back. Using this information it is possible an authentic SPO-15 to assemble.

Many device-specific functions have an optional parameter, namely the type of function that indicates whether the data for Ikarus or D.A.C. are determined.

As with the DCS modules all data query are here already stored can be transmitted by means of the D.A.C. Arcaze USB controller.

All Flaming Cliffs files are similar and sufficiently commented.

Unfortunately, having the US aircraft from Flaming Cliffs many ads with display, so not as much data can be output as in the Russian aircraft over the hardware. As an example, called the radar warning system, which represents the information in a way monitor in the US aircraft.

Generic Radio for DCS-Modules

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

This function can be useful 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)
4 x pushbuttons
4 x rotary encoder
optional
4 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 other 6 digits show the frequency.

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 frequency. (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).

One encoder change the preset-channel.

Two encoder are used to chose the frequency (1x before the comma, 1x after the comma).

The Last Encoder is necessary to change the volume.

The optional LEDs are useful to show the power-status, the squelch-status, the mode and optionally loading the presets status 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, load) 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

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 “Tools.lua” file.

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

An example of “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"
    }
}
```

You can see the structure of the value from the “ExportScript.Config.DAC” variable. The function always gives the type and the value back.

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

Most useful is the “ExportScript.Tools.dump()” function in combination with the “ExportScript.Tools.WriteToLog()” function. So you can get a logfile with all the data you need.

e.g. ExportScript.Tools.WriteToLog(ExportScript.Tools.dump(ExportScript.Config.DAC))

Error-messages

If there is a Error in one of the LUAs, during start of the simulation an error-message 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
'ExportScript.ProcessIkarusDCSConfigLowImportance'
    [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)

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

If no error-message appears in the log file, but still no data export takes place, which can activate the debug mode in the “Config.lua” provide more information.

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

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<MODUL>\Cockpit\Scripts\“.

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 description 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.

Which data come now where?

All IDs (arg_number) from the mainpanel_init.lua-file has to be inserted to the “ExportScript.ConfigEveryFrameArguments” 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 (in-line comment) comes the name of the display.

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

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 necessary 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 “ExportScript.ConfigArguments” 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 (in-line 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.

There is however a little more complicated.

In <id> </ id> is a serial number comes in, starting with the first

In <DeviceID> </ DeviceID> matching the device Device ID comes from the “devices.lua” file, for example, 12

In <buttonId> </ buttonId> the associated button is number, for example, 1.

In <Discription> </ Discription> an apt description comes making the switch.

In <DcsID> </ DcsID> DCS argument number comes.

```
<Clickabledata>
    <ID>1</ID>
    <DeviceID>12</DeviceID>
    <ButtonID>1</ButtonID>
```

```
    <Discription>Emergency Jettison External Stores</Discription>
    <Type>Switch</Type>
    <DcsID>101</DcsID>
</Clickabledata>
```

The type “Switch” is used because the “Emergency Jettison External Stores” is a button. This entry is necessary 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 possibility to edit the table with all this data via the „MasterData“-Tab in D.A.C.

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 program code at the end of the “ExportScript.ProcessDACConfigLowImportance()” 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)
    ExportScript.Tools.WriteToLog(ltmp2..': '..ExportScript.Tools.dump(ltmp1))
    ExportScript.Tools.WriteToLog(ltmp2..' (metatable):
'..ExportScript.Tools.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 module. (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<USER>\Saved Games\DCS\Logs\Export.log”) and check all the entries with the available 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 (Navigation 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 available 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 “ExportScript.ProcessDACConfigLowImportance()” in the export script, to get better performance.