1 Notice

This is a universal export script for DCS. Simultaneous export of data to virtual cockpit software and I/O hardware or their control software is possible.

At the moment the export to the following software is supported. * D.A.C. (DCS Arcaze Connector) (DAC in GitHub) to access the Arcaze USB controllers (http://wiki.simple-solutions.de/de/products/Arcaze/Arcaze-USB) * Ikarus from H-J-P, our virtual cockpit software

and other software/hardware that receive their data via a UDP network connection and process the data in a Key=Value format.

The DCS ExportScript package is divided into two rough 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", "genericRadio.lua", "Maps.lua" and "utf8.lua" in the subfolder "lib" there are responsible for the entire control and logic of the data export.

The module-specific scripts are located in the folder "ExportsModules" in the path "%USERPROFILE%\Saved Games\DCS\Scripts\DCS-ExportScript\ExportsModules\".

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

The basic settings are made in the "Config.lua" file.

The modules to be exported can be extended quite easily. All you have to do is write all the information about the data to be exported to a new file. This file must be saved under the name of the corresponding module in the "ExportsModules" folder. How the files must be structured is described below, or is documented in the existing files.

The DCS ExportScript supports the single and multiplayer mode, as well as changing aircraft while the simulation is running (RAlt + J).

---

2 Installation

> **IMPORTANT**: Please make a backup copy of any existing files beforehand.

Download the DCS ExportScripts by clicking on the green button "Clone or download" and "Download ZIP".

Temporary unpacking of the zip-archive on the desktop.

Copy the contained folder "Scripts" to the file path "%USERPROFILE%\Saved Games\DCS\".

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

If an "Export.lua" file of another software already exists, it must not be overwritten. In this case, the following lines must be added to the end of this file.

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

---

3 Configuration

> **IMPORTANT**: To edit the LUA files please use a reasonable editor, e.g.Notepad++

The "Config.lua" file contains the basic settings for the export.

Default settings:

```
ExportScript.Config                  = {}
ExportScript.Version.Config          = "1.1.0"
```

```
-- Ikarus a Virtual 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.SocketDebug            = false
ExportScript.Config.FirstNewDataSend       = true
ExportScript.Config.FirstNewDataSendCount  = 5
```

The configuration block is divided into 4 areas.

The first area is intended for the "Ikarus" software.

The variable "ExportScript.Config.IkarusExport" specifies whether data should be exported to Ikarus. Here you can specify "true" or "false".

The variable "ExportScript.Config.IkarusHost" specifies to which IP address the data should be sent. Here you can enter the IP address "127.0.0.1" (for localhost) or another valid IP address of the target computer.

The variable "ExportScript.Config.IkarusPort" specifies to which port the data should be sent. The port "1625" is the default port of Ikarus and does not need to be changed. If another port is specified here, this must also be done within the Ikarus configuration.

The second area is intended for the D.A.C. software to which data is to be exported. Possible IP addresses and ports for different installations of the software must be specified.

The variable "ExportScript.Config.DACExport" specifies whether data should be exported to (all) D.A.C. installations. Here you can specify "true" or "false".

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] = {}" must not be changed.

The variable "ExportScript.Config.DAC[1].Host" specifies the IP address to which the data is sent. Here you can enter the IP address "127.0.0.1" (for localhost) or another valid IP address of the target computer.

The variable "ExportScript.Config.DAC[1].SendPort" specifies the port to which the data is to be sent. The port "26026" is the default port of D.A.C. and does not need to be changed. If another port is specified here, this must also be done within the D.A.C. configuration.

The variable "ExportScript.Config.DAC[1].Separator" specifies the separator between the individual Key=Value pairs. For D.A.C., the default separator is the colon":", otherwise all characters except the"=" character are allowed.

For a further hardware export to another computer, these 4 lines are entered again under the already existing lines and supplemented by the respective data.

> **IMPORTANT**: It is important that the number in the square brackets is replaced by the next higher number.

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, an almost unlimited number of different hardware/software combinations can be addressed.

The third area concerns the data reception of Ikarus and D.A.C.

The "ExportScript.Config.Listener" variable specifies whether data should be received and

processed by the DCS ExportScript. Possible values are "true" and "false".

The variable "ExportScript.Config.ListenerPort" specifies the port on which the DCS ExportScript receives the data from Ikarus/D.A.C.. The port "26027" is the default port of Ikarus/D.A.C. and does not need to be changed.

The fourth range specifies general values and does not usually need to be changed.

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.SocketDebug            = false
ExportScript.Config.FirstNewDataSend       = true
ExportScript.Config.FirstNewDataSendCount  = 5
```

The individual values mean the following:

ExportScript.Config.ExportInterval: specifies the time interval in seconds in which the time-critical data is updated (default 0.05)

ExportScript.Config.ExportLowTickInterval: specifies the time interval in seconds in which the other data is updated (default 0.5).

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

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 using the "WriteToLog()" function.

ExportScript.Config.SocketDebug: indicates if all data should be logged to the socket connection, including all data sent and received via network.

ExportScript.Config.FirstNewDataSend: indicates whether all export data should be resent after the start of the simulation, this is important if no initial data is available after the start.

ExportScript.Config.FirstNewDataSendCount: specifies when the export data should be sent again, the value is not a time specification but a counter that counts how often the export script has delivered data.

> **IMPORTANT**: All configuration information must be specified in the default format.

4 ExportsModules

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

These files contain all exportable data of the respective modules.

Here are a few examples of a DCS and a Flaming Cliffs module.

A-10C.lua

The A-10C is a fully simulated DCS module. These modules contain very precisely simulated individual systems and a clickable cockpit. As a result, all information of individual devices or displays can be output for these modules.

Basically, the files for the DCS modules are divided into several blocks.

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

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

For example, the content 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 comment: AOA (as description what the value for data indicates)

The AOA value is listed under ID 4 at A-10C. This value is output as a floating point value with 4 decimal places. With the software, the whole thing arrives as "4=0.5486".

The variable "ExportScript.ConfigEveryFrameArguments" contains information about the time-critical values, e.g. the values of the instruments and status lamps.

The variable "ExportScript.ConfigArguments" contains information about the other values, e.g. switch settings.

The functions "ExportScript.ProcessIkarusDCSConfigHighImportance()" and "ExportScript.ProcessIkarusDCSConfigLowImportance()" can contain information about values that are calculated or determined with special functions. The values determined here are output to Ikarus.

For example, the radio frequencies are determined here.

```
local lUHFRadio = GetDevice(54)
ExportScript.Tools.SendData(2002, 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 the UHF radio in the A-10C. The Device ID can be found 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 out, converted, formatted and transferred to Ikarus with the function ExportScript.Tools.SendData().

ExportScript.Tools.SendData(): is the function that transfers the data to Ikarus.

2002: is the key under which Ikarus receives/expects the data

string.format(): is an LUA function to format data (see the officialLUA documentation)

"%7.3f": Format Description, the value to be formatted is output as floating point value with a total of 7 characters (including the decimal point) and 3 decimal places.

lUHFRadio:get_frequency(): the special function get_frequency() of the UHF Radio Device is called, this function returns the currently set frequency of the radio as decimal number in Hz

100000000: the frequency of the radio is divided by 1000000 to determine the frequency in MHz

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

The functions "ExportScript.ProcessDACConfigHighImportance()" and "ExportScript.ProcessDACConfigLowImportance()" correspond to the two functions mentioned above and are responsible for output to the hardware.

Here the corresponding data is read out from the simulator, processed again if necessary and then output to the hardware.

All data queries/outputs entered here are sent to all DAC installations entered in the "Config.lua" file. Only data that can be displayed with an Arcaze USB controller is output here. These are LEDs and 7 sequence displays.

The data is processed as described above. However, the data is exported using the "ExportScript.Tools.SendDataDAC()" function. All destination IP/ports entered in the "Config.lua" file receive the same data.

SU-25T.lua

The Su-25T corresponds to a Flaming Cliffs module and is described here as an example of all available Flaming Cliffs aircraft.

The Flaming Cliffs aircraft differ not only in the simulation depth and handling from the full DCS modules, but also in the amount of data that can be exported from these aircraft.

For this reason, the files for the Flaming Cliffs aircraft are structured differently from the A-10C file, for example.

The Su-25T file consists of 4 large blocks and various help functions.

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

The first two functions contain information about the data sent to Ikarus.

These values are determined via DCS own functions, converted if necessary and then output via Ikarus.

The two functions "ExportScript.ProcessIkarusFCLowImportanceConfig()" and "ExportScript.ProcessDACConfigLowImportance()" contain the information about the data output via D.A.C..

The data is output by DCS's own functions in function groups or by device. For this reason, individual devices specific functions were created which process and output the respective data of the aircraft.

All functions/devices used by more than one aircraft can be found in the file FC_AuxiliaryFuntions.lua.

For example, the ExportScript.AF.FC_SPO15RWR() function returns the information from the SPO-15 radar warning system. With the help of this information it is possible to build an authentic SPO-15.

Many device specific functions have an optional parameter, namely the function type, which indicates whether the data is intended for Ikarus or D.A.C..

As with the DCS modules, all data queries that can be transferred to the Arcaze USB controller with the help of DAC are already stored here.

All Flaming Cliffs files have a similar structure and are sufficiently commented.

Unfortunately, the US aircraft from Flaming Cliffs have many displays with displays, so that not as much data can be output via the hardware as with the Russian aircraft. One example is the radar warning system, which displays the information on a kind of monitor in US aircraft.

---

5 Generic Radio for DCS-Modules

For some of the DCS modules there is a "genericRadio" in the form of a special function which is configured and called within the ExportScript.

With the help of the generic radio, the different radios of an aircraft can be operated via one hardware.

This function works exclusively with D.A.C., as some special functions of D.A.C. and the Arcaze USB controller are used here.

The following hardware is required:

1 x Arcaze USB controller 1 x Display Driver 3 or 32 1 x 8x display board with 8 displays (7 segment displays) or 1 x 2gang and 1 x 6gang display board with 8 displays 1 x step rotary switch with at least 3 positions 4 x button 4 x rotary encoder

Optional 4 x low current LED

The radio to be displayed and/or configured is selected via the rotary switch.

One of the buttons is for activating/deactivating the radio. Usually the power supply is switched on.

The first two digits of the display show the Preset Channel (if available), the remaining 6 displays show the frequency.

If the radio has the option of selecting between Manual Frequency Input and Preset Channel, another button can be used to switch between them. If the Preset Channel setting has been activated, the display shows the respective frequency of the preset channel (this does not correspond to the display in the simulation).

The third button is for activating squelch.

The fourth button is for loading or saving preset channels (depending on the module and radio).

A rotary encoder sets the preset channel.

Two rotary encoders are responsible for the frequency setting, one setting the digits before the decimal point and the other setting the digits after the decimal point.

The fourth rotary encoder is responsible for volume control.

The four LEDs indicate the power, the preset, the squelch and if necessary the loading of the preset status of the radio.

Some radios do not support all functions of the generic radio, sometimes it is the other way around.

For the settings within DAC, please note that the four buttons (Power, Preset, Squelch, Load) and the rotary switch do not transmit an Off value.

The four rotary encoders must be set to a value range of 0.0 to 2.0, the increment must be 0.1.

---

6 Auxiliary functions

The individual module export files contain various (partly commented out) examples with special functions that are not provided by LUA. These functions are defined and described in the "Tools.lua" file.

Overview of the available help functions:

- ExportScript.Tools.dump()
- ExportScript.Tools.WriteToLog()
- ExportScript.Tools.trim()
- ExportScript.Tools.ltrim()
- ExportScript.Tools.rtrim()
- ExportScript.Tools.subst()
- ExportScript.Tools.negate()
- ExportScript.Tools.round()
- ExportScript.Tools.split()
- ExportScript.Tools.getListIndicatorValue()
- ExportScript.Tools.RoundFreqeuncy()
- ExportScript.Tools.DisplayFormat()

ExportScript.Tools.dump()

The function "ExportScript.Tools.dump()" is similar to the function var_dump() from PHP and returns information about the passed value. "ExportScript.Tools.dump()" can handle all LUA types and returns detailed information about each value.

This may look like this, e.g. output of the 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"
    }
}
```

Here you can see how the content of the variable "ExportScript.Config.DAC" is structured. For the individual values, the type of data and the data itself are always specified.

This is important if it is unclear how the data is to be processed further.

ExportScript.Tools.WriteToLog()

The "ExportScript.Tools.WriteToLog()" function writes the transferred data to the Export.log file.

For example, ExportScript.Tools.WriteToLog(ExportScript.Tools.dump(ExportScript.Config.DAC))

ExportScript.Tools.trim()

The "ExportScript.Tools.trim(String)" function returns a string where the leading and trailing spaces in the given string have been removed.

ExportScript.Tools.ltrim()

The "ExportScript.Tools.ltrim(String)" function removes the leading spaces from the given string and returns them as a string.

ExportScript.Tools.rtrim()

The "ExportScript.Tools.rtrim(String)" function removes the trailing spaces from the given string and returns them as a string.

ExportScript.Tools.negate()

The function "ExportScript.Tools.negate(Number)" returns the negated value of the given value (Number).

ExportScript.Tools.round()

The function "ExportScript.Tools.round(Number, Decimals, Method)" returns the rounded value of the given value (Number). The second specification (decimals) specifies how many digits are to be rounded. The third specification (method) specifies the type of rounding, possible values are: "ceil" Returns the smallest integer greater than or equal to the value passed. "floor": Returns the smallest integer less than or equal to the passed value.

ExportScript.Tools.split()

The "ExportScript.Tools.split(String, Delimiter)" function returns a table containing the character strings separated from the given character string (String) by the delimiter passed.

ExportScript.Tools.getListIndicatorValue()

The "ExportScript.Tools.getListIndicatorValue(IndicatorID)" function returns the contents of the specified indicator (display). The indicator is specified by the corresponding ID. The ID can be determined as specified under point 12 Determining the contents of cockpit displays. The return value is a table with the values that are addressed under the respective index. The return value should always be checked for nil before further processing, see example KA-50 UV-26 Display:

local ReturnValue = ExportScript.Tools.getListIndicatorValue(7)

```
if ReturnValue ~= nil and ReturnValue.txt_digits ~= nil then
    ... ReturnValue.txt_digits
ending
```

The contents of the List_Indicator(7) are as follows.

```
"---------------------------------------
txt_digits
64
"
```

The actual value can be accessed using the table index "txt_digits" (ReturnValue.txt_digits).

ExportScript.Tools.RoundFreqeuncy()

The function "ExportScript.Tools.RoundFreqeuncy(Freqeuncy, Format, PrefixZeros, LeastValue)" returns the correctly formatted radio frequency. The following values must/can be specified. * Frequency: MHz/KHz * Format: e.g. "7.3" for 7-character value (incl. decimal point) with 3 decimal places (127,000) or "5.2" for 5-digit value (incl. decimal point) with 2 decimal places (64.00) (default "7.3") * PrefixZeros: fill with leading zeros (default false) * LeastValue: Minimum frequency value (default 0.025 (MHz))

ExportScript.Tools.DisplayFormat()

The "ExportScript.Tools.DisplayFormat(String, maxChars, LEFTorRight, DAC)" function formats a string for output in an Ikarus display.

The following values must/can be specified.

String: the actual value to be displayed maxChars: how many characters the display in Ikarus displays (default 5) LEFTorRight: "l" or "r" for left or right alignment of the text in the display (default r) DAC: true or false for whether the output should be via DAC (default false)

7 Error-messages

If there is an error in one of the LUA files, a corresponding error message appears in the "DCS.log" file when the simulator is started.

Example:

```
...
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>.
...
```

The error message indicates the (almost complete) path to the file in question. Then follows the line number in which the error occurred. After that there is a more or less exact error description.

The lines below indicate which functions were called until the error occurred.

This information can be used to locate and correct the error.

If no error message appears in the log file, but no data export is performed, activating the debug mode in "Config.lua" can provide more information.

---

8 Additional infos

For more information on LUA, please visit the LUA homepage athttp://www.lua.org/manual/5.1/

Information about the devices or the IDs of the values can be found in the files "device.lua" and "mainpanel_init.lua" in the path "...\Eagle Dynamics\DCS World\Mods\aircrafts\\Cockpit\Scripts\".

---

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 following three files are of interest there:

- devices.lua
- mainpanel_init.lua
- clickabledata.lua

The files contain the following contents:

mainpanel_init.lua

Contains all instruments and warning lamps, usually all of which are clearly marked.

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
```

The following information is of interest here:

SAI_Pitch: To be used as designation of the device and the value displayed with it.

SAI_Pitch.arg_number = 63: The 63 is the ID under which the data will be requested later.

SAI_Pitch.output = {-1.0, 1.0}: -1.0, 1.0 the value range of the returned data.

If necessary, some definitions are stored in extra files and are then imported via a dofile() call.

devices.lua

Contains the device names and the corresponding IDs (it can happen that the IDs in the comment are not correct, then simply count from above).

e.g.

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

ELEC_INTERFACE: Electrical interface, has the ID 1

clickabledata.lua

Contains all information about buttons, switches, rotary switches and knobs. Usually there is a block with function descriptions at the beginning of the file, here the individual switches are described. Entries labeled "Cover" are not interesting, these are only the switch covers.

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} }
```

The following information is of interest here: class_type.BTN: means that it is a button, this information is only of interest for understanding.

_("OSB 1"): This is the English name that is displayed when you move the mouse over the button in the cockpit. This name should be used as part of the description.

devices.MFCD_LEFT: The device to which the button belongs is the Left MFCD (MFCD_LEFT). Take the Device Name as part of the description. Via the Device name you get the Device ID from the file "device.lua".

arg = {300}: the 300 is the ID under which the current value of the button can be read. This is important to check the corresponding switch position.

action = {device_commands.Button_1}: The 1 from Button_1 is of interest. Occasionally the switch at stop_action has a different button number, then it has to be remembered.

arg_value = {1.0}: This is the value that is sent when the button is pressed.

arg_lim = {{0.0, 1.0}: This is the possible value range, i.e. 0.0 if the button is not pressed and 1.0 if the button is pressed.

or

```
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 is a rocker switch on the left MFCD, roughly similar to the above, except the following things:

action = {MFCD_ADJ_Increase}: MFCD_ADJ_Increase is a variable, via the search in the file one finds the suitable button number, in this case 21.

stop_action = {MFCD_ADJ_Stop}: MFCD_ADJ_Stop is also a variable, this time it contains the number 23

or

```
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 stands for rotate.

_("DAY/NIGHT/OFF"): Here you see three names. If you know the switch, then you know that it has only three switching positions.

arg = {325, 325}: Twice the ID of the rotary switch, since it can be turned to the left and right.

arg_value = {0.1, -0.1}: These are the values that are sent when you press. Rotate to the left 0.1 and to the right -0.1

arg_lim = {{0.0, 0.2}, {0.0, 0.2}}}: 0.0 to 0.2 is the possible value range of the switch. 0.0 is the start value, as already seen with arg_value, either 0.1 is added or subtracted. So the possible values are 0.0, 0.1 and 0.2, which also corresponds to the three described switching positions.

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 function name that describes the switch, or in this case the knob.

Here you have to look at the function to see which value has which purpose.

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

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

    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},
            }
ending
```

("Adjust Display Brightness"): is the description of the knob.

devices.CMSP: The device to which the knob belongs, here the CMSP.

device_commands.Button_9: The button number again.

359: Is the value of arg, i.e. the ID under which the current value of the knob can be queried.

0.1: Is the value of gain, this means that the value is increased or decreased by 0.1 depending on the direction of rotation.

false, false: These values are of no interest.

0.15, 0.85}: These are the values of arg_lim, i.e. the possible value range of the knob. In this case from 0.15 to 0.85, that is 70 possible values with an accuracy of 0.1.

10 Which data goes where?

First, a copy of the file "Empty-DCS.lua" is created under the name of the module (aircraft). The values found are entered in this new export file. Furthermore, a copy of the file "Empty-DCS.xml" is created under the name of the module (aircraft). The values found are also entered in this XML file.

All IDs (arg_number) from the mainpanel_init.lua file come into the "ExportScript.ConfigEveryFrameArguments" variable (of type Table) in the export file.

The corresponding format results from the specified value range, but is usually a floating point number.

The description (in the form of a comment) is the name of the display.

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

The same IDs are also placed in the corresponding XML file in the "DCS_ID" area.

Copy the empty data block as a template a few times beforehand.

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

```
    <Type>Lamp</Type>
</DCS_ID>
```

With Type, "Lamp" is specified here, since "APU FIRE" is a status lamp. By specifying the type, the parameter appears within D.A.C. and Ikarus in the corresponding area for lamps (LEDs).

A further type is "Display". This type is only intended for 7-segment displays (e.g. radio frequencies), so that the entries appear in the corresponding area.

All switch/controller IDs from the "clickabledata.lua" file are placed in the "ExportScript.ConfigArguments" variable (of type Table) in the export file.

The corresponding format results from the specified value range, but is usually a floating point number.

The description is the name of the display.

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

The same data is also transferred to the corresponding XML file in the "Clickabledata" area.

Copy the empty data block a few times beforehand.

But here it gets a little more complicated.

In `<ID></ID>` a consecutive number comes in, starting at 1.

In `<DeviceID></DeviceID>` the device ID matching the device comes from the "devices.lua" file, e.g. 12.

In `<ButtonID></ButtonID>` comes the corresponding button number, e.g. 1.

In `<Discription></Discription>` comes a suitable description of what the switch does.

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

```
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 is "Switch", because "Emergency Jettison External Stores" is a switch.

By specifying the type, the parameter appears within DAC and Icarus in the corresponding range.

Another type is "Rotary". This type is only intended for axes (e.g. default_axis_limited), so that the entries appear in the corresponding area.

All Device IDs from the "devices.lua" file are placed in the "Devices" block, preferably in the same order as they are in the Device.lua file.

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

> **HINWEIS**: All data can also be entered within D.A.C. in the "MasterData" tab in the corresponding tables and then saved under the corresponding module name.

11 Special functions of the individual Devices

Some devices in the simulated aircraft have special functions with which certain values can be read out.

To find these functions, the following code area must be activated once (remove comment characters at the end of the "ExportScript.ProcessDACConfigLowImportance()" function).

```
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)))
ending
```

MAX-ID must be replaced by the highest ID from the device.lua file.

Then start the game with the appropriate module, preferably so that the aircraft is on the tarmac with the machine started. If the game only runs for a few seconds, the simulation can be stopped again.

Now the Export.log file ("C:\Users\\Saved Games\DCS\Logs\Export.log") contains a long list of possible functions.

Here as an example an excerpt of the log at the A-10C. This is only a few example data to illustrate the structure.

```
46: {
    [link] = userdata: 000000000083285608
}

46 (metatable): {
    [__index] = {
        [listen_event] = "function: 00000000008EEE7360, C function"
        [listen_command] = "function: 00000000008EEE72C0, C function"
        [performClickableAction] = "function: 00000000008EEE7310, C function"
        [SetCommand] = "function: 00000000008EEE7270, C function"
    }
}

47: {
    [link] = userdata: 000000000083367118
}

47 (metatable): {
    [__index] = {
        [get_sideslip] = "function: 00000000008EEEA3C0, C function"
        [performClickableAction] = "function: 00000000008EEE9E30, C function"
        [get_bank] = "function: 00000000008EEEA320, C function"
        [listen_event] = "function: 00000000008EEE9E80, C function"
        [get_pitch] = "function: 00000000008EEEA070, C function"
        [listen_command] = "function: 00000000008EEE9EN0, C function"
        [SetCommand] = "function: 00000000008EEE9D90, C function"
    }
}
```

First is always the Device ID of the requested device. The first data block shows possible data that could be processed directly. Mostly the block contains only"[link] = userdata:", this is a reference to data that cannot be processed directly.

Then the Device ID comes back, followed by the note "metatable". This block specifies functions with which the data of the device can be accessed.

The ID 46 has no special functions, since the Device 46 is the NMSP (Navication Mode Select Panel). And this panel only has a few switches.

Things are a little different with ID 47. The ID 47 is the ADI (Attitude Direction Indicator), here are some values you can output. All functions starting with "get" are interesting.

That would be "get_sideslip", "get_bank" and " get_pitch". The descriptions already indicate which data is returned here.

```
54: {
    [link] = userdata: 000000000083377A68
}

54 (metatable): {
    [__index] = {
```

```
        [listen_command] ="function: 00000000005DB929A0, C function"
        [set_frequency] = "function: 00000000005DBA2540, C function"
        [is_on] = "function: 00000000005DB69A10, C function"
        [get_frequency] = "function: 00000000005DB830F0, C function"
        [performClickableAction] = "function: 00000000005DB90500, C function"
        [set_modulation] ="function: 00000000005DB86B90, C function"
        [set_channel] = "function: 00000000005DB90640, C function"
        [listen_event] ="function: 00000000005DB910A0, C function"
        [SetCommand] ="function: 00000000005DB72CC0, C function"
    }
}
```

The ID 54 of the A-10C is the UHF radio, here the two functions "get_frequency" and "is_on" are interesting.

This way many special functions can be determined with which many data can be returned via the DCS ExportScript.

How to use these functions must be tried out in individual cases, it is helpful to watch the procedure in the already existing export scripts.

It is best to use the special functions in the functions "ExportScript.ProcessIkarusDCSConfigLowImportance()" and "ExportScript.ProcessDACConfigLowImportance()" of the DCS ExportScript. In this case, the functions are not called as often and therefore do not generate a higher workload.

12 Determining the contents of cockpit displays

It is possible to read the contents of displays in the cockpit, e.g. frequency displays, digital clocks,... There is the restriction that only alphanumeric values can be read out and no graphics such as camera, sensor images or moving maps.

To find these functions, the following code area must be activated once (remove comment characters at the end of the "ExportScript.ProcessDACConfigLowImportance()" function).

```
list_indication get the value of cockpit displays
local ltmp1 = 0
for ltmp2 = 0, 20, 1 do
    ltmp1 = list_indication(ltmp2)
    ExportScript.Tools.WriteToLog(ltmp2...': '..ExportScript.Tools.dump(ltmp1))
ending
```

If necessary, the counter value (20) must be increased to capture all displays.

Then start the game with the corresponding module, preferably so that the aircraft is in the parking position with the machine started. If the game only runs for a few seconds, the simulation can be stopped again. If necessary, some devices must be switched through in order to have all displays in

the log once.

Now the Export.log file ("C:\Users\Saved Games\DCS\Logs\Export.log") contains a long list of possible functions.

Here as an example an excerpt of the log at the A-10C. This is only a few example data to illustrate the structure.

```
...
19:38:15:348 : 7: string: "----------------------------------------
txt_UP
240 120 000  A
----------------------------------------
txt_DOWN1
CHAF
----------------------------------------
txt_DOWN2
FLAR
----------------------------------------
txt_DOWN3
OTR1
----------------------------------------
txt_DOWN4
PROG
"

19:38:15:348 : 8: string: "----------------------------------------
txt_CHAFF_FLARE
A240s120
----------------------------------------
txt_JMR
OFF
----------------------------------------
txt_MWS
ACTIVE
"

19:38:15:348 : 9: string: ""
...
```

The Indicator ID 7 provides content of the CMSP display on the right console. The Indicator ID 8 provides contents of the CMSC display on the middle instrument panel. The Indicator ID 9 does not provide any content.

The function ExportScript.Tools.getListIndicatorValue() is the easiest way to read out the display data. This function creates a table with all contents that can be called up under the corresponding indexes. See the function description.

13 Quick start instruction:

Table of contents

Installation Instructions:

Broadly, you need to:

```
download the zip from github
unzip Ikarus
setup Ikarus to talk to DCS
run Ikarus
customize Ikarus panels
Fly!
```

1. Download Ikarus Go to: https://github.com/s-d-a/Ikarus Download the Zip via the green "Clone or Download" button on the top-ish right. This zip is all the files you can see in the github repository... zipped.

You also need to get the DCS-ExportScripts zip in the same way:
https://github.com/s-d-a/DCS-ExportScripts

1. Unzip Ikarus Ikarus does not need to be "installed". Unzip to the desktop or some other random place. As soon as you've unzipped Ikarus, you can run it, there is no install process.

The Zip contains is one folder which contains all the files and sub-folders, called "Ikarus-master". Have a look at the file structure - its exactly the same as the structure on github. This folder needs to be extracted to where you want to run Ikarus from.

To keep everything together, you could make a new \Users\%User%\Saved Games\DCS\Ikarus directory and copy/move your newly unzipped Ikarus folder there.

\Users\%User%\Saved Games\DCS\ or \Users\%User%\Saved Games\DCS.openalpha\ is where DCS saves all of its settings/configuration/keybindings. If your not comfortable putting files in that directory, you can absolutely put Ikarus somewhere else.

I have a few folders: \%User\Saved Games\DCS\Ikarus\Ikarus 1.2.3.1n\ \%User\Saved Games\DCS\Ikarus\Ikarus 1.2.3.1s\ \%User\Saved Games\DCS\Ikarus\Ikarus 1.2.3.1w\

Where I have kept the various versions. I can run each version from within its own directory.

Note: THERE IS NO SIMPLE UPGRADE PATH BETWEEN VERSIONS.

Keep your "production/modified" version separate and backed up. If you modify Ikarus, you will need to then merge those changes into any newer version you download. The more you customize, the harder this may become.

Each aircraft has an .xml file and an .ikarus file which change as you customize panels, as well as artwork used in the panels (which you can also modify as you wish).

1. Setup Ikarus to talk to DCS In \Users\%User%\Saved Games\DCS\Scripts there is your

"Export.lua". Back it up. Back it up now. Of course you only can back it up if it exists.

Export.lua controls loading modules so that DCS and import/export data in various formats. You need to tell Export.lua to use DCS-ExportScripts, and you need to place those scripts in \Users\%User%\Saved Games\DCS\Scripts.

Please note that "DCS-ExportScript" is Ikarus specific. Think of it as "DCS-Ikarus-ExportScript". It is not a general purpose exporter, and needs to co-exist with any other exporter you may have. TacView and DCS-BIOS have their own exporters for instance.

The Zip has the following directory structure: DCS-ExportScripts.zip --> DCS-ExportScripts-master --> Scripts --> DCS-ExportScript

Inside the zip, "Scripts" has the same structure as \Users\%User%\Saved Games\DCS\Scripts. Inside the zip, there is an Export.lua (that is Ikarus only).

If you do not have your own Export.lua, use the one from the zip. Grab Export.lua and the "DCS-ExportScript" folder and copy to \Users\%User%\Saved Games\DCS\Scripts.

If you have your own Export.lua already, add this line (and copy the DCS-ExportScript folder): Code:

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

This instructs DCS to use the DCS-ExportScript you had just copied. Use notepad++ or another good text editor. Plain notepad will probably be painful.

You need to end up with: \Users\%User%\Saved Games\DCS\Scripts\Export.lua (with the Ikarus config) \Users\%User%\Saved Games\DCS\Scripts\DCS-ExportScripts

1. Run Ikarus Run Ikarus.exe. At this point you dont need to run DCS, just play around with Ikarus.

You need to approve Ikarus's network access - Windows Firewall will pop up a window each time you start a new version of Ikarus. Ikarus can run on a remote PC (without display export) and uses network protocols to talk to DCS. If you run Ikarus on your DCS-local machine then you can export MFCD's - remember though that Ikarus still needs network access on the local machine to talk to DCS.

Hit "show panels" and they should appear - all of them. Then most panels will immediately disappear. You can actuate (click) the cockpit panel buttons with the mouse. On the main A-10C panel, there are 8 silver switches down the bottom - these switch between the different panels like the CDU, the UFC etc. You can mouse hover over most things to get descriptions.

1. Customize the Panels You need to put in new co-ordinates if you want to move the panels around or move them to your second screen. You will need to do this as they default to 0,0 on the main DCS viewport. You cannot drag and drop cockpits or panels - you must hand-enter the co-ordinates. If you select "editor mode" then "show cockpit" you can drag and drop panel elements around to re-position them (gauges, buttons, knobs etc).

If you export DCS MFCD's or other displays, then you will need to either arrange/resize Ikarus to suit, or, move the displays to suit Ikarus and probably both.

1. Fly!

A few things:

```
You dont need a touchscreen, Ikarus will work with a mouse.
There are buttons in the cockpit to switch between panels. Only the main panel is
displayed at first. There is no way to switch between panels using the Ikarus
config window.
Ikarus auto-detects aircraft and displays the panels to suit
You can start and stop Ikarus during a mission and DCS will just work it out. You
can start Ikarus after DCS or before.
There is no "local machine" mode in Ikarus. If you run Ikarus on your DCS PC then
Ikarus needs to talk to 127.0.0.1 (locahost).
Ikarus has a hierarchy, there is the Cockpit, there are "Panels" and then there
"gauges/buttons".
```