DCS-Bios F-16 DED Display Export Technical Specification

Purpose

Add the DED Display to the DCS-Bios Export for the F-16C module

Primary Limitations

Primary consumer of the export will be Arduino class devices with limited processing power and storage space – to this end the data exported should be pre-processing in advance into individual lines that then be displayed as supplied by the Arduino device.

(If the DCS-Bios code is ever ported to higher powered devices like the Raspberry Pi then these limitations may be largely avoided in which case supplying the individual fields in the DCS-Bios export and then controlling the displaying of these fields on the end device may be preferable)

The DED device has an complication that it uses inverse characters in this display and experiments in using the font background and foreground colour for the inverse characters has taken too long with the Adafruit GFX library to be practical (this may not be an issue with other Arduino Graphics libraries but due to display issues that remains untested).

Additionally the display uses custom characters which are not part of the standard ASCII or extended ASCII character set.

As an attempt to resolve both these issues – and to replicate the font used on the real world display as closely as possible custom Arduino fonts have been developed based on the True Type font available here - https://fontstruct.com/fontstructions/show/1014500/falconded

However there is an ongoing problems reading ASCII character values about 127 from the DCS-Bios feed – the characters appear to be present on the Serial feed but are not being received correctly by the Arduino device and identifying and resolving the issue has been unsuccessful so far, so this code is not currently included in release version of the export script.

Implementation

The data for the DED display is exported by DCS in a standard List Indicator container – (currently List Indication 6 in DCS v2.7.0.46250)

A sample of this output generated using the DCS Witchcraft lua plug-in (in this case for the IFF STAT page) looks like –

```
"----------------------------------------\ IFF label\ IFF\ --
----------------------------------------\ IFF Power Status\ ON
\ ----------------------------------------\ Mode label\ STAT\
----------------------------------------\ M1
Mode_placeholder\ \ children are {\ ------------------------
---------------\ M1 Mode\ M1\ }\ ----------------------------
------------\ M2 Mode_placeholder\ \ children are {\ ---------
-------------------------------\ M2 Mode\ M2\ }\ ------------
----------------------------\ M3 Mode_placeholder\ \ children
are {\ ----------------------------------------\ M3 Mode\ M3\
}\ ----------------------------------------\ M1 Lockout
Status\ :\ ----------------------------------------\ M2
Lockout Status\ :\ ----------------------------------------\
M3 Lockout Status\ :\ --------------------------------------
-\ M1 Code\ 42\ ----------------------------------------\ M2
Code\ 6174\ ----------------------------------------\ M3
Code\ 1337\ ----------------------------------------\ M4
Mode_placeholder_inv\ \ children are {\ ----------------------
-------------------\ M4 Mode_inv\ M4\ }\ ---------------------
--------------------\ M4 Key\ (6)\ ----------------------------
-------------\ M4 Monitoring\ OUT\ --------------------------
---------------\ M4 Monitoring Key\ (7)\ ---------------------
--------------------\ MC Mode_placeholder\ \ children are {\ -
----------------------------------------\ MC Mode\ MC\ }\ ----
-----------------------------------\ MC Code\ \ ------------
----------------------------\ MC Key\ (5)\ ------------------
----------------------\ MS Mode_placeholder\ \ children are
{\ ----------------------------------------\ MS Mode\ MS\ }\
----------------------------------------\ MS Key\ (8)\ ------
---------------------------------\ Mode Asterisks_lhs\ *\ --
----------------------------------------\ Mode Asterisks_rhs\
*\ ----------------------------------------\ Mode
Scratchpad_placeholder\ \ children are {\ --------------------
---------------------\ Mode Scratchpad\ \ }\ -----------------
-----------------------\ POS EVENT - Side\ N\ ---------------
--------------------------\ POS EVENT - OF\ OF\ --------------
---------------------------\ POS EVENT - Number\ 1\ ----------
-----------------------------\ TIM EVENT - Time\ 01:23\"
```

This data is read using the standard Parse_Indication function which breaks the string down into a list of Name, Value pairs using the ----------\ as separators as follows:

```
{
["M2 Code"] = "6174",
["MC Mode_placeholder"] = "",
["M2 Lockout Status"] = ":",
["M4 Monitoring Key"] = "(7)",
```

```
["MS Mode"] = "MS",
["IFF label"] = "IFF",
["Mode Asterisks_lhs"] = "*",
["Mode Asterisks_rhs"] = "*",
["M2 Mode"] = "M2",
["M4 Mode_placeholder_inv"] = "",
["POS EVENT - Number"] = "1",
["POS EVENT - OF"] = "OF",
["M1 Mode_placeholder"] = "",
["M4 Mode_inv"] = "M4",
["Mode Scratchpad"] = " ",
["M1 Lockout Status"] = ":",
["M3 Lockout Status"] = ":",
["Mode Scratchpad_placeholder"] = "",
["M3 Code"] = "1337",
["IFF Power Status"] = " ON ",
["MS Mode_placeholder"] = "",
["M3 Mode"] = "M3",
["M4 Monitoring"] = "OUT",
["MC Mode"] = "MC",
["M1 Mode"] = "M1",
["Mode label"] = "STAT",
["M2 Mode_placeholder"] = "",
["TIM EVENT - Time"] = "01:23",
["MC Code"] = "",
["M1 Code"] = "42",
["MC Key"] = "(5)",
["POS EVENT - Side"] = "N",
["M4 Key"] = "(6)",
["M3 Mode_placeholder"] = "",
["MS Key"] = "(8)",
}
```

This list contains all the data contained in the DED display for all 5 lines, but the DCS-Bios export requires individual lines, so in order to break this data down line by line we have 5 list objects DEDLayout_l1 -> DEDLayout_l5 which contain the Name of each data item that could apprear on that line and for the Value a list of position and formatting information for that item.

The Corresponding Line 3 items for the IFF Page are –

```
--IFF
DEDLayout_l3["STAT M1 Mode"]={0,2,0,"_inv","I"}
DEDLayout_l3["STAT M1 Lockout Status"]={3,1}
DEDLayout_l3["STAT M1 Code"]={4,2}
DEDLayout_l3["STAT M4 Mode"]={8,2,0,"_inv","I"}
DEDLayout_l3["STAT M4 Code"]={12,1}
DEDLayout_l3["STAT M4 Key"]={14,3}
DEDLayout_l3["STAT POS EVENT - Side"]={19,1}
DEDLayout_l3["STAT POS EVENT - OF"]={20,2}
DEDLayout_l3["STAT POS EVENT - Number"]={22,1}
DEDLayout_l3["POS M1 Mode"]={1,2,0,"_inv","I"}
DEDLayout_l3["POS M1 Lockout Status"]={3,1}
DEDLayout_l3["POS M1 Code"]={4,5}
DEDLayout_l3["POS M4 Mode"]={9,2,0,"_inv","I"}
DEDLayout_l3["POS M4 Code"]={13,1}
DEDLayout_l3["POS M4 Key"]={14,2}
DEDLayout_l3["POS Mode Asterisks_both"]={18,1,23,"","I"}
DEDLayout_l3["POS Mode Scratchpad"]={14,5,0,"_inv","I"}
DEDLayout_l3["TIM M1 Mode"]={1,2,0,"_inv","I"}
DEDLayout_l3["TIM M1 Lockout Status"]={3,1}
DEDLayout_l3["TIM M1 Code"]={4,5}
DEDLayout_l3["TIM M4 Mode"]={9,2,0,"_inv","I"}
DEDLayout_l3["TIM M4 Code"]={13,1}
DEDLayout_l3["TIM M4 Key"]={14,2}
DEDLayout_l3["TIM Mode Asterisks_both"]={18,1,23,"","I"}
DEDLayout_l3["TIM Mode Scratchpad"]={14,5,0,"_inv","I"}
DEDLayout_l3["BACKUP label"]={9,6}
```

It's important to note that in this case the Names of these line items do NOT directly line up with the Names from the DCS Export – this is because the names in the DCS Export aren't unique and different versions with the same name are potentially displayed differently.

For example M1 Mode appears on the STAT, TIM and POS version of the page but the formatting entry is different for the STAT version than for the TIM or POS version

```
DEDLayout_l3["STAT M1 Mode"]={0,2,0,"_inv","I"}
DEDLayout_l3["POS M1 Mode"]={1,2,0,"_inv","I"}
DEDLayout_l3["TIM M1 Mode"]={1,2,0,"_inv","I"}
```

In order to handle this the LUA code searches for specific Names within the exported data that we can use to a) identify that this is an example of a page that potentially contains non-unique Names and b) allows us to identify which version of the page it is

```
-- Check for present of Objects that indicate Duplicate Key
Names that need resolving
      local guard = DED_fields["Guard Label"]
      local mode = DED_fields["Mode label"]
      local event = DED_fields["Event Occured"]
      local alow =  DED_fields["ALOW label"]
      local bingo = DED_fields["CMDS_BINGO_lbl"]
      local inflt_algn = DED_fields["INS_INFLT_ALGN_lbl"]
      local intraflight = DED_fields["INTRAFLIGHT lbl"]
      local dlnk_A_G = DED_fields["A-G DL lbl"]
      local nav_status = DED_fields["NAV Status lbl"]
```

So we then start looping through all the exported data items in the list returned by Parse_Indication and if we have found one of these data items that indicates duplicate names we append the value of that data item to the start of the name

```
--Loop through Exported DED Objects
      for k,v in pairs(DED_fields) do
-- Handle Duplicate Key Names on COM2 Guard page items
            if guard ~= nil then
                  label = guard.." "..k
-- Handle Duplicate Key Names on IFF STAT page items
            elseif mode ~= nil then
                  label = mode.." "..k
```

```
-- Handle Duplicate Key Names on IFF POS & TIM page items
            elseif event ~= nil then
                  label = event.." "..k
-- Handle Duplicate Key Names on ALOW page Line 1 items
            elseif alow ~= nil and line == 1 then
                  label = alow.." "..k
-- Handle Duplicate Key Names on CMDS Bingo page Line 1 items
            elseif bingo ~= nil and line == 1 then
                  label = bingo.." "..k
-- Handle Duplicate Key Names on INS INFL ALGN page Lines 1 & 3 items
            elseif inflt_algn ~= nil and (line == 1 or line==3) then
                  label = inflt_algn.." "..k
-- Handle Duplicate Key Names on DLNK INTRAFLIGHT page
            elseif intraflight ~= nil then
                  label = intraflight.." "..k
-- Handle Duplicate Key Names on DLNK A-G page Line 2 items
            elseif dlnk_A_G ~= nil and line == 2 then
                  label = dlnk_A_G.." "..k
-- Handle Duplicate Key Names on NAV page
            elseif nav_status ~= nil then
                  label = nav_status.." "..k
            else
                  label = k
            end
```

So in this case

```
["Mode label"] = "STAT"
```

And as a result

```
["M1 Mode"] = "M1"
```

Becomes

```
["STAT M1 Mode"] = "M1"
```

The next step in the loop is to find an item in the list of fields for a line number where the Name of the field matches the Name of the exported data (as updated in case of duplicates)

```
--Get layout data associated with current key
layout =
DEDLayoutLine[label:gsub("_inv","",1):gsub("_lhs","_both",1)]
```

Since nothing is ever simple with this we still need to do some manipulations to get the right field entry.

The first section of this - `label:gsub("_inv","",1` – looks for Names that end in "_inv" and ignores it when looking a comparison in the line data.

For example

```
["M4 Mode_inv"] = "M4",
```

Has already become

```
["STAT M4 Mode_inv"] = "M4",
```

And we now ignore the "_inv" in the name when comparing with the Line Data so it matches

```
DEDLayout_l3["STAT M4 Mode"]={8,2,0,"_inv","I"}
```

The second part is a bit weirder - `:gsub("_lhs","_both",1)` – this replaces anything that ends in "_lhs" with "_both", the reason we do this is that the "_lhs" and "_rhs" suffixes on the name are used to indicate pairs of asterisks on the display and to simplify the line data list we combine both the "_lhs" and "_rhs" entries into a single entry with the "_both" suffix

So

```
["STAT Mode Asterisks_lhs"] = "*",
["STAT Mode Asterisks_rhs"] = "*",
```

Connects to

```
DEDLayout_l2["STAT Mode Asterisks_both"]={17,1,23,"","I"}
```

So far I have put off discussing the format of the Value of the line data, but that now we have a Value that we need to add to the output its necessary to discuss this

The format of the Value consist of 5 values

1 – X position on Line to display the value (Required)

2 – Max Length of the Value to be displayed (Required)

3 – X Position on Line to repeat the value – this used for Asterisk values to indicate where the right hand asterisk appear (Optional)

4 – Format condition – if this value is present the string here must be present at the end the Name of the exported item in order for the formatting condition in position 5 to be applied (Optional)

5 – Formatting – this is the formatting to be applied to the item if either value 4 is empty or the string in value 4 is present at the end Name of the exported item (Optional)

There is also a 6th optional parameter which I don't think is currently used but this allows you to specify a static override value that will be displayed in place of the Value returned by the DCS Export.

Note - the formatting controlled by values 4 and 5 is currently not included in the release version due to problems accessing the necessary characters in the DCS-Bios data.

If we start with a simple example –

```
["M1 Code"] = "42",
```

Becomes

```
["STAT M1 Code"] = "42",
```

And this connects to

```
DEDLayout_l3["STAT M1 Code"]={4,2}
```

In this vase we only have the mandator fields populated and what this tells us that we need to display the Value (in this "42") at X-Position 4 on the DED line and it will be a maximum of 2 characters (the reason we need to know the length will be discussed shortly)

Moving up in complexity

```
["M4 Mode_inv"] = "M4",
```

Becomes

```
["STAT M4 Mode_inv"] = "M4",
```

And because we ignore the "_inv" on the end it links to

```
DEDLayout_l3["STAT M4 Mode"]={8,2,0,"_inv","I"}
```

Here we specify that the value is to be displayed at X-Position 4 and that it is a maximum of two characters long (parameters 1 & 2)

Parameter 3 is 0 which means it does nothing.

Parameter 4 is "_inv" which means that if the name of the exported item ends in "_inv" we need to apply the formatting in parameter 5 – the name of the exported item is "`STAT M4 Mode_inv`" so this format is to be applied (if the name was "`STAT M4 Mode`" then it wouldn't)

Parameter 5 is the formatting to be applies which in this case is "I" which means it's INVERSE characters.

As mentioned before formatting is not enabled in the release code.

Finally we have an Asterisk pair (in this case we need to look at data on Line 2 of this display)

```
["Mode Asterisks_lhs"] = "*",
["Mode Asterisks_rhs"] = "*",
```

Become

```
["STAT Mode Asterisks_lhs"] = "*",
["STAT Mode Asterisks_rhs"] = "*",
```

And the "`STAT Mode Asterisks_lhs`" only connects to

```
DEDLayout_l2["STAT Mode Asterisks_both"]={17,1,23,"","I"}
```

("`STAT Mode Asterisks_rhs`" has no corresponding line data so produces no output)

In this case we specify that the value is to be displayed at X-Position 17 and that it is a maximum of one characters long (parameters 1 & 2)

Parameter 3 is populated this time which means we no repeat the output and X-position 23 – this corresponds to the _rhs asterisk value we ignored.

Parameter 4 is empty so we always apply the formatting in parameter 5

Parameter 5 is the formatting to be applies which in this case is "I" which means it's INVERSE characters.

Again formatting is not actually enabled in the release code.

So now we know what we need to add to output line for each Exported Value (as mentioned before, if Value 6 is present then we use this instead of the export value)

```
--If layout value 6 is present then use this value to override
the value returned from DCS
if layout[6] ~= nil then
     value = layout[6]
else
     value = v
end

-- Add Value to dataLine using mergeString because some values
are are supposed to fit within others
dataLine = mergeString(dataLine, value, layout[1])
```

But there is an additional wrinkle that we don't have any control over the order the items are exported from DCS Bios in so we can't just concatenate the values together.

Instead we start with a string of spaces and merge the value we have calculated into that string.

Then finally if parameter 3 has been populated we repeat the merge process adding the Value atb the x-position in parameter 3

```
--If layout value 3 > 0 we need to duplicate this item at
position specific in value 3 (this is for "*"s marking
enterable fields
if layout[3] ~= nil and layout[3] > 0 then
      dataLine = mergeString(dataLine, value, layout[3])
end
```

After we've looped through all the items we now have now constructed our complete line and we can return that to the code that has called the process and ultimate this send the line to the DCS Bios output

```
local DEDLine1 = ""
local DEDLine2 = ""
local DEDLine3 = ""
local DEDLine4 = ""
local DEDLine5 = ""

-- Build DED Display Lines
moduleBeingDefined.exportHooks[#moduleBeingDefined.exportHooks
+1] = function()
      DEDLine1 = buildDEDLine(1);
      DEDLine2 = buildDEDLine(2);
      DEDLine3 = buildDEDLine(3);
      DEDLine4 = buildDEDLine(4);
      DEDLine5 = buildDEDLine(5);
end

-- Add DED Display Lines to data sent across
defineString("DED_LINE_1", function() return DEDLine1 end, 25,
"DED Output Data", "DED Display Line 1")
defineString("DED_LINE_2", function() return DEDLine2 end, 25,
"DED Output Data", "DED Display Line 2")
```

```
defineString("DED_LINE_3", function() return DEDLine3 end, 25,
"DED Output Data", "DED Display Line 3")

defineString("DED_LINE_4", function() return DEDLine4 end, 25,
"DED Output Data", "DED Display Line 4")

defineString("DED_LINE_5", function() return DEDLine5 end, 25,
"DED Output Data", "DED Display Line 5")
```

Appendix 1 – Format processing.

While not included in the release version the version of the code that enables the formatting is pretty simple. The basic concept is that when formatting needs to be applied to an output value character substitution is applied to the output to map the input character onto a character in the font which contains the character with the correct formatting.

The only formatting that is enabled at the moment is the inverse character but if the problems with sending the data through to Arduino can be resolved I'm hoping to include formatting for the double height character used on the T/ILS page when TACAN is turned off.

The Changed code to enable formatting is

```
local generalReplacements = {
      ["a"] = "@",
      ["o"] = "="
      }


local inverseFormatReplacements = {
      ["*"] = "×",
      [" "] = " ",
      ["0"] = "À",
      ["1"] = "Á",
      ["2"] = "Â",
      ["3"] = "Ã",
      ["4"] = "Ä",
      ["5"] = "Å",
      ["6"] = "Æ",
      ["7"] = "Ç",
      ["8"] = "È",
      ["9"] = "É",
      ["."] = "•",
      ["a"] = "@",
      ["o"] = "?",
    ["\'"] = "¬",
    [":"] = "¨"
```

```
}


------------------------------------------------------------
----DED Display Main Function---------------------------------
----------------------------------------
local function buildDEDLine(line)
-- Get Layout Information for line being built
      local DEDLayoutLine = DEDLayout[line]
-- Get Exported DED Objects
      local DED_fields = parse_indication(6)
      local layout
      local label
      local value
      local enableFormat = true


-- Base Output String
      local dataLine ="                          "


-- Check for present of Objects that indicate Duplicate Key
Names that need resolving
      local guard = DED_fields["Guard Label"]
      local mode = DED_fields["Mode label"]
      local event = DED_fields["Event Occured"]
      local alow =  DED_fields["ALOW label"]
      local bingo = DED_fields["CMDS_BINGO_lbl"]
      local inflt_algn = DED_fields["INS_INFLT_ALGN_lbl"]
      local intraflight = DED_fields["INTRAFLIGHT lbl"]
      local dlnk_A_G = DED_fields["A-G DL lbl"]
      local nav_status = DED_fields["NAV Status lbl"]
--Loop through Exported DED Objects
      for k,v in pairs(DED_fields) do
-- Handle Duplicate Key Names on COM2 Guard page items
            if guard ~= nil then
                  label = guard.." "..k
```

```
-- Handle Duplicate Key Names on IFF STAT page items
           elseif mode ~= nil then
                label = mode.." "..k
-- Handle Duplicate Key Names on IFF POS & TIM page items
           elseif event ~= nil then
                label = event.." "..k
-- Handle Duplicate Key Names on ALOW page Line 1 items
           elseif alow ~= nil and line == 1 then
                label = alow.." "..k
-- Handle Duplicate Key Names on CMDS Bingo page Line 1 items
           elseif bingo ~= nil and line == 1 then
                label = bingo.." "..k
-- Handle Duplicate Key Names on INS INFL ALGN page Lines 1 &
3 items
           elseif inflt_algn ~= nil and (line == 1 or line==3)
then
                label = inflt_algn.." "..k
-- Handle Duplicate Key Names on DLNK INTRAFLIGHT page
           elseif intraflight ~= nil then
                label = intraflight.." "..k
-- Handle Duplicate Key Names on DLNK A-G page Line 2 items
           elseif dlnk_A_G ~= nil and line == 2 then
                label = dlnk_A_G.." "..k
-- Handle Duplicate Key Names on NAV page
           elseif nav_status ~= nil then
                label = nav_status.." "..k
           else
                label = k
           end
--Get layout data associated with current key
           layout =
DEDLayoutLine[label:gsub("_inv","",1):gsub("_lhs","_both",1)]
           if layout ~= nil then
                local tempValue
```

```
--If layout value 6 is present then use this value to override
the value returned from DCS
                if layout[6] ~= nil then
                      tempValue = layout[6]
                else
                      tempValue = v
                end


--If layout value 5 is present then use this value to populate
the Format section of the output otherwise return ""
                if enableFormat and layout[5] ~= nil and
(layout[4] == "" or layout[4] == label:sub(#layout[4]*-1))
then
                      if layout[5] == "I" then
                            value =
tempValue:gsub(".",inverseFormatReplacements):lower()
                      end
                else
                      value =
tempValue:gsub(".",generalReplacements)
                end


-- Add Value to dataLine using mergeString because some values
are are supposed to fit within others
                dataLine = mergeString(dataLine, value,
layout[1])


--If layout value 3 > 0 we need to duplicate this item at
position specific in value 3 (this is for "*"s marking
enterable fields
                if layout[3] ~= nil and layout[3] > 0 then
                      dataLine = mergeString(dataLine, value,
layout[3])
                end
           end
     end
```

```
    return dataLine
end
```

The important sections are the generalReplacements and inverseFormatReplacements lists that detail the character swaps and the section that actually calls the swaps.

```
--If layout value 5 is present then use this value to populate
the Format section of the output otherwise return ""
if enableFormat and layout[5] ~= nil and (layout[4] == "" or
layout[4] == label:sub(#layout[4]*-1)) then
    if layout[5] == "I" then
        value =
        tempValue:gsub(".",inverseFormatReplacements):lower(
        )
    end
else
    value = tempValue:gsub(".",generalReplacements)
end
```

This is straightforward but there are a couple of notes – for inverse characters of A-Z we use lower case a-z which is wht we have the "`:lower`"

And the "`generalReplacements`" are there to do substitution of Up/Down arrow – "@" and Degree Symbon "o" even when no formatting is specified for a line.