

DCS World mod独立航电编程

Written By **CorsairCat**

Email to contactus@corsaircat.com if you have question

Specail Thanks to @Nero

这是一份开发独立舱内航电的文档(非官方)

这份东西不是翻译稿，全部是基于我对代码和mod的理解

当前版本0.3.0，暂时还没写完

遵循 CC BY-NC-SA 4.0 协议

前言

写这份东西是因为我发现中文版本的mod制作说明存在缺失,同时我对于国内的mod制作环境感到失望。这份东西算是一个给后来的DCS World mod开发者的小礼物。

在开始之前

首先请各位在看这份之前对于mod有基本的理解（会套官方的航电，比如FC3）。所以在这里我不会讨论包含建模和如何套壳，如何改SFM气动的话题。这里我主要会讲如何制作一个脱离官方模组的完全独立的航电系统（具体可以参考社区的A4-E模组）。

注意：很多SDK是不对非第三方开发的，作为mod开发者，只能使用DCS的Lua脚本引擎提供的很有限的功能。但就目前已知的基本上可以实现绝大多数航电控制。但是注意以下两个内容为完全无法控制:无线电通信和闭路光学观瞄

牢记以下内容

- 在开发中debug输出的最简单办法是用print_message_to_user("XXXXX")函数来输出状态，这个可以用来判断你的东西卡在哪里或者有没有执行到这一步
- 如果有出现奇怪的问题，请查看log文件去定位，语法问题导致的脚本错误会出现在log中

一、总览·加载逻辑

1. entry.lua

首先加载时候是走entry.lua文件,其中注意下面这行

```
make_flyable('YourPlaneName', current_mod_path..'Cockpit/Scripts/', FM, current_mod_path)
```

其中包含四个参数，第一个是飞机名称，第二个是座舱的加载文件夹，第三个是默认调用的飞行模拟模型，最后一个是基于通信模型

- `current_mod_path`代表目前的文件路径
- `..`是字符串拼接，比如'a'..'b'的结果是'ab'

2. device_init.lua

- 其中第二个参数所指的目录是座舱的默认加载目录
- 找到目录后会先自动执行device.init文件(暂时不讨论可点击的加载文件)
- `dofile` 代表加载一个附加的lua文件
- 一般是先计数所有的device，每个device后面有个目标的执行lua文件(一个device可以理解为一个模块)

a. MainPanel 仪表台

```
MainPanel = {"ccMainPanel", LockOn_Options.script_path.."mainpanel_init.lua"}
```

- 这一条代码代表执行创建一个主仪表：可以是一个或多个模型。

b. Creators 航电设备

```
creators = {}  
creators[devices.ELECTRIC_SYSTEM] = {"avSimpleElectricSystem", LockOn_Options.script_path..
```

- `creators`是一个表，用来描述所有的航电设备逻辑
- 在任意一个设备内，都要描述设备类型和设备程序脚本的位置

主要设备类型

- `avSimpleElectricSystem`, 这是基础的电力模块，内置函数与功能将在后续单独章节描述
- `avSimpleWeaponSystem`, 这是武器模块
- `avIntercom`, 这是无线电模块
- `avUHF_ARC_16`, 这是UHF无线电模块，与Intercom搭配可以启用与地勤沟通
- `avSimpleRadar`, 这个是基础雷达模块，后续在雷达章节详细描述
- `avLuaDevice`, 这是标准的设备

c. Indicators 显示屏设备

```
indicators = {}  
indicators[#indicators + 1] = {"ccIndicator", LockOn_Options.script_path.."MFD/Indicator"
```

- indicators 也是一张表，用来描述所有的显示屏。具体后续逻辑在显示器章节详细讲

3. mainpanel.lua

a. 座舱内模型

```
shape_name          = "Cockpit-EDM-File-Name"  
is_EDM              = true  
new_model_format    = true
```

- 这里可以定义一个仪表台的模型

b. 座舱内动画声明

```
Canopy              = CreateGauge()  
Canopy.arg_number    = 1  
Canopy.input         = {0,1}  
Canopy.output        = {0,1}  
Canopy.parameter_name = "Canopy_Anima"
```

- 类似于上面这样的是定义舱内动画,CreateGauge()代表创建一个动画

4. clickable.lua 可点击描述

- 注意，需要虚拟体（3dsmax Helper）支持
- 具体函数见样例说明

二、总览·动画渲染

注意，所有动画都需要arg based animation 支持，此处不做详细论述
特别注意，由于内外有两份座舱，外部模型中的arg114动画为使外部舱内消失

1. 舱外动画

- 首先建议舱外采用标准化的arg数值，以便AI操作飞机和SFM的表现正常

a. 关于在自动驾驶时如何反馈到外部动画以及获取外部动画位置

- 建议采用内置函数
- 绘制外部：

```
set_aircraft_draw_argument_value(9, ROLL_STATE)
```

- 读取外部：

```
get_aircraft_draw_argument_value(9)
```

第一个参数为动画arg号，第二个值为设置到的状态

2. 舱内动画（不含屏幕）

- 舱内动画需要在mainpanel.lua注册，具体见加载顺序部分

a. 关于舱内动画注册

```
Canopy                = CreateGauge("parameter")
Canopy.arg_number      = 1
Canopy.input           = {0,1}
Canopy.output          = {0,1}
Canopy.parameter_name  = "Canopy"
```

- CreateGauge("parameter")为注册对象（一般推荐用这个来进行完全控制，避免采用默认的值，比较难以控制）
- .arg_number 为注册其动画对应的arg
- .input和.output为输入输出对应值，即：set(value)函数 到 最终绘制输出的值
- .parameter_name为用来进行控制的索引，注意这里的命名，避免冲突以及不要和系统内置冲突

b. 如何绘制动画

```
a = get_param_handle("LandingGearLevel")
```

- 这里采用a作为样例，首先需要用get_param_handle(target_param_name)函数来获取这个对象
- 后可采用:set()来设置其的值或:get()来获取其的值

```
a:set(value)
a:get()
```

c. 注册中的input和output

- 舱内动画的绘制并不是直接进行设置动画的，而是你对param对象的set()值用input到output值进行对应关系

- 例如.input={-1,1},.output={0,1}时, set(-1)最后反应到动画绘制的结果为0位置

三、总览·获取飞行器外部参数

- 核心函数为get_base_data()这样是申请一张表,详细见[sensorData.md](#)
- 获取GPS坐标数据: local self_loc_x , own_alt, self_loc_y = Data_Raw.getSelfCoordinates()

四、航电设备·关于avLuaDevice

1. 这里从一个标准device的脚本开始

```
-- 获取脚本本身设备
local Device = GetSelf()

-- 刷新函数间隔设置
local update_time_step = 0.02
make_default_activity(update_time_step)

function post_initialize()
    --默认初始化函数
end

Device:listen_command(KeyCommand)

function SetCommand(command,value)
    --对listen_command的处理函数
end

function update()
    --内置刷新函数
end
```

- LuaDevice本身只有listenCommand函数

2. 关于初始化的问题

- 初始化中(post_initialise)可以设置在不同出生情况下的设置

```
local birth = LockOn_Options.init_conditions.birth_place
if birth=="GROUND_HOT" then
    --地面热车
elseif birth=="AIR_HOT" then
    --空中出生
elseif birth=="GROUND_COLD" then
    --地面冷舱
end
```

五、航电设备·关于avSimpleElectricSystem

1. 概述

- 这个设备是游戏内置的基础电力模块
- 当你希望使用avSimpleRadar,avSimpleWeaponSystem等等时，这个设备必须存在并且任意电力总线处于激活状态

2. 设备内置函数

注意，这部分内容来源于一个较早的mod样例

- 首先，调用设备对应的内置函数需要采用如下方式

```
local Device = GetSelf()
Device:Device_Function()
```

a. 电力系统内置初始化函数

- 这些函数是初始化，请尽量放在脚本靠近开头位置

```
左发发电机(单引擎请只启用这个)
AC_Generator_1_on(true/false)
```

```
右发发电机
AC_Generator_2_on(true/false)
```

```
电池组(推荐启用)
DC_Battery_on(true/false)
```

b. 电力系统内置监测函数

交流总线

get_AC_Bus_1_voltage() -- 左发运转正常且启用左发电机时返回115

get_AC_Bus_2_voltage() -- 右发运转正常且启用右发电机时返回115

直流总线

get_DC_Bus_1_voltage() -- 左发总线有115或电池开关打开时返回28

get_DC_Bus_2_voltage() -- 右发总线有115或电池开关打开时返回28

c. 注意事项

- 直流总线的1和2是和交流总线的1和2绑定的
- 电池开关是默认的iCommand指令直接传递给游戏核心的
- 这是一些和电力相关的默认icommand数值

指令名称	指令编号
iCommandPowerOnOff	315
iCommandGroundPowerDC	704
iCommandGroundPowerDC_Cover	705
iCommandPowerBattery1	706
iCommandPowerBattery1_Cover	707
iCommandPowerBattery2	708
iCommandPowerBattery2_Cover	709
iCommandGroundPowerAC	710
iCommandPowerGeneratorLeft	711
iCommandPowerGeneratorRight	712
iCommandElectricalPowerInverter	713
iCommandAPUGeneratorPower	1071
iCommandBatteryPower	1073
iCommandElectricalPowerInverterSTBY	1074
iCommandElectricalPowerInverterOFF	1075
iCommandElectricalPowerInverterTEST	1076

d. 关于iCommand指令

- 如果不希望直接由输入直接发指令给游戏核心，可以更改input为自定义的指令值
- 然后采用

```
dispatch_action(nil, iCommandNum, value)
```

来发送原生指令

六、航电设备·关于avSimpleWeaponSystem

1. 概述

- 这个系统是航电中的武器控制模块，其内置很多与游戏武器API交互的函数
- 需要电源系统接通

2. 内置函数

- 注：函数调用和电力模块一致需要获取getSelf()表

函数	作用
drop_chaff()	抛洒箔条
drop_flare()	抛洒红外干扰弹
emergency_jettison()	紧急抛离(效果未验证)
emergency_jettison_rack()	紧急抛离挂架(效果未验证)
get_chaff_count()	获取箔条数量
get_ECM_status()	获取ECM干扰器状态(未实验)
get_flare_count()	获取干扰弹数量
get_station_info()	获取挂架信息(返回为一张表)
get_target_range()	获取目标距离
get_target_span()	获取目标宽度
get_weapon_count()	获取挂架上武器数量(部分武器为单挂架多个)

函数	作用
launch_station()	发射武器(需要先选中挂架)
select_station()	选中挂架()
set_ECM_status()	设置ECM干扰器状态
set_target_range()	设置目标距离
set_target_span()	设置目标宽度

3. 关于武器发射

- 武器发射不管是否锁定，都可以发射（是否发射可以做判断控制）
- 武器发射前需要选中挂架
- 雷达弹锁定将在雷达系统部分详细说
- 红外弹会在这里说
- 抛离也需要挂架号

4. 获取挂载信息

- 可以使用该函数获取挂架上挂载的表：station = WeaponSystem:get_station_info(pylonSelected-1)
- 挂架号从0开始
- 判断挂载是什么的简单方法是查询挂载表的CLSID(station.CLSID)和游戏武器数据库中的CLSID（在某个目录，稍后补充）
- station.count可以判断挂架上有没有东西（==0就是必然没东西）
- 对地攻击的精确制导武器目前不清楚

5. 关于红外导弹

- 红外导弹用的制导控制是parameter（和动画一样的操作逻辑）

```
-- 获取锁定状态
-- 是否锁定
local ir_missile_lock_status = get_param_handle("WS_IR_MISSILE_LOCK")
-- 锁定的目标的高程和方向
local ir_missile_target_az = get_param_handle("WS_IR_MISSILE_TARGET_AZIMUTH")
local ir_missile_target_el = get_param_handle("WS_IR_MISSILE_TARGET_ELEVATION")

-- 设置搜索方向
-- 方位角
local ir_missile_desire_az = get_param_handle("WS_IR_MISSILE_SEEKER_DESIRED_AZIMUTH")
-- 高程
local ir_missile_desire_el = get_param_handle("WS_IR_MISSILE_SEEKER_DESIRED_ELEVATION")
```

附录