DCS-BIOS

Release v0.10.0

Nov 12, 2019

Contents:

If you are a new user, start with the next chapter: *Installing DCS-BIOS.*

If you already know what you are looking for, use the search bar below the logo image on the left or consult the table of contents below.

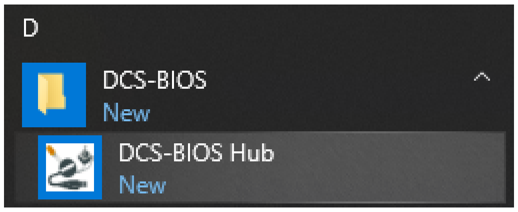
CHAPTER 1

Installing DCS-BIOS

Download and run the installer (called DCS-BIOS-Hub-Setup-*version*.msi) for the latest stable release from GitHub.

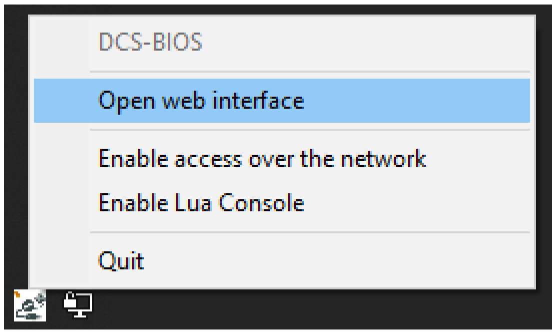
Note: If you want a bleeding edge version or are looking for a previous release, take a look at the complete list of releases instead.

After the installation is complete, start the DCS-BIOS Hub using the shortcut in your start menu. The DCS-BIOS Hub is a program that exchanges cockpit data and commands between DCS: World, custom-built panels, and third-party software.



After clicking the start menu shortcut, you will find a new icon in your system tray. If you just installed DCS-BIOS, the icon will probably be hidden; you will have to click the arrow to show hidden icons. If you want Windows to always display the icon, you can drag it onto the taskbar.

Left click the icon and select the "Open web interface" menu item:

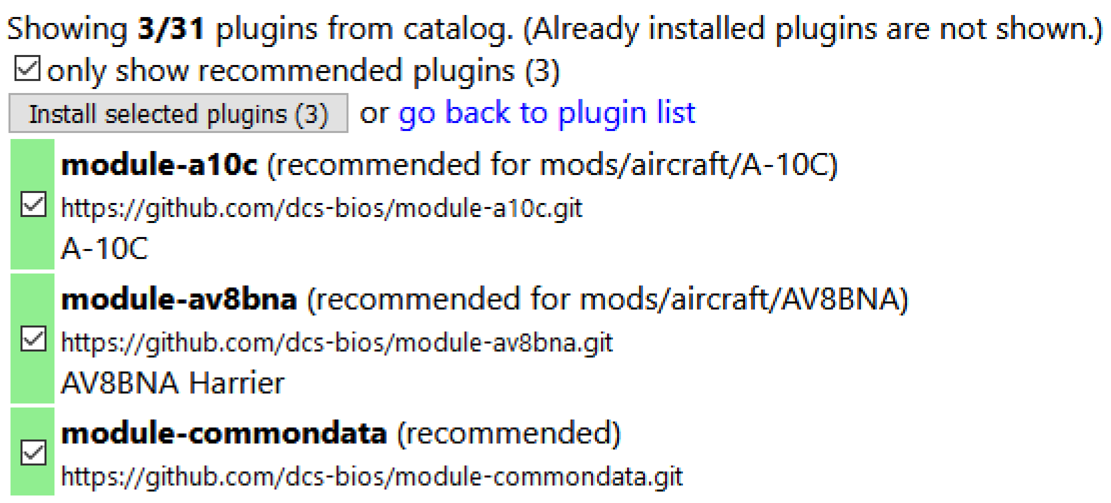


You should now be looking at a web browser showing the DCS-BIOS Hub web interface. If that is not the case, verify that your firewall is not blocking the DCS-BIOS Hub from listening on TCP port 5010.

1.1 Installing Plugins

The DCS-BIOS Hub only knows how to get the name of the currently active aircraft from DCS, which is not very useful. To teach it to interact with a specific aircraft in DCS, you need to install a module definition plugin.

Click "Plugins" in the navigation menu on the left, then click "Open the plugin catalog". You will see a list of recommended plugins for the DCS aircraft modules that DCS-BIOS has found on your hard drive:



Showing **3/31** plugins from catalog. (Already installed plugins are not shown.)

☑ only show recommended plugins (3)

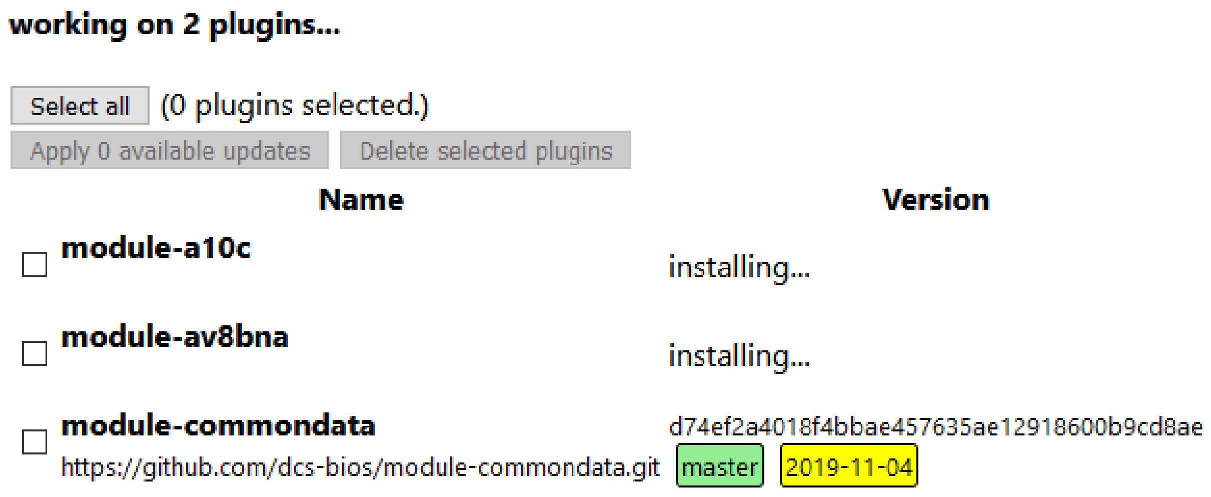
☑ **module-a10c** (recommended for mods/aircraft/A-10C)
https://github.com/dcs-bios/module-a10c.git
A-10C

☑ **module-av8bna** (recommended for mods/aircraft/AV8BNA)
https://github.com/dcs-bios/module-av8bna.git
AV8BNA Harrier

☑ **module-commondata** (recommended)
https://github.com/dcs-bios/module-commondata.git

Fig. 1: List of recommended plugins

Click the "Install selected plugins" button.



working on 2 plugins...

(0 plugins selected.)

Name	Version
☐ **module-a10c**	installing...
☐ **module-av8bna**	installing...
☐ **module-commondata** https://github.com/dcs-bios/module-commondata.git	d74ef2a4018f4bbae457635ae12918600b9cd8ae master 2019-11-04

Fig. 2: Plugin installation in progress

Once a plugin has finished, its version will be shown. In the screenshot above, the “module-commondata” plugin has already finished installing, while the plugins for the A-10C and the Harrier are still being downloaded.

If the installation takes a while, you can just continue with the next step. The installation will continue in the background and you can monitor the progress by opening the “Plugins” page again at a later time.

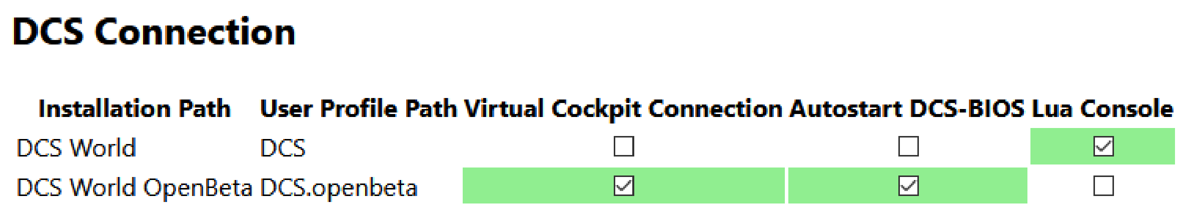
Continue with the next section: *Connecting to DCS*.

CHAPTER 2

Connecting to DCS

Before DCS-BIOS can communicate with DCS: World, you need to install some Lua scripts that will be picked up by DCS: World and will send data to and receive commands from DCS-BIOS.

You can do this on the "DCS Connection" screen in the web interface. It will look for the release and open beta versions of DCS: World and provide checkboxes to install the scripts:



DCS Connection

Installation Path	User Profile Path	Virtual Cockpit Connection	Autostart DCS-BIOS	Lua Console
DCS World	DCS	☐	☐	☑
DCS World OpenBeta	DCS.openbeta	☑	☑	☐

Most users will want to check "Virtual Cockpit Connection" and "Autostart DCS-BIOS" for all DCS: World installations, and leave "Lua Console" unchecked. In the image above, for illustration purposes the Lua Console is enabled for the release version, while the Virtual Cockpit Connection and the autostart feature are enabled for the open beta.

- The **Virtual Cockpit Connection** feature opens a communication channel between DCS: World and the DCS-BIOS Hub. It sends information about the current state of switches, indicators and displays in the virtual cockpit to the DCS-BIOS Hub and allows the DCS-BIOS Hub to trigger actions such as setting a dial to a specific position or flipping a toggle switch.
- The **Autostart DCS-BIOS** setting will automatically start the DCS-BIOS Hub in the background after you have started DCS: World.
- The **Lua Console** is an optional feature for users who write Lua scripts for DCS: World or make missions with custom triggers. You can learn more about it in the *Lua Console* section. Leave it disabled if you do not need it.

In the next section, we will take a closer look at the *Dashboard* screen.

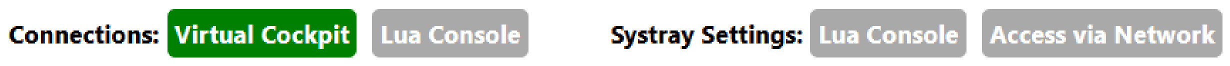
CHAPTER 3

The Dashboard

The Dashboard is the first screen you see when you open the web interface.

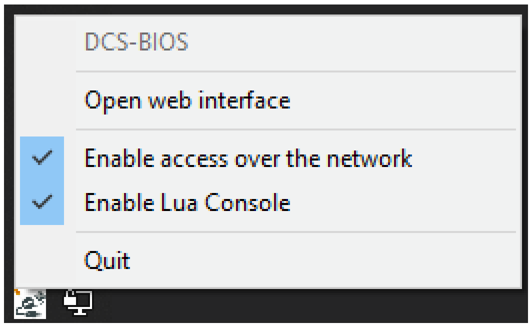
3.1 Status Indicators

At the top, you can find various status indicators:



- **Virtual Cockpit** indicates that the DCS-BIOS Hub is exchanging cockpit data and commands with DCS: World.
- The **Lua Console** indicator shows that DCS: World is ready to accept arbitrary snippets of Lua code from the DCS-BIOS Hub.

The other two indicators show the current state of two settings that you can toggle from the system tray menu:

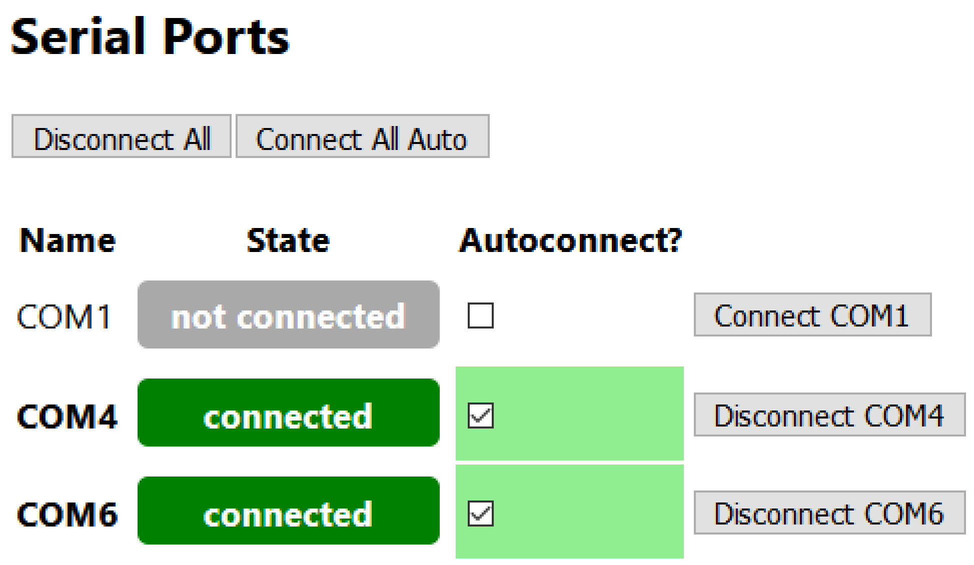


- When **Enable access over the network** is checked, the web interface is accessible from other computers on the network. This can be helpful if you need to use the web interface while flying and it's more convenient to use an external device rather than switching between DCS: World and a web browser on the same machine.
- When **Enable Lua Console** is checked and the Lua Console has been set up on the *DCS Connection* screen, you can use the web interface to execute arbitrary snippets of Lua code within DCS.

Warning: Note that if the Lua Console is enabled, anyone who can access TCP port 5010 on your computer can run arbitrary code on your machine. If you enable "access over the network" as well, this includes anyone who can directly connect to your computer over the network. If you do this, make sure your computer is not directly reachable from the public internet.

3.2 Managing Serial Port Connections

The Dashboard screen displays a list of serial ports and allows you to configure which of these you want the DCS-BIOS Hub to connect to. If you build custom control panels, each Arduino board you connect to your PC will show up as a (virtual) COM port here. You might also see some real RS-232 ports listed, if your main board still has any. In the following screenshot, COM1 is a real RS-232 port and is not being used, while COM4 and COM6 belong to Arduino boards that are connected via USB.



Serial Ports

Disconnect All | Connect All Auto

Name	State	Autoconnect?	
COM1	not connected	☐	Connect COM1
COM4	connected	☑	Disconnect COM4
COM6	connected	☑	Disconnect COM6

The "Autoconnect" checkbox tells the DCS-BIOS Hub that it should connect to this port when the DCS-BIOS Hub is started or when the COM port "appears", i.e. the device is plugged in and the port shows up as a new device. If you unplug a device, its COM port will disappear from the list if autoconnect was not checked. If autoconnect was enabled, the port will be listed as "missing" instead and the connection will be reestablished as soon as it appears again.

The individual "Connect" and "Disconnect" buttons on the right can be used to temporarily connect or disconnect a port without changing the autoconnection setting.

The "Disconnect All" button disconnects from all COM ports.

The "Connect All Auto" button connects to all COM ports that have autoconnect enabled.

3.3 Managing Hub Scripts

At the end of the Dashboard, you can manage a list of "hub scripts". Hub scripts can be used to remap commands and exported data, so you can use a simpit you built for one specific airframe with other DCS: World modules. Learn more in the *Hub Scripts* section.

- Add a hub script by clicking the "Add" button and entering the full path to the script file. You can copy the full path of a file to your clipboard by holding the Shift key while right-clicking it in Windows Explorer and then selecting “Copy as path” from the context menu.

- To remove a hub script from the list, click the “x” button next to it.
- Enable or disable a hub script with the checkbox in front of it. Disabled scripts remain in the list but are not loaded.
- Reload all hub scripts by clicking the “Reload Scripts” button above the list. Note that scripts are not automatically reloaded when you enable, disable, add or remove a list item; you have to click the “Reload Scripts” button for changes to take effect.

CHAPTER 4

The Control Reference

“Module definitions” tell DCS-BIOS which input and output elements exist in an aircraft, how to determine their current state, and how to operate them in the virtual cockpit using Lua scripts in DCS: World. The Control Reference lets you look at everything that is defined in a module definition. In other words, it shows you every dial, toggle switch, push button, indicator light and display that DCS-BIOS knows about.

Note: Module definitions are installed using the plugin manager (see *Installing DCS-BIOS*). The exception are the built-in “MetadataStart” and “MetadataEnd” modules, which provide the name of the currently active aircraft and a counter that is increased after every frame.

On the first screen, you can select a module definition, which usually corresponds with an aircraft model.

Once you click on a module definition, you can either select a category of “controls” to browse, or search for a specific control by name or description using the search field.

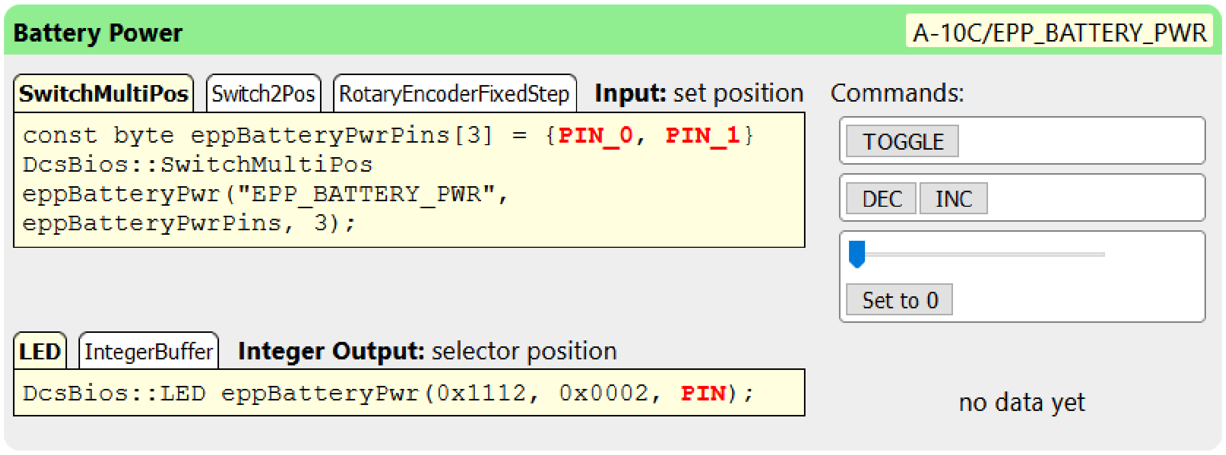
Note: Tip: each module definition page in the Control Reference has a unique URL, so you can create a bookmark in your browser for the reference page of your favourite aircraft.

4.1 Controls, Inputs and Outputs

An “input element” or “input” is something that accepts commands, like a toggle switch or rotary knob. An “output element” or “output” is everything that produces a piece of data made available to DCS-BIOS. A “control” is a thing in the cockpit that has a name and can act as an input, output, or both.

Let’s take a look at the Battery Power switch in the A-10C.

- Open the web interface, navigate to the Control Reference and select the A-10C.
- Click the “Electrical Power Panel” category and scroll down to the entry for “Battery Power”. You can type a few characters into the “Filter…” text box to help you find the category link. You can also try to use the search function to jump directly to the “Battery Power” entry.



```
const byte eppBatteryPwrPins[3] = {PIN_0, PIN_1}
DcsBios::SwitchMultiPos
eppBatteryPwr("EPP_BATTERY_PWR",
eppBatteryPwrPins, 3);
```

```
DcsBios::LED eppBatteryPwr(0x1112, 0x0002, PIN);
```

You are now looking at a "control", the battery power switch on the A-10C. It is identified by a short name ("EPP_BATTERY_POWER"). Since a different module definition could have its own EPP_BATTERY_POWER control, control identifiers can be prefixed with the module definition and a slash to make them unique. The full control identifier is shown in the top right corner.

The black text on the green bar is the description of the control, in this case "Battery Power".

Below the description and control identifier, we see that this control accepts input commands and has one integer (number) output.

The left side shows some code snippets. In the next section, you will learn how to copy and paste these into a program for an Arduino microcontroller to connect real hardware to DCS: World.

On the right side next to the input code snippets, we see buttons for each command we can send to this control. If DCS: World is running and you enter the cockpit of an A-10C, these buttons will operate the switch in your virtual cockpit. You can use these buttons to verify that the connection to DCS is working correctly and that DCS-BIOS is operating the control in the virtual cockpit correctly.

Next to the output code examples, we see the current value that is being exported. It shows "no data yet" in the screenshot above because DCS: World is not running, so there is no virtual A-10C cockpit to connect to. Looking at live data while DCS is running can be very useful to figure out what range of values to expect from a certain output under different flight conditions.

4.2 Output Types

An output is either a number (called an "integer output") or a piece of text (called a "string output"). Most controls have either an integer output or a string output, while some controls have no output at all. A few controls will have both an integer and a string output.

4.3 Input Interfaces

Note: If this part sounds confusing, don't worry about it and just continue with the next sections. It is not necessary to fully understand the concept of input interfaces to build custom hardware.

A control in DCS-BIOS can support one or more "input interfaces". An "input interface" is a set of commands that a particular control can understand.

The **set_state** input interface is the most common. It means that DCS-BIOS knows how to set a control like a toggle switch to a specific position, independently of its current position. In the control reference, it is represented with a slider and a button. Use the slider to select the value you want, then click the button to send that value to DCS:

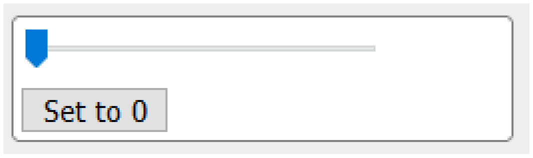


Fig. 1: set_state

The **fixed_step** input interface understands the "DEC" and "INC" commands, which decrease or increase the current position of a control by one. A typical example is a dial that lets you change radio presets. The buttons to operate a fixed_step control look like this:

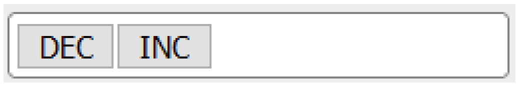


Fig. 2: fixed_step

The **variable_step** input interface is similar to fixed_step, but the "decrement" and "increment" commands have a magnitude from 0 to 65535. A typical example is a volume knob. In the control reference. the variable_step commands are represented by a slider which represents the amount of change you want, and two buttons that let you send a decrement or increment command of the selected magnitude:

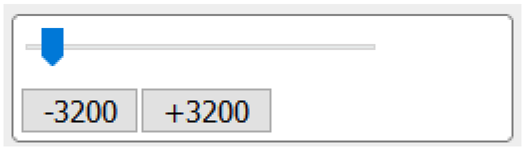


Fig. 3: variable_step

The **action** input interface is an arbitrary action. It is represented in the control reference by a button labelled with the name of the action. Any two-position switch supports the "TOGGLE" action. It is rarely used, but can come in handy in some special cases, for example if you want to bind the built-in push button of a physical rotary encoder to toggle the "mute" function of a volume knob.

Fig. 4: toggle

CHAPTER 5

Building Hardware Panels

With DCS-BIOS, you can build your own control panels and connect them to DCS: World. In general, the process looks like this:

- Order whatever buttons, switches, rotary encoders, displays and LEDs you need for your project. AliExpress and eBay are popular sources worldwide; of course, you can also order from one of the big electronics distributors like DigiKey, Mouser, and Newark/element 14.
- Find a way to mount all of those to a front plate. Depending on what tools and skills you have available, this can range from cardboard, a felt-tip pen and hot glue to backlit panels cut and engraved on CNC machines that are almost indistinguishable from the real thing at first sight.
- Electrically connect your components to an Arduino microcontroller board and program that microcontroller board to talk to DCS-BIOS. This is the step that will be described in this section of the manual.

5.1 Choosing an Arduino board

Arduino boards are small PCBs built around a microcontroller, which you can program with the Arduino IDE (Integrated Development Environment). You can learn more about the Arduino project at https://www.arduino.cc.

There are many different Arduino boards available. DCS-BIOS supports boards that feature the ATMega328 or the ATMega2560 microcontrollers. That includes the Arduino Uno, Nano, Pro Mini and Mega 2560 boards.

We recommend the Arduino Nano. It has a small form factor but includes the USB port needed to program it, unlike its smaller sibling the Pro Mini.

The Arduino platform is “Open Hardware”, which means anyone is allowed to make and sell these boards without paying a royalty fee. This has led to the availablility of very inexpensive “clones” on eBay and AliExpress. While some of these use a different chip (CH340) for the USB communications, they all work perfectly fine for use with DCS-BIOS.

5.2 Wiring up your controls

To connect your electromechanical components to the Arduino board, you will use some combination of soldering jumper wires, solderless breadboards, and/or custom printed circuit boards.

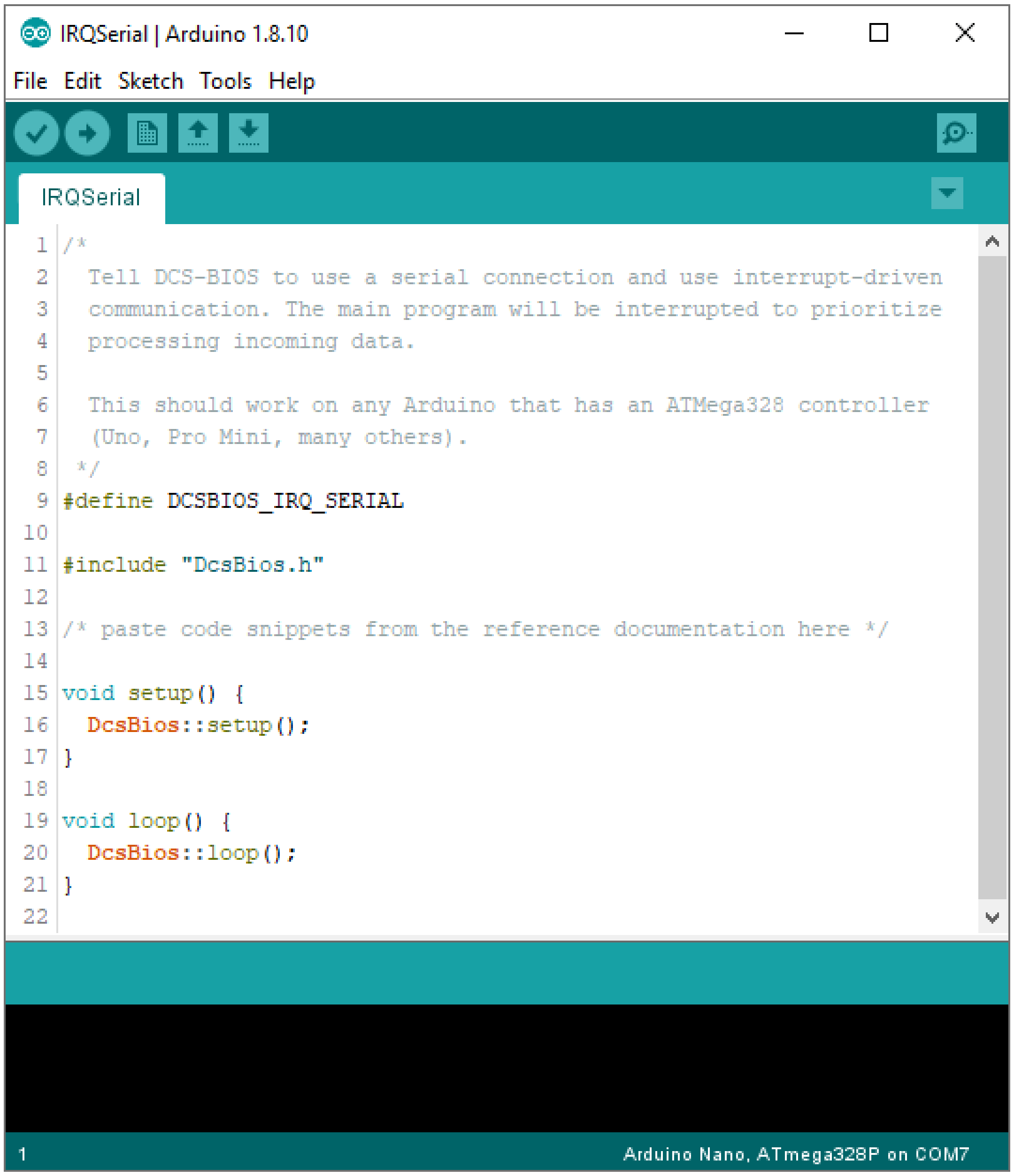
Each Arduino board has a number of digital I/O pins (on the Arduino Nano, these are labelled D0 to D13) and a number of analog I/O pins (A0 to A7 on the Arduino Nano). Digital pins can be used as a digital input (read a high or low voltage) and as a digital output (be turned on or off by the program). Analog inputs can also be used to read an analog voltage level, which can be used to connect a potentiometer (e.g. to control a volume knob). With the exception of pins A6 and A7 on the Arduino Nano and Pro Mini, any analog pin on the controllers mentioned above can also be used as a digital pin.

Digital pins 0 and 1 are used to communicate with the computer, so you cannot use them for your panels. Most Arduino boards have a built-in LED connected to pin 13, so that pin should only be used as an output.

5.3 Programming your Arduino board

To program your Arduino board, you will need the Arduino IDE, which you can download for free on the official Arduino website..

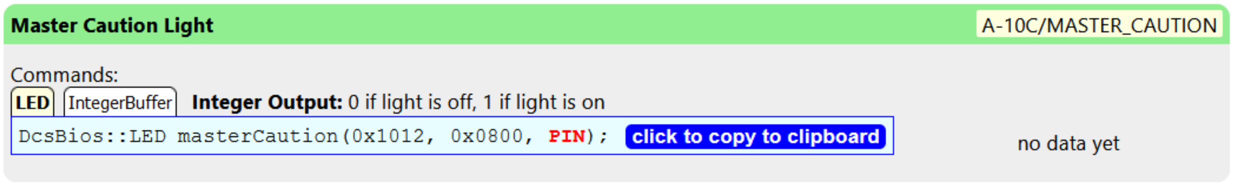
You will also need the DCS-BIOS Arduino library, which you can download here. After downloading the library, open the Arduino IDE, select "Sketch" > "Include Library" > "Add .ZIP Library…" from the menu and choose the .ZIP file you downloaded. You should now find a "DCS-BIOS" submenu under "File" > "Examples". Start your Arduino program by opening the "IRQ Serial" example from that menu.



```
/*
  Tell DCS-BIOS to use a serial connection and use interrupt-driven
  communication. The main program will be interrupted to prioritize
  processing incoming data.

  This should work on any Arduino that has an ATMega328 controller
  (Uno, Pro Mini, many others).
 */
#define DCSBIOS_IRQ_SERIAL

#include "DcsBios.h"

/* paste code snippets from the reference documentation here */

void setup() {
  DcsBios::setup();
}

void loop() {
  DcsBios::loop();
}

```

The example program already includes the DCS-BIOS Arduino Library, which will take care of communicating with the DCS-BIOS Hub. However, you still need to tell it about what is connected to which pins on your Arduino board.

Let's connect the built-in LED on pin D13 to an indicator light in the cockpit. This manual section will use the Master Caution warning light in the A-10C as an example, but you can substitute any indicator light in your aircraft of choice. Find your indicator light in the *control reference* and copy the code snippet that begins with "DcsBios::LED" to the clipboard by clicking on it.

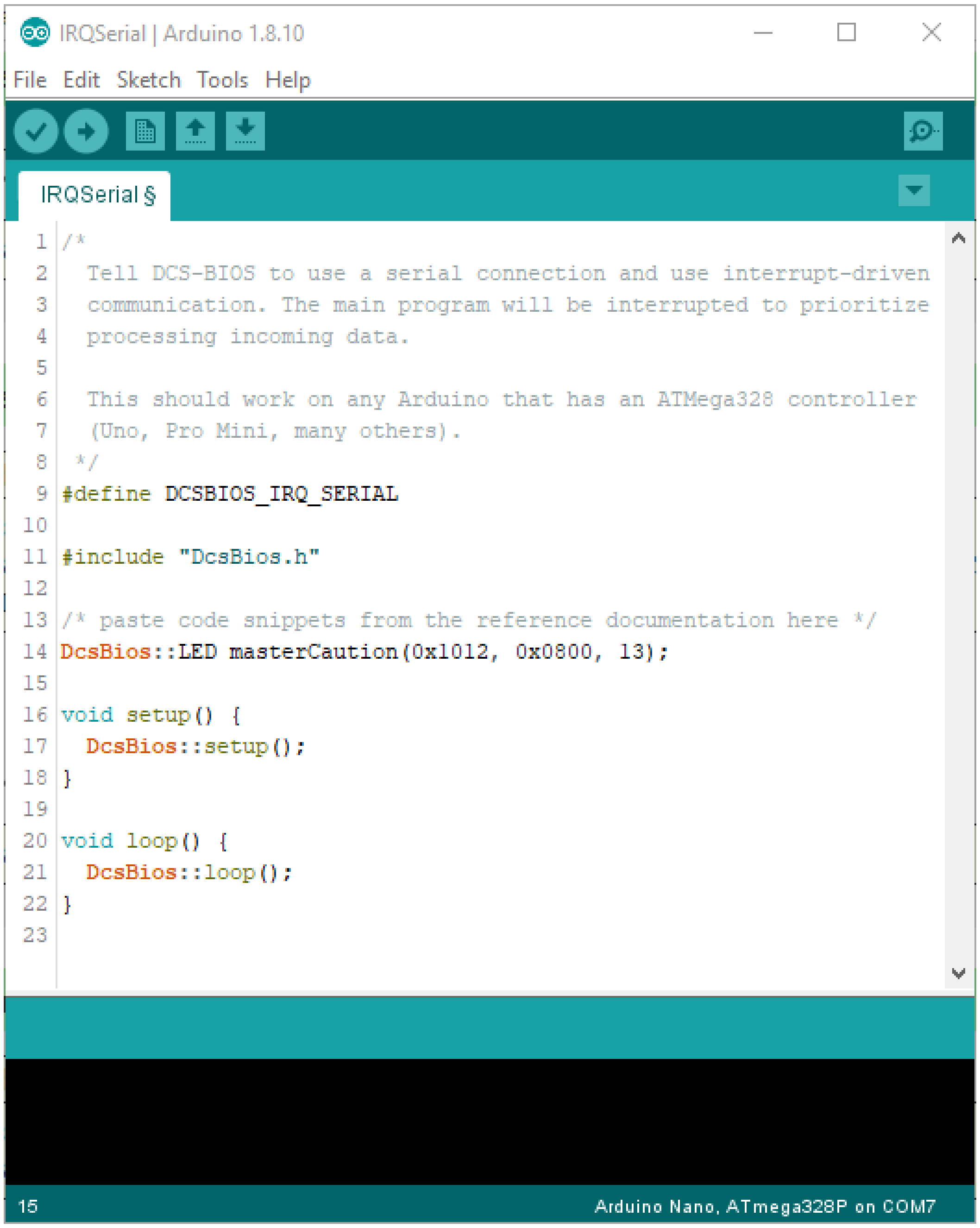

Master Caution Light A-10C/MASTER_CAUTION

Commands:

LED IntegerBuffer **Integer Output:** 0 if light is off, 1 if light is on

```
DcsBios::LED masterCaution(0x1012, 0x0800, PIN);
```

Paste it into your Arduino program below the comment in line 13. Replace the text PIN at the end with the pin number the light is connected to, in this case 13 (we are using the built-in LED). When using a digital pin, you just use the pin number; analog pins are designated as A0, A1, etc.



```
 1 /*
 2   Tell DCS-BIOS to use a serial connection and use interrupt-driven
 3   communication. The main program will be interrupted to prioritize
 4   processing incoming data.
 5
 6   This should work on any Arduino that has an ATMega328 controller
 7   (Uno, Pro Mini, many others).
 8  */
 9 #define DCSBIOS_IRQ_SERIAL
10
11 #include "DcsBios.h"
12
13 /* paste code snippets from the reference documentation here */
14 DcsBios::LED masterCaution(0x1012, 0x0800, 13);
15
16 void setup() {
17   DcsBios::setup();
18 }
19
20 void loop() {
21   DcsBios::loop();
22 }
23
```

```
15                                   Arduino Nano, ATmega328P on COM7
```

Press the checkbox in the toolbar to check your program for errors. If you get an orange bar and an error message at the bottom of the window, double-check that you copied the code snippet to the correct place, replaced the text “PIN” with “13”, and left the rest of the program intact. The compiler (the program that translates your program text to something the Arduino board can understand) can react to a single missing character with a long, incomprehensible error message.

To prepare to upload your program to the Arduino board, perform the following steps:

- Connect your Arduino board to your computer using a USB cable

- Select your board type from the “Tools” > “Board” menu.
- Select the COM port your board is connected to from the “Tools” > “Port” menu. If you cannot tell which port number belongs to your board, watch the Dashboard screen in the DCS-BIOS Hub web interface while you plug in your board.
- Make sure the DCS-BIOS Hub is not currently connected to your board’s COM port. If the port shows up as “connected” in the Dashboard screen, click the disconnect button next to it.

Now you can click the “right arrow” symbol in the Arduino IDE’s toolbar or select the “Sketch” > “Upload” menu item to transfer your program to the Arduino board.

After the progress messages have scrolled by and you see “Upload complete” at the bottom of the Arduino IDE window, connect the DCS-BIOS Hub to the COM port via the Dashboard screen in the web interface. Start DCS and make your indicator light turn on or off by messing with the aircraft controls (in our A-10C example, you could press the “Signal Lights Test Button”). The LED on your Arduino board should mimic the indicator in your virtual cockpit.

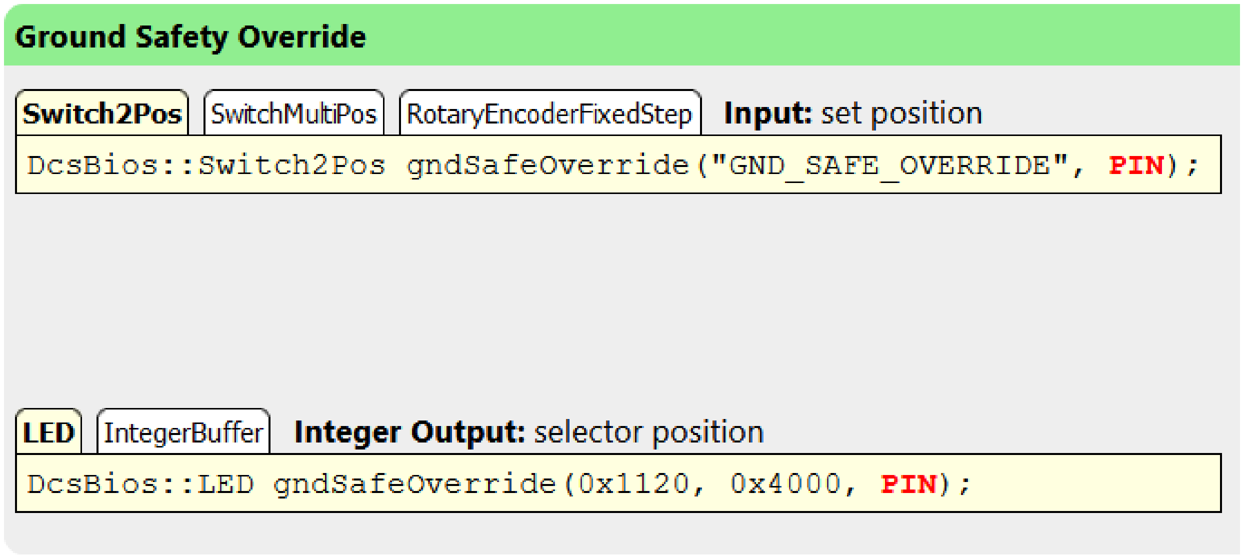
The next section will explain each type of code snippet in detail and also explain the circuitry that goes along with it.

CHAPTER 6

Code Snippet Reference

The Control Reference offers different types of code snippets meant to connect different types of electromechanical components. You can switch between different types of code snippets using the little tabs above the snippet.

Take a look at the reference for the reverse gear in the A-10C:



```
DcsBios::Switch2Pos gndSafeOverride("GND_SAFE_OVERRIDE", PIN);
```

```
DcsBios::LED gndSafeOverride(0x1120, 0x4000, PIN);
```

For the input, you have a choice between "Switch2Pos", "SwitchMultiPos" and "RotaryEncoderFixedStep" code snippets. The output for the switch position offers a "LED" and an "IntegerBuffer" code snippet.

The control reference tries to present the most relevant code snippet first. It is correct most of the time, but you should still read this section and know which code snippet to use in which situation. For example, the control reference prefers a ServoOutput code snippet for analog instruments, but you will often want the IntegerBuffer snippet instead to use custom code to drive a stepper motor or display the value on a small OLED display instead.

The rest of this section will explain when to use each type of code snippet and what circuitry goes along with it.

6.1 On logic levels and pull-up resistors

In digital logic, the voltage level on an I/O pin can either be LOW (connected to ground, at 0V) or HIGH (connected to supply voltage, which is 5V on most Arduino boards, but can also be 3.3V).

The software on the Arduino can also choose to connect each I/O pin to the supply voltage over a built-in "pull-up resistor". This ensures that the pin will always read logic HIGH when nothing is connected to it (instead of switching around randomly whenever a radio wave passes by). The DCS-BIOS Arduino Library makes extensive use of these internal pull-up resistors to simplify the ciruits required. Any switches are wired up so they connect the Arduino pin to ground when they are closed. The direct ground connection to the switch is "stronger" than the connection to the supply voltage through the pull-up resistor, so the pin will read logic LOW when the switch is closed.

6.2 Momentary vs latched switches

For the purposes of the Arduino code, there is no difference between a momentary switch like a push button or spring-loaded toggle switch and a "latching" switch like a normal toggle switch. The only difference is that one is spring-loaded to return to a certain position and the other is not. Electrically, they are the same, so they use the same code snippets.

6.3 Switch2Pos

The *Switch2Pos* snippet is one of the simplest and most common snippets you will use. It is used for switches that have exactly two positions: push buttons and two-position toggle switches (on-off switches).

Connect one contact of your switch to ground and the other one to the Arduino pin you specify in the code snippet.

- With the switch open, the Arduino pin will be HIGH and the switch will be set to position 0 in DCS: World.
- When the switch is closed, the Arduino pin will be LOW and the switch will be set to position 1 in DCS: World.

If you notice that you mounted the switch upside down, you can add ", true" after the pin number like this to reverse the behavior of the Switch2Pos code snippet:

```
DcsBios::LED gndSafeOverride(0x1120, 0x4000, 3, true);
```

The same trick works if you want to connect a normally-closed push button.

6.4 Switch3Pos

The Switch3Pos snippet is used for three-position toggle switches (on-off-on switches).

An on-off-on switch has three terminals. Connect the center terminal to ground and the other two terminals to the Arduino pins named PIN_A and PIN_B in the code snippet.

- When the switch is in the DOWN or LEFT position, PIN_A will be connected to ground through the center terminal on the switch, the Arduino will read PIN_A as LOW and set the switch to position 0 in DCS: World.
- When the switch is in the CENTER position, none of the pins will be connected to the center terminal, the Arduino will read PIN_A and PIN_B as HIGH and set the switch to position 1 in DCS: World.
- When the switch in in the UP or RIGHT position, PIN_B will be connected to ground through the center terminal, the Arduino will read PIN_B as LOW and set the switch to position 2 in DCS: World.

6.5 SwitchMultiPos

The SwitchMultiPos code snippet is meant for multi-position selector switches with more than three positions.

A rotary switch has one center terminal and one terminal for each switch position. Connect the center terminal to ground and each switch position terminal to an Arduino pin.

When the Arduino sees that an I/O pin is LOW, it will send the corresponding switch position to DCS: World. When no pin is LOW, it will assume the switch position has not changed.

6.6 RotaryEncoderFixedStep

Rotary encoders are dials that can be turned infinitely in both directions. They have three terminals: A, GND and B. If there are three terminals in a row, A and B will be the outer ones and GND will be the middle one. If your rotary encoder has five terminals, two of them will probably be connected to an integrated push button.

Connect A to PIN_A, GND to ground and B to PIN_B.

If your rotary encoder is reversed, exchange pins A and B.

The most common rotary encoders use 4 "steps per detent". This is what the DCS-BIOS Arduino Library uses by default. If your rotary encoder uses 2 or 1 steps per detent, specify the number of steps per detent in angle brackets after "DcsBios::RotaryEncoder" like this:

```
// two steps per detent:
DcsBios::RotaryEncoder<2> ilsMhz("ILS_MHZ", "DEC", "INC", PIN_A, PIN_B);
```

If your rotary encoder is a "four steps per detent" model but you specify two steps per detent, you will need to advance it by two detents to move the cockpit dial by one detent. Some people do this intentionally to make the encoder "less sensitive".

6.7 RotaryEncoderVariableStep

The RotaryEncoderVariableStep code snippet works just like the RotaryEncoderFixedStep snippet above, but instead of sending "DEC" and "INC" commands to DCS: World, it sends "-" or "+" followed by a number. This number must be between 0 and 65535 and determines the amount that the cockpit dial moves when the rotary encoder is moved by one detent.

Most code examples set this number to 3200 by default, but you can change it to suit your needs. For a course dial (such as found on an HSI), setting this to 182 (65536/360) will likely advance the virtual cockpit dial by one degree for each detent.

6.8 Potentiometer

The Potentiometer code snippet will read the position of a potentiometer connected to an analog input pin:

```
DcsBios::PotentiometerEWMA<5, 128, 5> stallVol("STALL_VOL", A0);
```

Connect one outer pin of the potentiometer to the supply voltage, the other outer pin to ground, and the middle pin ("slider") to an analog input pin on your Arduino board.

The numbers in the angle brackets are the poll interval in milliseconds (default 5), the hysteresis (default 128) and the divisor (default 5) for an exponentially weighted moving average filter. The default values should be OK for most cases. If the potentiometer is so noisy that it constantly sends new values, try increasing the hysteresis or increasing the divisor. If the potentiometer in the virtual cockpit takes too long to catch up to the real one, you can try decreasing the divisor or decreasing the poll interval.

Note: The hysteresis is applied after scaling the value read from the Arduino's analog input to a range of 0 to 65535. Because the resolution of the Arduino board's analog-to-digital converter is 10 bit (0 to 1024), the hysteresis value should be a multiple of 64 (modifications in smaller increments do not make much sense). The default of 128 is equal to two counts of the ADC.

6.9 ActionButton

The ActionButton code snippet is meant for a momentary push button. Connect your push button like you would with a Switch2Pos code snippet – one terminal to ground, the other to the Arduino pin specified in the code snippet.

When the switch is pushed, the given command will be sent to DCS. When the switch is released, nothing happens.

ActionButtons are used in situations where the hardware on your panel does not directly correspond to what it is operating in the virtual cockpit.

Here are a few examples:

```
// make a push button connected between D3 and ground toggle the ILS Mute
// in the A-10C (handy to hook up to the built-in push button on a rotary
// encoder that controls the ILS volume knob)
DcsBios::ActionButton intIlsUnmute("INT_ILS_UNMUTE", "TOGGLE", 3);

// Have two push buttons on pins 3 and 4 that decrease or increase
// the radio preset in the A-10C
DcsBios::ActionButton intIlsUnmute("UFC_PRESET_SEL", "DEC", 3);
DcsBios::ActionButton intIlsUnmute("UFC_PRESET_SEL", "INC", 4);

// Two limit switches in a refueling door lever assembly that
// put the lever in the open or closed position when the corresponding
// limit switch is pressed:
DcsBios::ActionButton intIlsUnmute("FSCP_RCVR_LEVER", "0", 3); // close
DcsBios::ActionButton intIlsUnmute("FSCP_RCVR_LEVER", "1", 4); // open
```

6.10 LED

The LED code snippet is offered for any number output that can either be 0 or 1. It configures the Arduino pin as an output and turns it on when the value is 1 and off when the value is 0.

On Arduino boards with an ATMega328 or ATMega2560 controller, the I/O pins are powerful enough to drive an LED directly. You will need to connect an LED in series with a matching current-limiting resistor between the Arduino pin and ground.

For a normal Arduino board with a 5V supply voltage, 220 ohm is a common resistor value that will allow about 15 mA of current to flow through your LED. This is a safe value for the vast majority of LEDs and is well below the maximum allowed current for the Arduino pin, which is about 40 mA.

6.11 ServoOutput

The ServoOutput uses the Servo library that comes with the Arduino IDE to position a servo depending on the current value of the output from DCS: World.

A ServoOutput code snippet has three parameters: the pin number connected to the servo's control wire, the servo position (a pulse length in microseconds) to use for a value of 0 (544 by default) and the servo position to use for a value of 65535 (2400 by default).

Tweak the 544 and 2400 values to match the servo position to the correct scale. To reverse it, swap the two numbers.

To use a ServoOutput without getting a compile error, you need to add an "include" statement for the standard Servo library to your sketch. It has to be placed **before** the include statement for the DCS-BIOS Arduino Library:

```
#include <Servo.h>
#include "DcsBios.h"
```

6.12 StringBuffer and IntegerBuffer

The StringBuffer and IntegerBuffer are "generic" code snippets for text and number outputs. They call a function every time a new value arrives. You can add code to that function to do whatever you need to with the new value.

For a StringBuffer, this usually means writing it to a display of some sort.

For an IntegerBuffer, you will likely do some calculations with the value to scale it to the correct unit of measurement, and then display it somewhere or use it to drive a stepper motor to move an indicator needle.

Sometimes it is more convenient to get rid of the callback function and check the current value yourself from code in the loop() function. This is usually the case when you are displaying multiple values on the same display and want to have a sequence of events like this:

- clear the display buffer
- write the first value to the display
- write the second value to the display
- update…

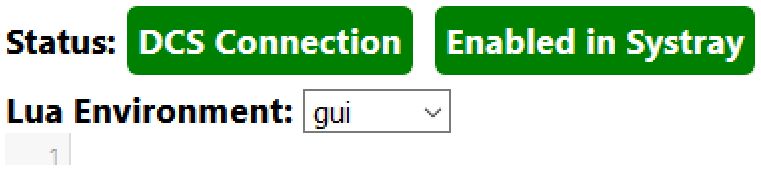
CHAPTER 7

The Lua Console

The Lua Console page in the web interface allows you to execute snippets of Lua code in one of three different environments within DCS ("gui", "export" and "mission").

You can also use it to debug your *Hub Scripts* by choosing the "hub" environment.

At the top of the screen are some status indicators; both have to be green to execute code within DCS: World. To execute code in the "hub" environment, the DCS connection is not required.



If "DCS Connection" is not active, check that you have enabled the Lua Console in the *DCS Connection* page and that DCS: World is running. If "Enabled in Systray" is inactive, enable the Lua Console through the system tray menu.

To use the Lua Console:

- select the environment you want from the dropdown below the status indicators
- enter some code in the first text field
- Press Ctrl+Enter or click the "Execute" button to send the code snippet to DCS: World and wait for the result to be displayed at the bottom

7.1 The Environments

- "hub" executes code in the DCS-BIOS Hub itself. See the *Hub Scripts* section for more information.
- "gui" executes the code in the same environment that the hook that implements the Lua Console is running in. For more information about hooks, see "APIDCS_ControlAPI.html" in your DCS: World installation folder.

- "export" executes the code in the Export.lua environment. This is useful if you are developing a new module definition for DCS-BIOS.
- "mission" executes the code in the environment that the code snippets that are generated by the DCS: World mission editor live in. Note that this is not quite the same environment that scripts added do a mission using the "Do Script" trigger action run in.

7.2 Using the Lua Console to help with mission building

If you are a mission builder, this section will show you a few code snippets that can help you debug your trigger logic or custom Lua scripts. All of these examples assume that the "Environment" drop-down menu is set to "mission". Generally, you can check every condition and execute every trigger action that is available in the DCS: World mission editor.

- Check if flag 42 is set:

```
return c_flag_is_true(42)
```

- Set flag 42 to the value 23:

```
a_set_flag_value(42, 23)
```

- Get a list of values to use with the "X: COCKPIT PARAM..." condition:

```
return list_cockpit_params()
```

- Get a list of indications for the "X: COCKPIT INDICATION IS EQUAL TO" condition:

```
return list_indication(n)
-- n = device ID, see mods/aircraft/.../Scripts/devices.lua
```

- Use a "DO SCRIPT" action to show the current value of a flag in a message:

```
return  a_do_script([[
  trigger.action.outText(trigger.misc.getUserFlag(42), 5 , false)
]])
-- you can also use DO SCRIPT actions to call some of your own custom Lua code.
```

- See a list of what is available in the "mission" environment:

```
local keys = {}
for k, v in pairs(_G) do
    keys[#keys+1] = k
end
return keys
```

CHAPTER 8

Hub Scripts

Note: Hub Scripts are a new feature. Although it has been tested in a practical use case for about a week, the API that is available to the scripts might still be changed if serious bugs or performance issues are discovered with the current implementation.

Hub scripts are scripts written in the Lua programming language that are executed by the DCS-BIOS Hub. A hub script can read data that is coming from DCS ("sim data") and override data that is sent to the serial ports ("panel data"). It can also intercept commands being sent to the DCS-BIOS Hub and send commands to DCS.

Hub scripts are mostly used to make a physical cockpit that was built for a specific airframe work with other DCS: World modules.

8.1 Lifecycle

On the *Dashboard* screen, you can configure a list of hub scripts. These scripts are executed when the DCS-BIOS Hub starts. When the "Reload Scripts" button is clicked, the Lua state is thrown away, a new Lua state is created and all hub scripts are executed again (that means no data survives a click of the "Reload Scripts" button).

8.2 Lua Environment

The DCS-BIOS Hub is using gopher-lua, an implementation of Lua 5.1 in the Go programming language. Lua 5.1 is the same version that DCS: World uses for its Lua scripts. However, as it is written in Go, you won't be able to load Lua libraries written in C++. See also: Differences between Lua and GopherLua

All hub scripts are executed within the same Lua state, but each hub script is loaded in its own global environment. That means you can use global variables in your hub script without worrying about conflicts with other hub scripts.

For debugging, you can use the *Lua Console* and select the "hub" environment. To execute code in the global environment of a specific hub script, use the *enterEnv* function like this:

```
enterEnv("myscript.lua") -- myscript.lua must be a suffix of the script path
-- any code after the enterEnv line will be executed in the global Lua environment
-- of the first hub script whose path ends with "myscript.lua" (case insensitive␣
↪match).

return MyGlobalVariable -- inspect the value of MyGlobalVariable
```

The DCS-BIOS Hub provides a few useful functions in the "hub" Lua module, which is provided automatically in the global variable "hub". The following sections will describe these functions.

8.3 Registering Callbacks

An **output callback** is a function that takes no output parameters and will be called whenever new data from DCS has been received. Use *hub.registerOutputCallback()* to create an output callback:

```
hub.registerOutputCallback(function()
    -- use hub.getSimString() and hub.getSimInteger() here
    -- to access data from the sim
end)
```

An **input callback** is a function receiving two arguments (*cmd* and *arg*) and returning a boolean. It is called whenever a command is received via a serial port.

If the callback function returns *true*, the command is not forwarded to DCS.

An input callback will typically remap commands by using *hub.sendSimCommand()* to send a different command and then returning true to prevent the original command from being sent to DCS:

```
-- remap the "UFC_B1" command from a Harrier cockpit to "UFC_1" in the Hornet:
hub.registerInputCallback(function(cmd, arg)
    local acftName = hub.getSimString("MetadataStart/_ACFT_NAME")
    if acftName == "FA-18C_hornet" then
        if cmd == "UFC_B1" then
            hub.sendSimCommand("UFC_1", arg)
            return true
        end
    end
end)
```

Note: Output callbacks are executed in the order they were registered. An output callback that has been registered later can overwrite panel data that was set by callbacks that were registered earlier.

Input callbacks are executed in the *reverse* order they were registered. If one input callback returns true, the remaining ones will not be called. An input callback that has been registered later can intercept a command before callbacks that were registered earlier get the chance.

This means that when multiple hub scripts want to set the same output value or intercept the same command, the script that is last in the list always wins.

8.4 Sending Commands to DCS: World

You can send a command to DCS: World using the *hub.sendSimCommand* function. For example, to click the master caution button in the A-10C:

```
hub.sendSimCommand("UFC_MASTER_CAUTION", "1")
hub.sendSimCommand("UFC_MASTER_CAUTION", "0")
```

The channel for commands to DCS has a buffer size of 10, so you can send small sequences like pushing and releasing a button without worrying about blocking anything.

8.5 Reading Data from DCS: World

You can access the most recent data that was received from DCS: World with the *getSimString* and *getSimInteger* functions. They take a control identifier of the form *AircraftName/ElementName* and return a string or integer value. If the control identifier is invalid, *getSimInteger* will return -1 and *getSimString* will return the empty string.

Note that calling these functions with control identifiers that do not belong to the currently active aircraft in DCS: World will result in undefined behavior (returning garbage data).

Refer to the next section for an example that uses the *getSimString* function.

8.6 Overriding Panel Data

The DCS-BIOS Hub keeps two copies of export data. One is the *Sim Data* buffer which contains the most recent cockpit state received from DCS: World. The other is the *Panel Data* buffer which contains the data that is sent to the serial ports.

When receiving new data from DCS: World, the following steps are executed:

- Copy the *Sim Data* buffer to the *Panel Data* buffer
- Execute all output callback functions
- Send the current state of the *Panel Data* buffer to the serial ports

The functions *hub.setPanelInteger* and *hub.setPanelString* can be used to overwrite data in the *Panel Buffer*. The first parameter is a control identifier and the second is the new value.

For example, the following output callback will display the F-18C Hornet's UFC data on a simpit that was built for the AV8BNA Harrier:

```
local function remapOutput(a, b)
    hub.setPanelString(b, hub.getSimString(a))
end

hub.registerOutputCallback(function()
    local acftName = getSimString("MetadataStart/_ACFT_NAME")
    if acftName == "FA-18C_hornet" then
        remapOutput("FA-18C_hornet/UFC_COMM1_DISPLAY", "AV8BNA/UFC_COMM1_DISPLAY")
        remapOutput("FA-18C_hornet/UFC_COMM2_DISPLAY", "AV8BNA/UFC_COMM2_DISPLAY")
        local scratchpad = getSimString("FA-18C_hornet/UFC_SCRATCHPAD_STRING_1_DISPLAY
↪")
        scratchpad = scratchpad .. getSimString("FA-18C_hornet/UFC_SCRATCHPAD_STRING_
↪2_DISPLAY")
```

(continues on next page)

(continued from previous page)

```
        scratchpad = scratchpad .. getSimString("FA-18C_hornet/UFC_SCRATCHPAD_NUMBER_
↪DISPLAY")
        setPanelString("AV8BNA/UFC_SCRATCHPAD", scratchpad)

        remapOutput("FA-18C_hornet/UFC_OPTION_CUEING_1", "AV8BNA/AV8BNA_ODU_1_SELECT")
        remapOutput("FA-18C_hornet/UFC_OPTION_DISPLAY_1", "AV8BNA/AV8BNA_ODU_1_Text")
        remapOutput("FA-18C_hornet/UFC_OPTION_CUEING_2", "AV8BNA/AV8BNA_ODU_2_SELECT")
        remapOutput("FA-18C_hornet/UFC_OPTION_DISPLAY_2", "AV8BNA/AV8BNA_ODU_2_Text")
        remapOutput("FA-18C_hornet/UFC_OPTION_CUEING_3", "AV8BNA/AV8BNA_ODU_3_SELECT")
        remapOutput("FA-18C_hornet/UFC_OPTION_DISPLAY_3", "AV8BNA/AV8BNA_ODU_3_Text")
        remapOutput("FA-18C_hornet/UFC_OPTION_CUEING_4", "AV8BNA/AV8BNA_ODU_4_SELECT")
        remapOutput("FA-18C_hornet/UFC_OPTION_DISPLAY_4", "AV8BNA/AV8BNA_ODU_4_Text")
        remapOutput("FA-18C_hornet/UFC_OPTION_CUEING_5", "AV8BNA/AV8BNA_ODU_5_SELECT")
        remapOutput("FA-18C_hornet/UFC_OPTION_DISPLAY_5", "AV8BNA/AV8BNA_ODU_5_Text")
    end
end)
```