EAN-Script Development

PN: EAN-Script-Development

12/4/2018

Contact:

Web: sightlineapplications.com

Sales: sales@sightlineapplications.com

Support: support@sightlineapplications.com

Phone: +1 (541) 716-5137

Export Controls

Exports of SightLine products are governed by the US Department of Commerce, Export Administration Regulations (EAR); classification is ECCN 4A994. The export summary sheet located on the support/documentation page of our website outlines customers responsibilities and applicable rules. SightLine Applications takes export controls seriously and works to stay compliant with all export rules.

Copyright and Use Agreement

© Copyright 2018, SightLine Applications, Inc. All Rights reserved. The SightLine Applications name and logo and all related product and service names, design marks and slogans are the trademarks, and service marks of SightLine Applications, Inc.

Before loading, downloading, installing, upgrading or using any Licensed Product of SightLine Applications, Inc., users must read and agree to the license terms and conditions outlined in the End User License Agreement.

All data, specifications, and information contained in this publication are based on information that we believe is reliable at the time of printing. SightLine Applications, Inc. reserves the right to make changes without prior notice.

Alerts

The following notifications are used throughout the document to help identify important safety and setup information to the user:

⚠ **CAUTION:** Alerts to a potential hazard that may result in personal injury, or an unsafe practice that causes damage to the equipment if not avoided.

ⓘ **IMPORTANT:** Identifies crucial information that is important to setup and configuration procedures.

📄 *Used to emphasize points or reminds the user of something. Supplementary information that aids in the use or understanding of the equipment or subject that is not critical to system use*.

Contents

List of Figures

List of Tables

Appendix Figures

Appendix Tables

1   Overview

This document provides information and steps for developing and running custom scripts in Lua for the 1500-OEM and 3000-OEM video processing boards.

1.1 Developing On-Board Applications

SightLine provides two primary ways for customers to develop their own on-board applications: C/C++ and Lua. Each technology has benefits and costs for solving a problem. It is impossible to prescribe the right technology for every scenario. This section helps provide general guidelines to assist in understanding the tradeoffs.

1.1.1   Lua

Lua is recommended for light-weight applications that need to perform simple data processing and interaction with the onboard video processing application, VideoTrack. Applications such as dynamic on-screen displays based on telemetry data, or simple command and control from serial ports are good uses for Lua. Lua scripts are executed in-line with our video processing and cannot be synchronized with the processing of video frames. Issues such as increased latency and other performance impacts can arise from Lua scripts that can be very complex.

1.1.2   C/C++

If an application requires complex data handling, frequent real-time access to IO, or should be run in parallel with VideoTrack, SightLine recommends creating C/C++ applications that can be run on the ARM processor. Information on creating embedded C/C++ applications can be found in EAN-ARM-Application-Development.

When reviewing options, contact Support to discuss your application.

Table 1: Lua and C/C++ Comparison Table

Benefits	Drawbacks
Lua • Simple to deploy • Frame synchronized execution • Can leverage numerous examples from SightLine or the internet	**Lua** • Not as widely used as C/C++ • Access to IO is complex and difficult • Real-time debugging is not available • Networking is not yet supported
C/C++ • Wide acceptance within the embedded programming industry • Can be easy to test on a PC before deploying on target hardware. • Real-time debugging • Complete access to IO, file system, etc. • Can leverage numerous examples from SightLine or the internet • Can run in parallel to existing applications • Portable to numerous platforms	**C/C++** • Deploying application to launch at run time can be error prone (file location, system permission, etc.) • Existing setup procedure is complex[1] (VMWare, CCStudio, mapped drives, NFS booting, ...)

---

[1] These tools and procedures are complex but used industry wide with TI embedded systems.

1.2 Associated Documents

EAN-Firmware Upgrade Utility: Outlines the steps for installing and running the Firmware Upgrade Utility to manage the firmware, parameter, license and other program files critical to hardware and software functions.

EAN-Startup Guide 1500-OEM: Describes steps for connecting, configuring, and testing the 1500-OEM video processing board on the 1500-AB accessory board.

EAN-Startup Guide 3000-OEM: Describes steps for connecting, configuring, and testing the 3000-OEM video processing board on the 3000-IO interface board.

EAN-Parameter File: Outlines the differences between dynamic and non-dynamic parameter file settings and how to correctly save them to the board.

EAN-ARM Application Development: Describes how to setup a PC to develop applications that can be run on the ARM processor of the 1500-OEM or the 3000-OEM video processing boards

Interface Command and Control (IDD): Describes the native communications protocol used by the SightLine Applications product line. The IDD is also available as a local download on the Software Download page.

Panel Plus User Guide: Provides descriptions of all the settings in the Panel Plus application. (Located in the Panel Plus application in the *Help* menu.)

1.3 SightLine Software Requirements

ⓘ **IMPORTANT:** The Panel Plus software version should match the firmware version running on the board.

2 Example Scripts

The sample applications/scripts installer (*SLA ARM Examples*) can be downloaded from the SightLine Applications website. Run the installer before setting up the development environment.

The example scripts are intended to serve as a starting point for any script development. A summary of each example script is shown below. The example scripts are located in C:\SightLine Applications\SLA-Examples-ARM 2.**.**\SLAScripts\Scripts.

📄 *Some example scripts in the directory are not shown below since they are used internally.*

- helloworld.lua: Sends a command to VideoTrack to draw a text overlay with the following text: *Hello World.* The example program is updated in 2.24 to remove the overlay when the script is unloaded.

- snapshot.lua: Retrieves the current version of SightLine software, starts a track at coordinates 320x,240y, and then takes a snapshot. This example highlights sending commands to the board and retrieving information.

- gpio.lua: Toggles the GPIO based on MTI detections. See EAN-GPIO for more information on available GPIO on the 1500-OEM and 3000-OEM. This example uses GPIO 175, which is a valid GPIO for the 1500-OEM. The example highlights how to get telemetry from the SightLine software, and how to set GPIO from a script.

- lensctrl.lua: A complex script that provides example implementations of auto focus algorithms and other lens control functions. More details can be found in the [Appendix A](#).

- reticles.lua: Script to generate custom on screen display reticles. There are four different types of reticles to choose for each camera. Configuration details for reticles script can be found in the [Appendix B.](#)

- telemdata.lua: Sends commands to the VideoTrack to display telemetry data and messages received by the system. This diagnostic tool provides examples of message parsing.

- telemlogger.lua: Similar to telemdata.lua. Instead of sending commands to draw the information on top of the video, it logs the telemetry data and messages to a file on the microSD card. Since the path to the microSD card is different from the 1500-OEM and the 3000-OEM, it should be correctly set in the SLLoad function for the platform being used. Examples are provided for both. Uncomment the correct one and comment out the others.

- snapFocus.lua: This script uses focus metric and takes a group of snapshots when the focus is greater than the focus of a past window of frames. It also demonstrates how to kick off a script task using an SightLine command. Focus metric telemetry must be enabled using SLACoordinateReportingMode_t. More user information is included in comments in the script.

- klvstatic.lua This script pushes static metadata values to VideoTrack for KLV output with streaming video.

Table 2: Example Scripts and Script Interfaces

Example Scripts	Software Version	Script Interfaces	Software Version
klvstatic.lua	2.25.07	SLUnload	2.24.xx
snapFocus.lua	2.25.07	SLLoad	2.24.xx
telemdata.lua	2.25.xx	SLNewCmdCallback	2.24.xx
Telemlogger.lua	2.25.xx	SLPostAnalyze	2.23.xx
helloworld.lua (updated)	2.24.xx		
lensctrl.lua	2.24.xx		
reticles.lua	2.24.xx		
snapshot.lua	2.23.xx		
gpio.lua	2.23.xx		

3   Development Environment

Lua scripts can be developed with almost any IDE, however to get syntax highlighting, static analysis, and autocomplete capabilities, SightLine recommends using the ZeroBrane IDE from [ZeroBrane Studio](#).

In this example, the application has been installed to C:\ZeroBraneStudio. The steps below reference the install location as *ZBS*.

1. Download and install ZeroBrane IDE from ZeroBrane Studio.

2. Find sla_internal.lua in the installed sample directory /SightLine Applications/SLA-Examples-ARM 2.**.**/SLAScripts/ and copy it into ZBS\api\lua.

    For version 2.23.xx, find - sla.lua

3. Open ZeroBrane Studio and choose the menu option project directory: *Project* » *Project Directory* » *Choose*.

4. Select the directory with the example scripts. C:\SightLine Applications\SLA-Examples-ARM 2.**.**\SLAScripts\Scripts.

5. Choose the menu option *Edit* » *Preferences* » *Settings:User*. This opens a user settings file. Add the following line to end of the file:

    api = {'sla_internal'}

    For version 2.23.xx add the following line to the end of the file: *api = {'sla'}*

6. Restart the ZeroBrane Studio application.

7. The *helloworld.lua* example can now be edited. To verify autocomplete is working, type *ffi.n* after the definition of framecount. It should show *new* as an option.


📄 *ZeroBrane Studio includes a static analyzer. To use this, go to the menu option Project » Analyze after editing a file. This is strongly recommended before sending files to the SLA-hardware. If using a different IDE that doesn't contain a static analyzer, there are other third-party tools available on the web.*

4   Uploading Scripts

User developed scripts can be uploaded to the SLA-hardware using the SLA-firmware upgrade utility.

1.   Click Find IP Address to look for all the devices on the network

2.   Select the target hardware from the list of devices that were discovered

3.   From the upgrade utility menu select *File » Program Files » Put Files (PC->SLA) …*
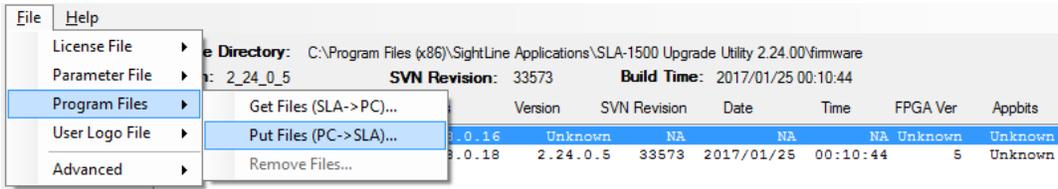
4.   Select the directory containing the developed Lua scripts or the *Example* scripts directory. This uploads the all the Lua files (*.lua) from that directory to the SightLine hardware.

   📄 *The upgrade utility can also be used to retrieve all the scripts from the hardware.*

5.   Confirm success by checking the *Status* window
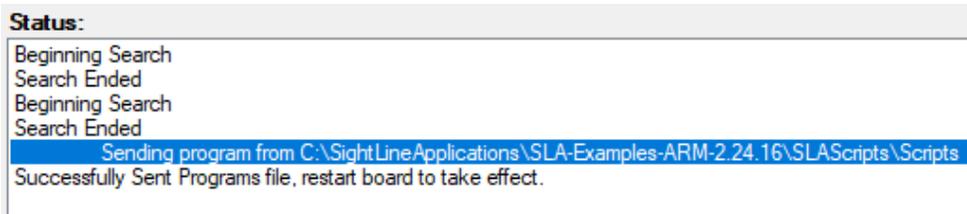
   📄 *Ignore the Restart board to take effect alert.*

6.   Once the scripts are uploaded, they can now be enabled (see below).

5   Enabling / Disabling Scripts

Scripts can only run if they are enabled. This can be done through the command and control protocol using the *Set External Program (0x8F)* message. It can also be done from the *External Programs* dialog window in Panel Plus, main menu » *File » Programs*.

To enable a script, select it in the drop-down menu and click *Send*. To disable a script, select *None* and click *Send*.
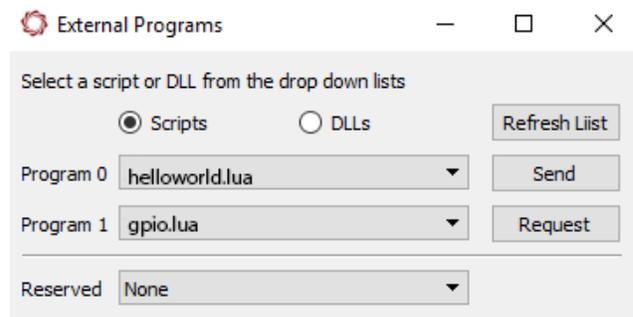

Figure 1: External Programs

Sending the message again or clicking the *Send* button again will cause the scripts to be reloaded. This is especially useful when debugging or incrementally adding functionality to the script.

   📄 *If a script uses another script internally, include the keyword internal in the script name so it will not show up in the Panel Plus list of programs.*

5.1   Running Scripts at Startup

To run the script at startup, enable the script and then save the settings to the board, main menu » *Parameters* » *Save to board*.

ⓘ **IMPORTANT:** Problems with scripts show up as a user warning. When loading scripts, it is important to monitor the *User Warning* dialog window. From the Panel Plus main menu, go to *View* » *User Warning*.
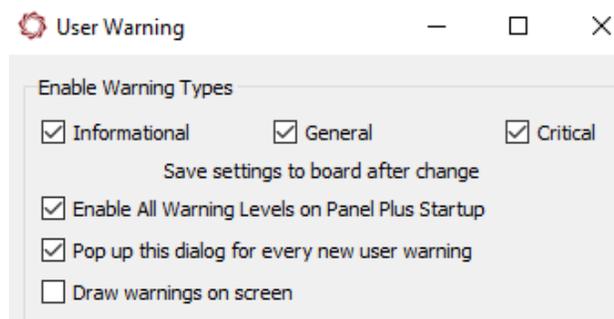
Figure 2: User Warning Dialog

6   Script Interface

The script interface to VideoTrack is documented in the IDD. The IDD provides details on structures, functions and other protocols available through the script interface.

6.1   Creating a Script Function

The example below shows below describes how to create a script function/definition from the IDD.

Example: To add an SLADoSnapshot command to a script, the SLADoSnapShot_t struct is defined in the IDD as shown in Figure 3

📄 *This can change with software versions.*

Description

Execute an image snapshot to the MicroSD Card or an external FTP server.

```
typedef struct {
  u8 frameStep;
  u8 numFrames;
  SVPLenString_t fileName;
  u8 snapAllCameras;
  u8 shouldScan;
} SLADoSnapShot_t;
```

Figure 3: SLADoSnapShot_t struct in IDD

All variable names and valid values are called out in the IDD. Variable names are case sensitive. Optional values (varies) do not need to be defined.

Message ID 0x60

Byte Offset	Name	Description
4	frameStep	Frame Step – step between frames (e.g. 2 = every other)
5	numFrames	Number of snapshots to take (1 to 254), 255 = continuous, 0 = Stop; Ignored if Snap All Cameras is used
6	Filename.len	String length
7-…	Filename.str	Base file name of saved files
varies	snapAllCameras	Mask of Cameras to Snap (for example, use 0x5 to snap cameras 0 and 2); For multicamera, only single snap-shot is allowed.
varies	shouldScan	When using file auto-numbering, begin file numbering after highest-numbered existing filename
varies	autoFolder	Creates new files every maxFiles
varies	maxFiles	Max files per folder (2000 is default, 20000 is max). Note: maxFiles is the maximum number of files per folder, the folder may auto increment sooner under certain conditions such as snapshots being taken too fast.

Using the above information, the Lua code segment shown in Figure 5 can be created to take a snapshot.

```lua
local snap = ffi.new("SLADoSnapShot_t")
snap.frameStep = 30 -- every 30 frames
snap.numFrames = 255 -- continuous
snap.fileName.str = "Snapshot_"
snap.fileName.len = string.len(snap.fileName.str)
local result = ffi.C.SLADoSnapShot(_vtstate, snap, out, 3)
if result ~= SUCCESS then
    print( "Failed to take snapshot\n" )
end
```

Figure 4: Lua Code Segment for Snapshot Command

7   Key interfaces From VideoTrack to Script

7.1   SLPostAnalyze

This function is called by VideoTrack immediately after the analysis of a frame is complete. The analysis step includes registration, tracking, detection, etc. All scripts should implement this function.

In the *helloworld.lua* example, this function maintains a global framecount, and then uses it to determine when to draw *Hello World* on the video. This function is passed two parameters:

- _vtstate: This is a handle to the VideoTrack context and is needed in calls back to the VideoTrack program such as ffi.C.SLADrawObject shown in the example.

- cameraIndex: Provides the camera index for which the script is being called. Allows users to take actions on a specific camera or keep camera specific data separate. On the 3000-OEM, the *SLPostAnalyze* function is called for all actively processed cameras.

📄 *On the 3000-OEM the SLPostAnalyze function is called for all actively processed cameras. It is up to the user to decide how to use the cameraIndex. For example, to draw Hello World for camera two, return at the top of SLPostAnalyze if the camera index is not equal to 2.*

```lua
15    -- Global frame count, incremented in SLPostAnalyze
16    local ffi = require("ffi")
17    local framecount = 0
18
19    -- At frame 100 print "Hello World Analyze" onto the video and out the console.
20    -- This function is called after SightLine VideoTrack software processes a frame
21    -- of data.
22    function SLPostAnalyze( _vtstate, cameraIndex )
23        framecount = framecount + 1
24
25        if framecount == 100 then
26            local out = ffi.new("SVPOut_t");
27            local rv = ffi.new("SLAUnion");
28            out.out = rv;
29
30            local drawobj  = ffi.new("SLADrawObject_t");
31            drawobj.objId  = 98
32            drawobj.action = 1
33            drawobj.propertyFlags = 132
34            drawobj.type = 6;
35            drawobj.a = 100
36            drawobj.b = 100
37            drawobj.c = 8224
38            drawobj.d = 0
39            drawobj.backgroundColor = 16
40            drawobj.text.str = "Hello World " .. cameraIndex   -- Lua uses ".." for string concatenation
41            local result = ffi.C.SLADrawObject(_vtstate, drawobj, out, ffi.sizeof(drawobj))
42            -- It is a good idea to check the return from the calls back into SLA software to ensure success
43            if result == 0 then
44                print("Hello World " .. cameraIndex) -- This prints to the console
45            else
46                print("Failed To Send Hello World " .. cameraIndex) -- This prints to the console
47            end
48
49        end
50
51    end
```

Figure 5: SLPostAnalyze usage from Hello World Example

7.2   SLUnload (New in 2.24.xx)

This function is called by VideoTrack whenever the script is disabled. This can happen if the board is shutting down or if the script is disabled through Panel Plus. In the *helloworld.lua* example, this function removes the *drawObject with id 98* created in SLPostAnalyze. This function is passed only one parameter:

_vtstate: This is a handle to the VideoTrack context and is needed in calls back to the VideoTrack program such as ffi.C.SLADrawObject shown in the example.

```
-- When unloading the script remove any objects that the script has added to the video.  This
-- isn't a requirement of the script.
function SLUnLoad(_vtstate)
    local out = ffi.new("SVPOut_t");
    local rv = ffi.new("SLAUnion");
    out.out = rv;

    local drawobj  = ffi.new("SLADrawObject_t");
    drawobj.objId  = 98
    drawobj.action = 0
    drawobj.propertyFlags = 132
    drawobj.type = 6;
    drawobj.a = 100
    drawobj.b = 100
    drawobj.c = 8224
    drawobj.d = 0
    drawobj.backgroundColor = 16
    drawobj.text.str = "" -- Note since we are just removing many of the parameters don't matter
    local result = ffi.C.SLADrawObject(_vtstate, drawobj, out, ffi.sizeof(drawobj))
    -- It is a good idea to check the return from the calls back into SLA software to ensure success
    if result == 0 then
        print("Removing Hello World") -- This prints to the console
    else
        print("Failed To Remove Hello World") -- This prints to the console
    end
end
```

Figure 6: SLUnload Usage from Hello World Example

7.3   SLLoad (New in 2.24.xx)

This function is called by VideoTrack when loading the script. This function is passed only one parameter.

vtstate: This handles the VideoTrack context and is needed in calls back to the VideoTrack program.

7.4   SLNewCmdCallback (New in 2.24.xx)

This function is called by VideoTrack when it receives a command. This function is passed four parameters:

- _vtstate: This is a handle to the VideoTrack context and is needed in calls back to the VideoTrack program.
- _temp: This is a handle to the structure containing the message data.
- len: This is the length of the message data.
- type: This is the message type

```
54      -- Example of parsing commands sent to VideoTrack.
55    function SLNewCmdCallback( _vtstate, _temp, len, type )
56        if type == 0x05 then
57            local temp = ffi.cast("SLAModifyTracking_t*", _temp)
58            print( "Tracking Row = "..temp.row )
59        end
60    end
```

Figure 7: SLNewCmdCallback Example

8   Trouble Shooting

Problem	Recommendation
Script does not run.	From the Panel Plus main menu, go to *View* » *User Warnings* and enable the user warnings. Start the script and monitor the User Warning dialog window to determine if there are any User Warning preventing the script from running.
Cannot debug script.	Print statements are helpful but require access to the Linux console to see the output. This requires a serial connection as well as modifying the silent argument in u-boot. Contact support if you have further questions on how to do this. Additionally, debug statements can be drawn to the screen in the same manner using the helloworld.lua example.
System is sluggish after loading script.	From the Panel Plus main menu, go to *View* » *Performance Graphs*. Select *Enable System Status* and *CPU Timing*. After a few seconds *ARM* should appear in the list of timings. Expand this and look at the time for *Post AnalyzeScript*. The times presented are (*Average, Minimum, Maximum*) over 100 samples in microseconds. If these numbers are very large, e.g., several thousand microseconds, reevaluate the level of complexity of the script.

8.1   Questions and Additional Support

For questions and additional support, please contact Technical Support. Additional support documentation and Engineering Application Notes (EANs) can be found on the Support pages of the SightLine Applications website.

Appendix A - Lens Control Script

New in software version 2.24.xx.

A1   Overview

The lens control example script shows users how to implement a basic auto focus algorithm in Lua. It also shows how to receive lens commands sent from Panel Plus or another user interface and translate those commands into lens commands such as zoom or focus.

This example also shows how to interface with a serial port from Lua, how to do bitwise operations in Lua, plus many other helpful functions. This script was not designed to provide an out-of-the box auto focus algorithm that is plug-and-play for any system. An in-depth knowledge of the lens and camera system is required to take this from an example and turn it into a working product.

A2   Testing 1500-OEM and 3000-OEM

ⓘ **IMPORTANT:** A Sony or Hitachi camera is needed for this procedure.

The following instructions assume a working knowledge of how to modify and load scripts to the SLA-hardware.

1. Using Panel Plus, configure the system to stream video from the digital camera.

2. In Panel Plus verify that serial port 2 is configured as *Port Not Used*. If not, make the change, save the parameters, and then restart the system.

3. Open the *Lens* tab in Panel Plus and verify that the *Lens Type* is set to *None*. If not, make the change, save parameters, and then restart the system.

Lens Type: [None ▼] Port Num [0 ⏶⏷] *Disable Serial Passthrough

4. To switch from the Sony to the Hitachi camera, edit *lensctrl.lua* to require the hitachictrl_internal.lua script as shown below.

```
-- Use the correct "Lens/Camera" interface here.  Currently the options are
-- "hitachictrl" and "sonyctrl".   At this point this should be the only thing you
-- need to change to go between the Sony and Hitachi cameras.
local lensctrl = require("hitachictrl_internal")
```

5. Send the scripts to the 1500-OEM/3000-OEM, and then load the *lensctrl.lua* script.

6. Verify basic functionality by using the *Narrow* or *Wide* zoom controls.

A3   Basic Troubleshooting

•   Monitor the user warnings in Panel Plus.

•   If there are no user warnings and the camera is not responding, try different baud rates. Common baud rates for the Hitachi are 4800 and 9600. Common baud rates for the Sony are 9600 and 19200.

•   If the camera is still not responding, contact Technical Support.

A4 3000-OEM Serial Ports

The 3000-OEM serial ports are somewhat different. If the camera is connected on VIN0 no changes should be needed. If the camera is connected on VIN1, the scripts that open serial ports need to be updated to port 3 instead of serial port 2 (see *sonyctrl_internal.lua* or *hitachictrl_internal.lua*).

If multiple cameras are being processed on the 3000-OEM, add code to exit early from the *SLPostAnalyze* function for the non-lens-controlled camera.

```lua
-- This function is called after SightLine VideoTrack software processes a frame
-- of data.
function SLPostAnalyze(_vtstate, cameraIndex)
  if not cameraIndex == 0 then
    return
  end
```

Figure A1: Exit from SLPostAnalyze Function - Multiple Cameras

The example script does not check the commanded camera. To create auto focus scripts for two different cameras, additional code should be added to check the camera in the *SLNewCmdCallback* function.

Appendix B - Reticle Selection Script

New in software version 2.24.xx.

B1 Overview

Reticle selection scripts can be used to draw overlay reticles on top of the stream and can be customized for each camera. Two script files are available:

1. ReticlesConfig_internal.lua: Configuration parameters to customize the display of reticles.

2. Reticles.lua: Adds reticles to the display image. This is the primary script file that should be loaded to the target to display the reticles on the screen.

B2 Configuring Reticle Scripts

ⓘ **IMPORTANT:** Make sure to edit *reticlesConfig_internal.lua* with a text editor before uploading *reticles.lua* to the hardware.

Table B1: User Configuration Parameters

Parameter	Description
CAM<X>_RETICLE_INDEX	Reticle to draw for camera with index X. See Reticle index section for available options.
CAM<X>_FIELD_OF_VIEW	Field of view of camera with index X (degrees).
CAM<X>_CIRCLE_DEGREE	Diameter of Circle (degree) of camera with index X.
CAM<X>_CENTER_CIRCLE_RADIUS	Radius of the center point (circle) of camera with index X, default is 2.
CAM<X>_COLOR_FOREGROUND	Foreground color(Color 1) to draw for camera with index X. Check Reticle Color mapping section for options.
CAM<X>_COLOR_BACKGROUND	Background color(Color 2) to draw for camera with index X. Check Reticle color mapping section for options.

Example setting for Camera 1:

```
CAM1_RETICLE_INDEX = 2
    – set camera 1 to use reticle with index 2
CAM1_COLOR_FOREGROUND = 13
   – set camera 1 foreground color as yellow
CAM1_COLOR_BACKGROUND = 12
  – set camera 1 background color to orange
CAM1_FIELD_OF_VIEW = 50
  – set camera 1 field of view to 50 degrees
CAM1_CIRCLE_DEGREE = 10
  – set camera 1 field of view to 10 degrees
CAM1_CENTER_CIRCLE_RADIUS = 2
  – set camera 1 center point radius to 2
```

Field of View: Field of view of camera (degrees) is used to calculate the size of lines used to draw the overlays.

Circle Degree: Diameter of Circle (degree) relative to the field of view.

Center Circle Radius: Center point used in some reticles. Diameter of the circle in Degrees.

B3   Reticle Color Mapping

Reticle color mapping to be used for background and foreground color settings in the configuration file.
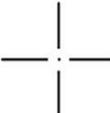
Table B2: Reticle Color Mapping

Color	Value	Color	Value
WHITE	0	DARK BLUE	7
BLACK	1	LIGHT GREEN	8
LIGHT GRAY	2	GREEN	9
GRAY	3	DARK GREEN	10
DARK GRAY	4	RED	11
LIGHT BLUE	5	ORANGE	12
BLUE	6	YELLOW	13

B4   Reticle Index

There are four different types of reticles available now with index ranging from 0 to 3. Each camera index is assigned a single reticle. To choose a different reticle for a certain camera change corresponding camera reticle index. For example: To change camera 2 to use reticle with index 0 change CAM2_RETICLE_INDEX = 0. All the reticles will be drawn with the same color combination based on COLOR_TO_DRAW value.

Table B3: Reticle Indexes

Reticle Index	Picture	Reticle Index	Picture
0		2	
1		3	