EAN-ARM-Development

2023-04-24

Exports: Export Summary Sheet

EULA: End User License Agreement

Web: sightlineapplications.com

Sales: sales@sightlineapplications.com

Support: support@sightlineapplications.com

Phone: +1 (541) 716-5137

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.

1 Overview

This document describes the development environment setup and build, deployment, and debug processes for applications compiled to run on the ARM processor on SightLine 4000-OEM and 1750-OEM video processing boards.

See the EAN-ARM-Development-1500-3000-OEM for build and deployment processes for applications built to run on the 1500-OEM and 3000-OEM ARM processor.

IMPORTANT: OEM references are for both the 4000-OEM and 1750-OEM except where noted.

The sample programs for ARM-side development:

- can be built on the 4000-OEM processing using the GNU G++ compiler for the ARM64 architecture or on the 1750-OEM processing using the Poky G++ compiler,
- can communicate with the VideoTrack application running on the SightLine OEM hardware using the SightLine Command and Control protocol over IP sockets,
- can additionally be built and run on Windows using Visual Studio.

This allows a large part of the development and debugging of custom user applications to be done on a windows platform.

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 not possible to describe the right technology for every situation. However, this section helps provide general guidelines to assist in understanding the benefits and 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 VideoTrack application. 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 can 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. See the EAN-Script Development for more information for developing and using Lua scripts.

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.

When reviewing options, contact Support to discuss a specific 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 OEM 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] (CCStudio, mapped drives ...)

1.2 Sample Applications

This document was written for the SLA-Gimbal sample application. However, the same procedures are applicable to other sample applications described below. The sample applications send data to VideoTrack on port 14003 reserved for user programs. The sample applications receive responses and telemetry on port 16002. Multiple applications can simultaneously transmit packets to port 14003, but each application must use a unique receiving port.

Each of the following applications uses predefined example values for serial ports, network ports and IP Addresses. These will be different for specific customer applications. Please review the code and replace the example values in the sample application as needed. These values will be shown in all capital letters, e.g., from gcMain.cpp: `IPADDR_VIDEOTRACK`, `SLFIP_TO_BOARD_PORT2`, `GC_FROM_VT_PORT`.

SLAGimbal — Provides an example of controlling a gimbal to follow a track. Receives TrackPosition messages from the VideoTrack ARM application over a local UDP socket. Track position information is used to update a PID controller, which then sends serial control commands to an external gimbal microcontroller to steer in the direction of the track.

SLAGPIO — Toggles SD card video recording on/off using a GPIO input, displays recording status via an LED attached to another GPIO output, and initiates a snapshot to the SD card using a second GPIO input.

Only the snapshot feature works on the 1750-OEM.

SLALandingApp — Receives landing aid telemetry from VideoTrack and sends commands to an autopilot.

Overlay DLL — Provides access to the image data prior to encoding for rendering custom overlays. This creates a library and is enabled differently than the above applications. See the Overlay DLL - Build and Enable section.

SLAPelcoTelemetry — Provides a simple and complete example to create a control loop system that moves a camera so that the tracked object remains in the center of the scene. Uses the Pelco D camera control protocol and telemetry data that has been evaluated on the 4000-OEM.

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

1.3 Development and Debugging - Windows Visual Studio

Each of the sample projects listed above includes a Windows Visual Studio 2017 solution.

This process allows a large part of the development and debugging of custom user applications to be done on a Windows platform.

For example, if a custom user application is setup to communicate with an external Gimbal over Serial 2 on the ARM hardware, the application development can be done using a PC and a PC serial port connection to the Gimbal. Any communication between the user application and the VideoTrack application (for commands/telemetry) can also be done on the PC.

SightLine software uses an operating system abstraction so that Windows and Linux serial ports, network ports, and threads all use a common interface. After developing, testing, and debugging on a PC the application can be deployed to the ARM with few changes, e.g., numbering of serial/network ports and IP addresses.

The OS conversion is accomplished during compilation using:

```
#ifdef _WIN32 or #ifdef LINUX
```

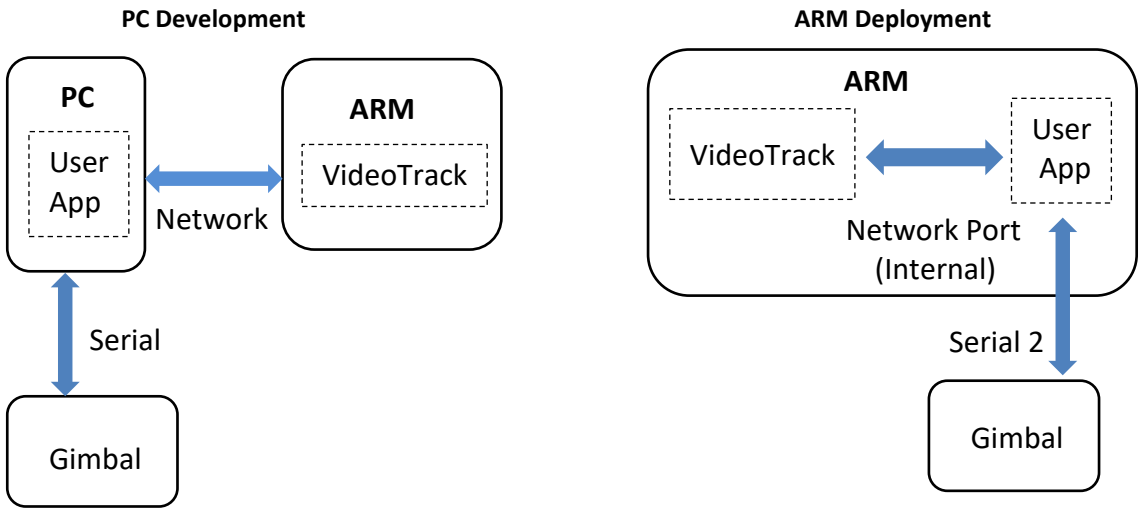


Figure 1: PC Development and ARM Deployment

The code includes #defines for serial and network addresses to make this conversion simple.

From gcMain.cpp:

```
#ifdef _WIN32
  #define MICROCTRL_SERIALPORT 4  // replace with your PC COM PORT
  #define IPADDR_VIDEOTRACK   "192.168.1.207"
  #define IPADDR_LISTEN       "0.0.0.0"
#else
  #define MICROCTRL_SERIALPORT  0 // SLA-Board Serial Port
  #define IPADDR_VIDEOTRACK     "127.0.0.1"
  #define IPADDR_LISTEN         "0.0.0.0"

#endif
```

1.4 Associated Documents

EAN-GPIO-and-I2C: Describes how to create an application compiled for the SightLine OEM processor that reads the GPIO state and sends commands to the VideoTrack application.

EAN-Script-Development: Describes everything needed to develop and run custom scripts in Lua on the OEM hardware.

EAN-Startup-Guide-Tracker-Integration: Describes an example project using SightLine software, hardware, and camera interface to create a control loop system that moves a camera so that the tracked object remains in the center of the scene.

1.5 SightLine Software Requirements

IMPORTANT: Starting with 3.6.x software and above, only the 4000 and 1700 platforms will be supported. The 1500 and 3000 platforms will continue to be supported in 3.5.x software. Some features in 3.6.x and above may not be available on 1500 and 3000 platforms.

IMPORTANT: The Panel Plus software version should match the firmware version running on the board. Firmware and Panel Plus software versions are available on the Software Download page.

2 Third Party Software

The software listed is based on working in a development environment using a PC running Windows 7 or above.

Tera Term (recommended) or PuTTY: Terminal emulator programs used for debug output, or to issue commands on SLA hardware.

WinSCP: A file transfer utility used to access the file system on the OEM video processing boards.

3 Installation

3.1 Sample Application Installation

Download the ARM code examples package from the example code page on the SightLine website. Launch the installer and follow the installation prompts.

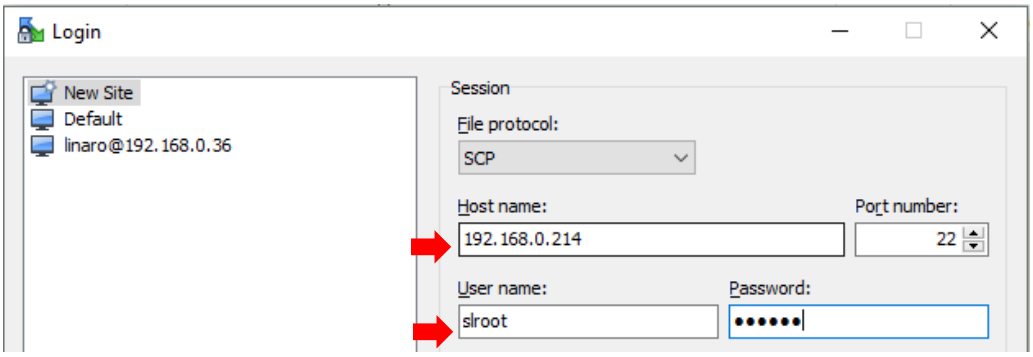
Most of the sample applications include Visual Studio solutions and scripts for building the code on the OEM. While the Visual Studio solutions can be used to build windows applications and are designed to aid in the development and debugging process, the following sections only cover the build and deployment processes for applications on the OEM.

Before starting the examples must be copied to the OEM first. To copy the files, use the WinSCP application.

For more information on how to download and setup WinSCP see EAN-Using-WinSCP. When logging into the 4000-OEM or 1750-OEM the username and password are both *slroot*.

Copy examples to OEM using WinSCP:

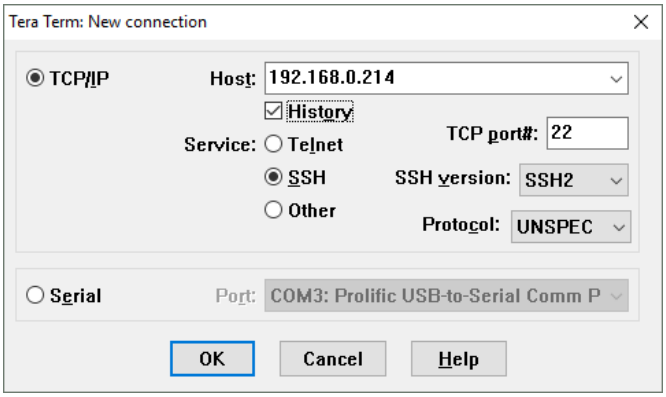
1. Open WinSCP and login to the board.



2. Navigate to the folder where the ARM example is installed, e.g., *C:\SightLine Applications\SLA-Examples-Arm ***\4000*
3. Copy the file *SLAArmExamples.tgz* to /home/slroot/ on the OEM.

4 Start Development

1. Use Tera Term to establish an SSH session to the OEM. Use the username and password: *slroot*.



2. Extract the samples by running the following command:

```
SD> tar -xvf SLAArmExamples.tgz
```

3. Navigate to the SLAGimbal directory *SD> cd SLAArmExamples/SLAGimbal*.
4. Build the application using the build script in the SLAGimbal directory.
   For the 4000-OEM: `SD> ./buildGimbal.sh`
   For the 1750-OEM: `SD> ./buildGimbal.sh -b 1700`
5. Run the application:
   SD> ./SLAGimbal.out

```
SD> ./SLAGimbal.out
runMain Called
Later in runMain
Calling GcStartup
GOT VERSION 3.0
GOT VERSION 3.0
```

Figure 2: Example Output

Make sure that the MICROCTRL_SERIALPORT define at the top of gcMain.cpp is set to the serial port on the OEM that is uses to talk to the external microcontroller.

4.1 Deploying the Application

Copy *SLAGimbal.out* to /home/slroot/sl/bin/.

4.1.1 OEM Startup Script

The following two scripts are executed by the OEM during the startup process:

- /home/slroot/sl/scipts/sla_init.sh
- /home/slroot/sl/scripts/vt_start.sh

SightLine recommends modifying vt_start.sh (Figure 2) to start SLAGimbal automatically when the board is powered on. This is a better option than modifying sla_init.sh, because vt_start.sh is run as the user slroot, and sla_init.sh is run as root.

SightLine recommends running applications as a user whenever possible.

```
#!/bin/bash
cd /home/slroot/sl/bin
../scripts/setHdmiModes.sh
./SLAGimbal.out |& logger -t SLAGimbal &
#./VTNext_ARM64_Release.out -FULL
sleep 2
(./VTNext_ARM64_Release.out -FULL |& logger -t VTNext ; dmesg | logger -t dmesg) &
```

Figure 3: OEM Startup Script

The SLAGimbal example application was improved in version 3.0.xx to work with the OEM startup sequence. Instead of launching the application and immediately sending commands to the OEM, it requests the version once a second until it receives a response, and then continues the setup.

This ensures the system is running before sending other setup commands. It also removes the need for sleep commands in the script.

In the example above the output of the SLAGimbal is redirected to a logger, which is a Linux utility. To view the output of the application try using the following command and to look at the logger output.

Display all output from SLAGimbal application:

```
strings /var/log/user.log | grep SLAGimbal
```

Display the last 100 lines of the log:

```
strings /var/log/user.log | tail -100
```

4.2 Overlay DLL - Build and Enable

Follow the previous steps in the Start Development section to extract the ARM examples.

1. Navigate to the Overlay DLL directory *SD> cd SLAArmExamples/overlay_dll*.

2. Build the library using the build script *SD> ./buildOverlayDll.sh*. This will build the library *x_SLOverlay_DLL_Release.so*
3. Copy the file to the bin directory *(/home/slroot/sl/bin) SD> cp x_SLOverlay_DLL_Release.so ../..*
4. Reboot the OEM system.
5. Connect to the system with Panel Plus.
6. Enable the library:
   a. From the main menu » *File* » *Programs*.
   b. Select the DLLs option, and then select *x_SLOverlay_DLL_Release.so* in the list.
   c. Click *Send*.
7. From the main menu » *Parameters* » *Save to Board,* and then main menu » *Reset Board* to persist the settings through subsequent restarts.
8. Verify a red X is drawn over the video

5 Questions and Additional Support

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

Appendix A - Managing Parameter File - ARM or PC Application Development

When developing an ARM or PC application to send SLA commands to the OEM hardware, commands sent by external applications can affect the state of the system when saving the parameter file. To alleviate these issues SightLine recommends the following guidelines when managing the parameter file:

- Disable ARM or PC applications when configuring and saving parameters to OEM hardware.
- Configure a single system with parameters and test the configuration.
- If the configuration test passes, use the SightLine upgrade utility application to retrieve the parameter file from the OEM hardware and save it to a separate location as a known good system configuration file.
- The upgrade utility can then be used to upload the known good system parameter file to OEM hardware.

See the EAN-Firmware Upgrade Utility for information on how to use the SightLine upgrade utility to manage the parameter file.