EAN-ARM-Development-1500-3000-OEM

2022-10-10

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 OEM video processing boards.

The sample programs for ARM-side development:

- can be built under Windows using CodeSourcery compiling tools,
- use TI CodeComposer Studio 5.x for development, debugging, and deployment,
- 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.

See the EAN-ARM-Development-4000-OEM for build and deployment processes for applications built to run on the 4000-OEM ARM processor.

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 VideoTrack application. Applications such as dynamic on-screen displays based on telemetry data, or simple command and control from serial ports are effective 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 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 SLAGimbal 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 sample applications uses predefined example values for serial ports, network ports and IP Addresses. These will be different for specific customer applications. Review the code and replace the example values in the sample applications as needed. These values will be in 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 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.
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.
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. Evaluated on the 3000-OEM and 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
```

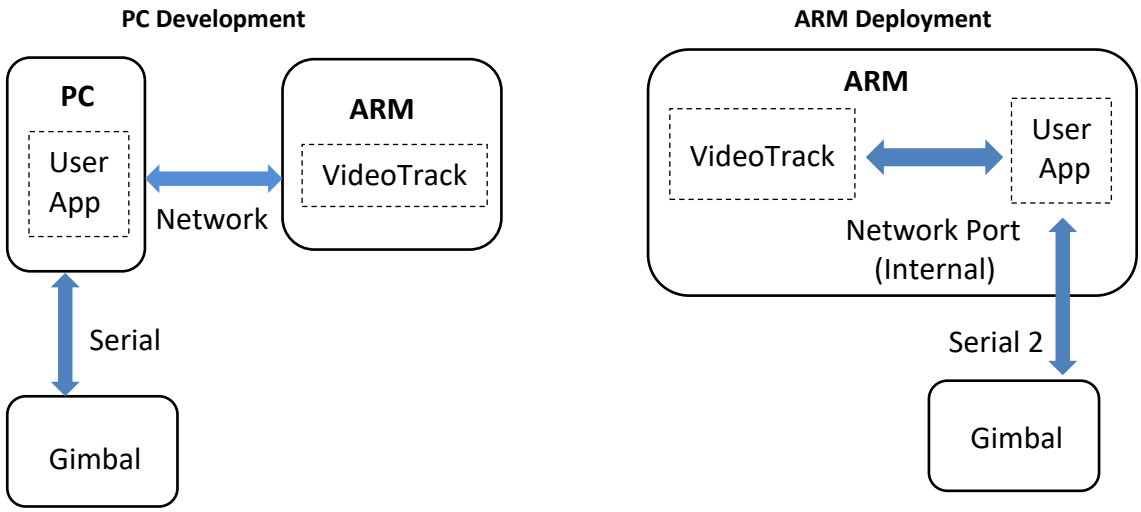


Figure 1: PC Development and ARM Deployment

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 Additional Support Documentation

EAN-GPIO-and-I2C: Describes how to create an application compiled for the 1500-OEM ARM processor that reads the GPIO state and sends commands to the VideoTrack1500 application.

EAN-Script Development: Describes everything needed to develop and run custom scripts in Lua on the 1500-OEM and 3000-OEM hardware.

1.5 SightLine Software Requirements

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

2 Third Party Software

The software listed in Table 2 is based on working in a development environment using a PC running Windows 7 or above.

Table 2: Third Party Software Requirements

Software	Purpose
SightLine provided Linux sources	Hosts Linux sources Map these files as a shared folder for PC development.
Code Composer Studio from Texas Instruments (see the Code Composer Studio section)	Used to compile applications for deployment on SightLine OEM hardware.
CodeSourcery from Mentor Graphics (see the CodeSourcery section)	GDB Debugger
Tera Term (or PuTTY)	SightLine recommends Tera Term for troubleshooting, debugging, and issuing commands on SightLine OEM hardware.

3 Conventions

SightLine hardware	SightLine embedded video processor (1500-OEM, 3000-OEM, etc.)
OMAP Logic #	Typed U-Boot command to serial terminal on 1500-OEM
SLA3000#	Typed U-Boot command to serial terminal on 3000-OEM
root@sla1500:~#	1500 Linux console (usually through serial port, or use SSH terminal)
root@sla3000:~#	3000 Linux console (usually through serial port, or use SSH terminal)
PC>	Windows command prompt (cmd.exe). Actual command prompt may be different.)
<< some text >>	Indicates that user supplied information is required in place of << some text >>

4 Preparation

4.1 Required Linux Sources

Download the Linux sources (VHDX Disk File) from the Example Code page of the SightLine Applications website. The Linux sources are stored on a Hyper-V virtual hard disk image (VHDX). Under Windows, this will be assigned drive letter *R*. Code Composer Studio will use this drive to pull in sources when building applications for the ARM processor on the 1500-OEM and 3000-OEM.

Previous versions of documentation for software 3.4.x reference using VMWare. In software version 3.5.x VMWare is no longer required. All Linux sources that are needed to compile ARM Applications are provided in the Linux sources archive.

4.2 Setup R: Drive

To mount the Linux sources image and assign drive letter R, navigate to where the disk image was downloaded. Use the following PowerShell command:

```
mount-diskimage SLA_1500_3000_ARMDEV_3_5_2.vhdx -NoDriveLetter | Get-Disk |
Get-Partition | Where -Property PartitionNumber -eq "2" | Set-Partition -
NewDriveLetter R
```

Alternate method:

1. Right click the *SLA_1500_3000_ARMDEV_3_5_2.vhdx* file and select *Mount.*

After the image is mounted the virtual disk must be assigned drive letter R.

2. Open the start menu, enter *Disk Management* and select *Create and format hard disk partitions*.
3. Right-click the virtual drive that was mounted and select *Change Drive Letter and Paths*.

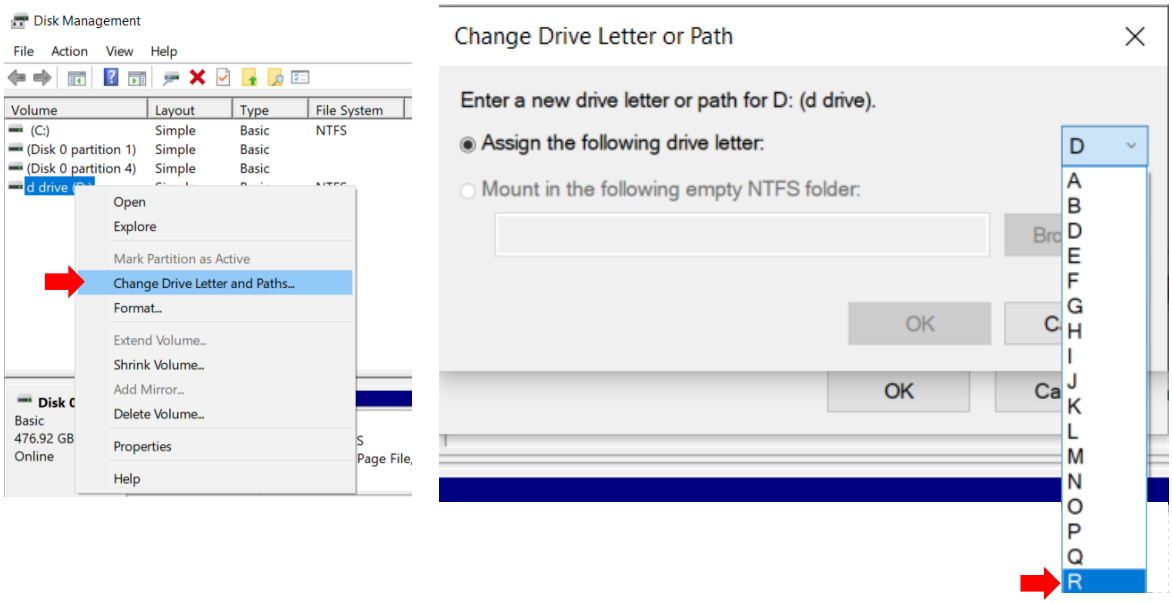


Figure 2: Create and Format Hard Disk Partitions

4. Click *Change*. From the drive drop-down menu select *R,* and then click *Yes* through all the warnings.
5. To unmount the drive open File Explorer and right-click on the drive and then select *Eject*.

4.3 CodeSourcery

CodeSourcery from Mentor Graphics is the standard for embedded Linux cross-compiler/linker tool sets (*toolchain*). CodeSourcery for Windows is required to compile applications for the ARM processor on SightLine boards under Windows. Currently the sample applications/libraries are built with one of the following versions:

- arm-2009q1-203-arm-none-linux-gnueabi (most common)
- 2010.09-50 version

When installing CodeSourcery select the *Minimal* option. This installs all components other than the IDE. All other installation settings should be left at default.

To verify the correct version of CodeSourcery has been installed use the following Powershell command:

```
(Get-Package -ProviderName Programs '*Sourcery G++ Lite for ARM GNU/Linux*').version
```

This can also be verified by from the Control Panel in Windows. Navigate to *System* and search *Apps and Features*. CodeSourcery should be listed as *Sourcery G++ Lite for ARM GNU/Linux*. Select it to see the version number.

4.4 Code Composer Studio

Use Code Composer Studio v5.5 (CCS) as the IDE to develop and debug the application. Install CCS with default options. When prompted for *Processor Support*, check *Select All*. When CCS is launched for the first time, it will prompt the user to select a license. Select *FREE LICENSE* at the prompt.

For Windows 7/8, use Code Composer Studio v5.4.

4.5 Windows Environment Variables

The System Environment Variables must be defined on the host PC.

1. In the *Start* menu search field, enter *Edit the system environment variables* to open the *Advanced* tab of *System Properties* dialog window.
2. Click *Environment Variables.*
3. In the *System variables* dialog window click *New*.
4. Add the following variable names and values shown in Table 3.
5. Click *OK* after each entry and repeat.

Variable names are case sensitive.

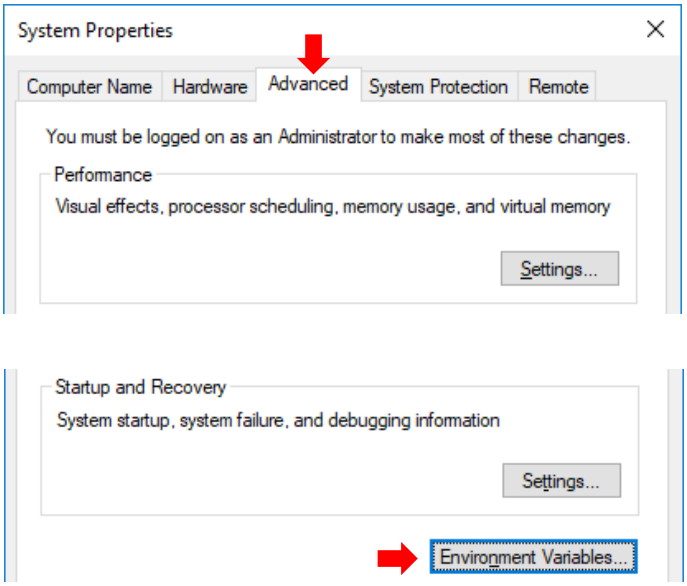


Figure 3: Define System Environment Variables

Table 3: System Variables

Variable Name:	Variable Value:	Notes:
SL_1500FS	R:\sla1500fs	1500 only
SL_1500ArtiFacts	R:\sla1500fs\root	1500 only
SL_3000FS	R:\sla3000fs	3000 only
SL_3000ArtiFacts	R:\sla3000fs\home\root	3000 only
SL_TOOL_CHAIN	C:\Program Files (x86)\CodeSourcery\Sourcery G++ Lite	
SL_TOOL_PREFIX	arm-none-linux-gnueabi-	
SL_XDC	C:\ti\xdctools_3_25_00_48	
SL_SDK	C:\SL\SDK	

4.6 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.

5 Building, Debugging and Deployment

5.1 Importing the Application into CCS

1. Open the Code Composer Studio application.
2. Select Workspace *C:\SightLine Applications\SLA-Examples-Arm ***\workspace*.
3. Select *View* » *Project Explorer*.
4. Select *File* » *Import.*
5. Expand *General*. Select *Existing Projects into Workspace* and click *Next*.
6. In the *Import Projects* dialog click *Browse* and navigate to (*C:\SightLine Applications\SLA-Examples-Arm ***\workspace*).
7. Select the desired project directory and click *Finish*.

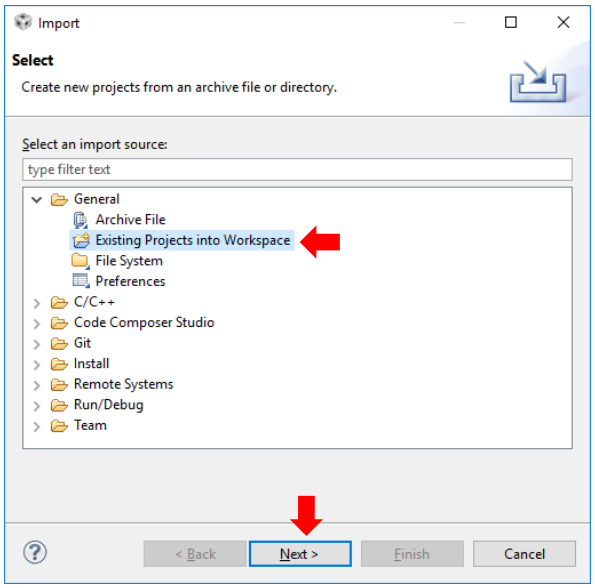


Figure 4: Importing a Project

A predefined CCS 5.x preferences file (ccs_eclipse_preference.epf) is in the workspace folder. Import this file via the Import Preferences dialog. Expand General and select Preferences.

When importing preferences ensure that Import all is selected, and no other options are checked.

5.2 Building the Application

Previous documentation for software 3.4.x and below reference booting from an NFS filesystem. As of version 3.5.x this is no longer required. Booting from the NAND filesystem on the board is sufficient for developing and debugging applications.

IMPORTANT: Connect the SightLine hardware and PC Ethernet interfaces to a network switch. Disable all other network adapters on the PC (wireless, VPN, etc.). Interfaces connected to other networks may interfere with the following steps.

The *.cproject* file that comes with each example project contains a post-build command to automatically copy the application to the OEM board for both release and debug configurations. This does not need to be changed for normal development. The post-build command calls *postBuild.bat* that is found in *SLA-Examples-ARM 3.05.00\workspace\0_ArtiFacts*. The post-build command is automatically loaded into the editor GUI. It can be found from the *Build Steps* tab » *C/C++ Build* » *Settings*.

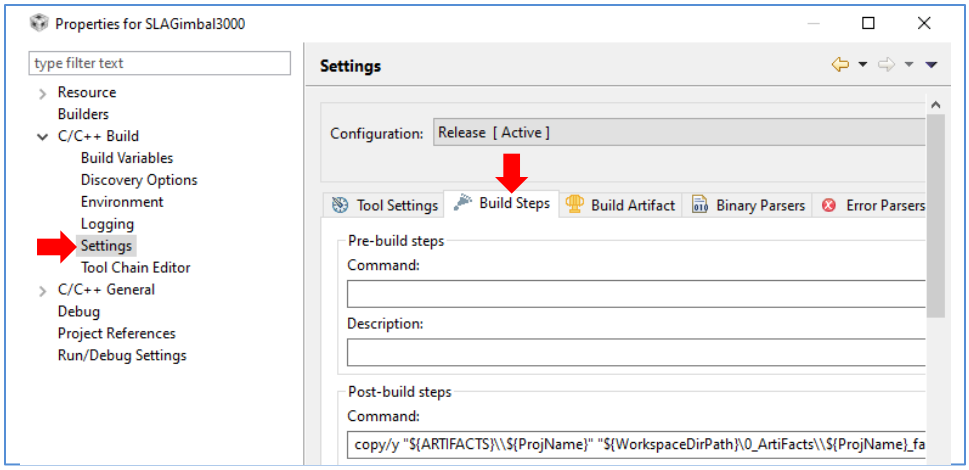


Figure 5: Post-Build Steps for Release

The build variable *${SL_**00_IP}* must be created and set to the IP address of the board for the post-build command to successfully copy the application over to the board.

1. In the *C/C++ Build* menu select *Build Variables*.
2. Ensure *Configuration* is set to *All Configurations.*
3. Click *Add*.
4. Input the variable name as *SL_**00_IP* and the value as the IP address of the board for all configurations as shown in Figure 6.

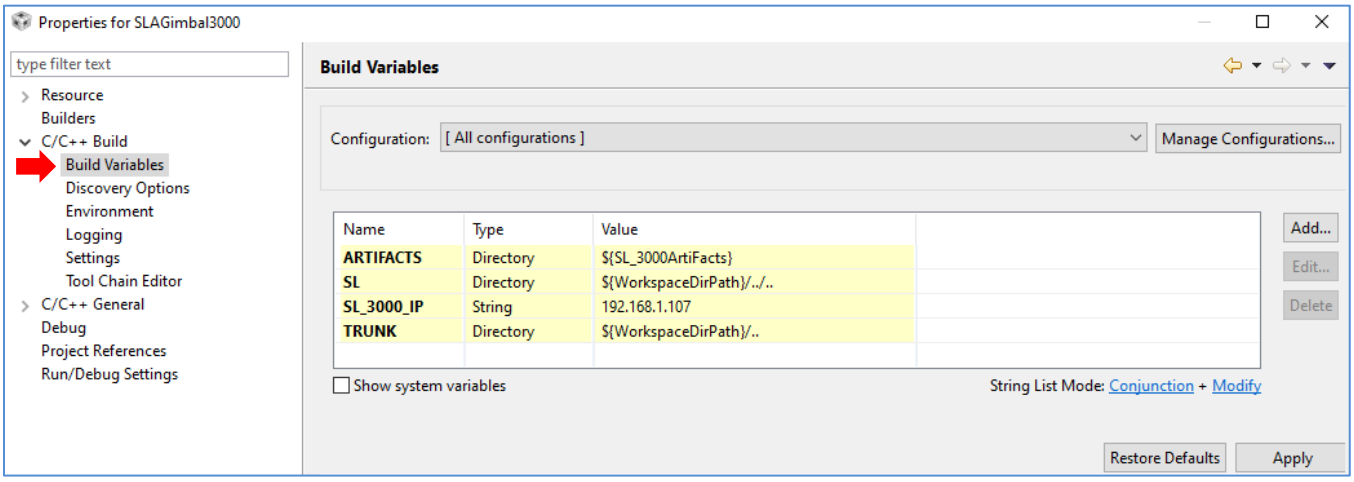


Figure 6: Build Variables with OEM Board IP Address

To prevent the application from being copied to the board automatically after building in CCS, remove the command from the post-build step and save it to another location (e.g., in the post-build description) first.

To build the project:

1. Right click on a project.
2. Select *Build Project*.

3. Test the application. From the embedded Linux console (the 1500 prompt is given as an example):

```
root@sla1500:~#    ./YourApp_Debug
```

or

```
root@sla1500:~#    ./YourApp
```

The prompt may look different on older models and on other board. For example, older versions of the 1500 may have DM-37x as the prompt.

4. For debugging see the Appendix B.
5. The Overlay DLL project creates a library instead of an executable. To enable the library, see the Overlay DLL - Build and Enable section.

Since an executable project depends on another static library project, editing a file in the static library project may not cause the executable to be rebuilt. It is safer to manually build each project. The file may need to be edited in the executable project to correctly rebuild the executable.

5.2.1 1500-OEM and 3000-OEM: Access Files on NAND through SCP Protocol

The filesystem on SightLine hardware also allows files to be copied to and from the board using the SCP protocol. Free GUI SCP clients such as WinSCP are available for Windows. The *scp* command is included in many common Linux distributions.

See EAN-Using-WinSCP for additional information.

5.3 Deploying the Application

5.3.1 1500-OEM Startup Script

When the 1500-OEM starts it executes */etc/rc.d/rc.local* at the end of initialization. If the application needs to start automatically edit the script file accordingly.

Edit the *rc.local* script on the NAND filesystem on the board for deployment. It may be necessary to insert a 10-20 second *sleep* before executing the custom application. This ensures that VideoTrack is running when the application is launched.

Near the end of this script there is an example of delaying and then starting *rtspMain*. This can be copied and modified to start the application with an appropriate delay.

Building a release version of the application for deployment is recommended.

5.3.2 3000-OEM Startup Script

When the 3000-OEM starts it executes */home/root/sla3000_init.sh* at the end of initialization. If the application needs to start automatically edit the script file accordingly.

Edit the script on the NAND filesystem on the board for deployment. It may be necessary to insert a 10-20 second *sleep* before executing the custom application. This allows time for VideoTrack to start before the application is launched.

The SLA3000 file system must be changed to writeable before editing the script.

```
mount -w -o remount /       # To make the filesystem writable
```

Near the end of this script there is an example of delaying then starting *rtspMain*. This can be copied and modified to start the application with an appropriate delay.

Building a release version of the application for deployment is recommended.

5.4 Overlay DLL - Build and Enable

Follow the steps in the Building the Application section to build the *SLAOverlayDLL3000* project.

1. Select the release build to build the file *R:\sla3000fs\home\root\\x_SLOverlay_DLL_Release.so*.
2. Reboot the system.
3. Connect to the system with Panel Plus.
4. 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*.
5. From the main menu » *Parameters* » *Save to Board,* and then main menu » *Reset Board* to persist the settings through subsequent restarts.
6. Verify a red X is drawn over the video.

6 Troubleshooting

6.1 Code Composer Studio Troubleshooting

6.1.1 Misleading Link Errors

Occasionally CCS reports numerous linker errors that do not make sense. In many cases this can be resolved by deleting all the projects from the *Project Explorer* list and then reimporting them.

6.1.2 False Compiling Errors

Occasionally CCS reports syntax and semantic errors that do not actually keep the project from building.

To eliminate these errors:

1. Go to the *Window* drop-down menu in CCS.
2. Select *Preferences*.
3. Expand the *C/C++* menu.
4. Check and then uncheck *Syntax and Semantic Errors*.

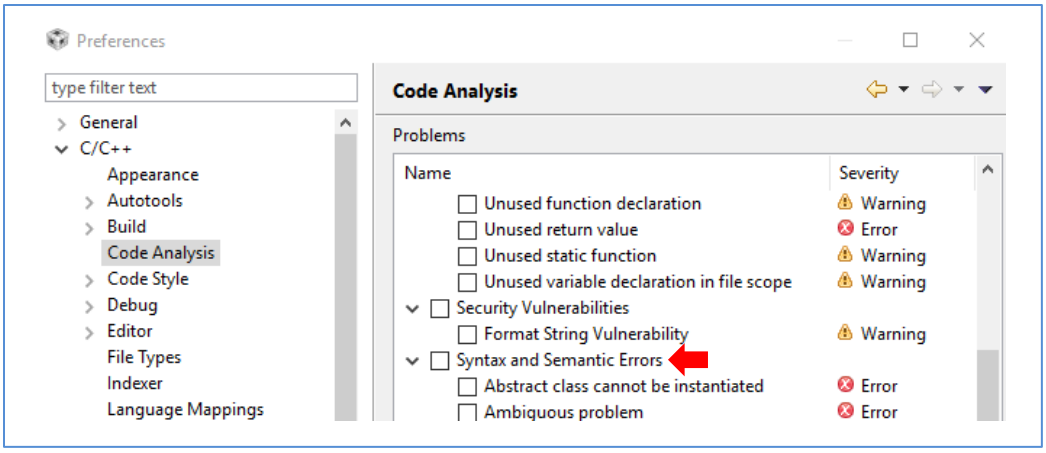


Figure 7: Syntax and Semantic Errors Checkbox

6.2 General Troubleshooting

Applications may not function as expected when deployed to SightLine hardware.

- ✓ Check the param*.txt file in the */root* (1500-OEM) or */home/*root (3000-OEM) directory. This file stores VideoTrack configuration settings.
- ✓ Disable silent mode before running applications (setenv silent from the U-Boot prompt).
- ✓ For long command line commands, create a shell (.sh) script that executes them for ease of use, e.g., starting GDB server on target during debugging, etc.
- ✓ Verify that the license file on the SightLine hardware is valid.
- ✓ Make sure all file permissions are set properly (chown, chmod a+x, etc.).

6.3 Issues with Code Composer Studio and Windows 10

The Code Composer Studio warning about incompatibility with Windows 10 can safely be ignored.

Figure 8: Code Composer Incompatibility Warning

6.4 Questions and Additional Support

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

Appendix B - Using GDB Debugging in CCS

1. From the *Window* drop-down menu in Code Composer Studio select *Preferences*.
2. Expand the *General* menu in the sidebar tree.
3. Check *CDT GDB debugging* in the *Capabilities* list.
4. Click *OK* to proceed.

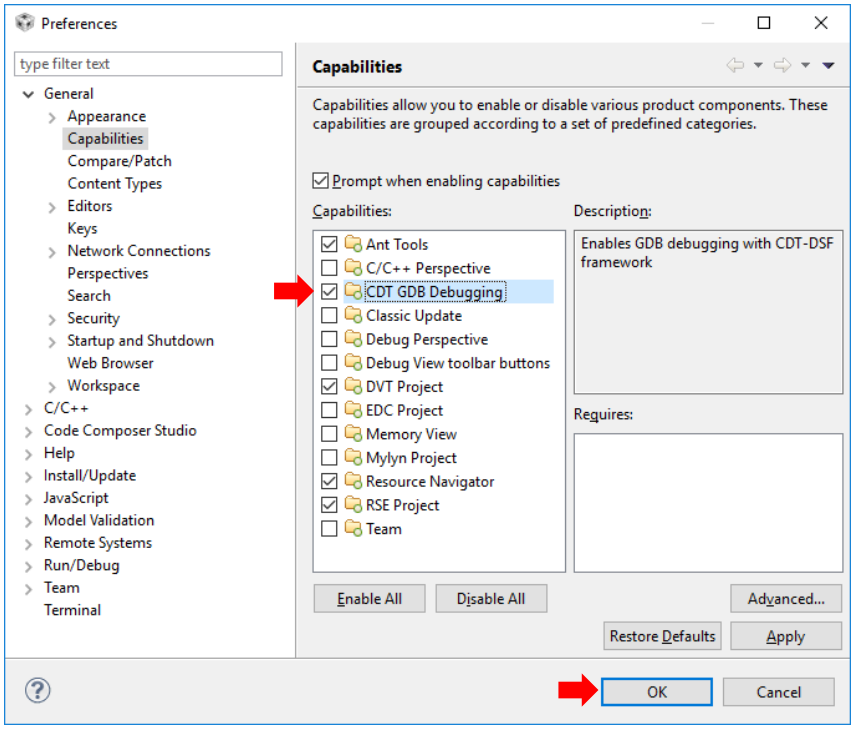


Figure B1 : Select CDT GDB Debugging

SLAGimbal1500 is used in this example.

5. Right click on *SLAGimbal1500* and select *Debug As* » *Debug Configurations*.
6. Right click on *C/C++ Remote Application*, select *New*.
7. Configure as shown in Figure B2.

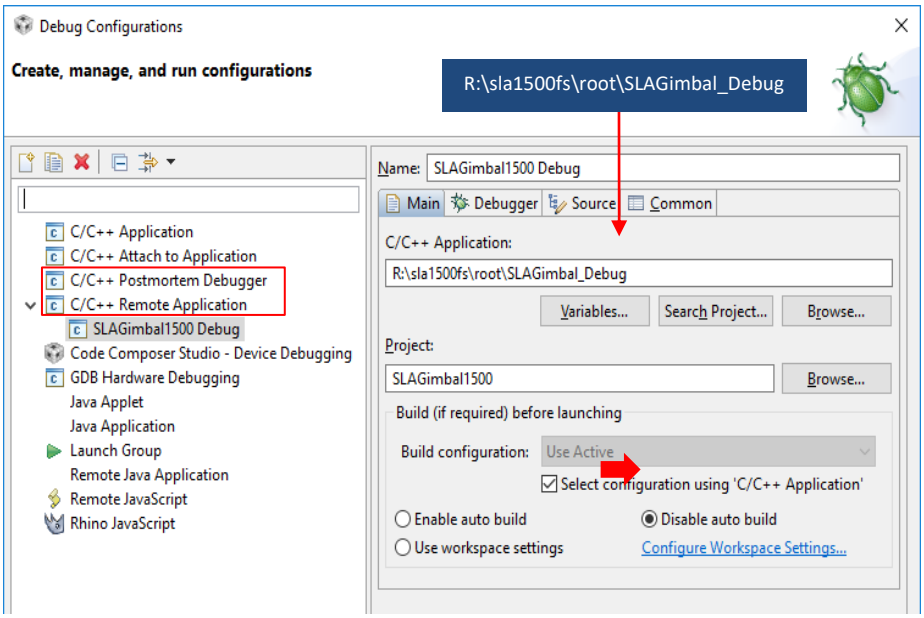


Figure B2: Create, Manage, and Run Configuration

8. Select the *Debugger* tab and copy the configuration as shown in Figure B3.

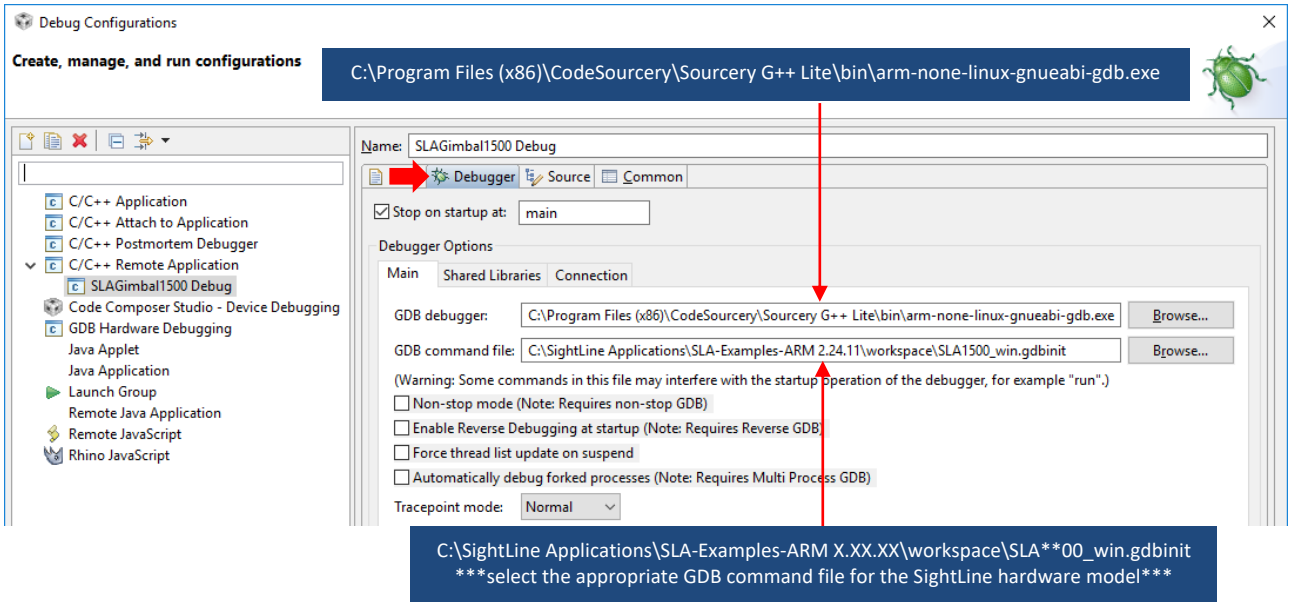


Figure B3: Debugger Tab - Copy Configuration

9. Under *Debugger Options* select the *Connection* tab and enter the IP address of the SightLine hardware. To determine the IP address, enter the `ifconfig` command at the embedded Linux console.

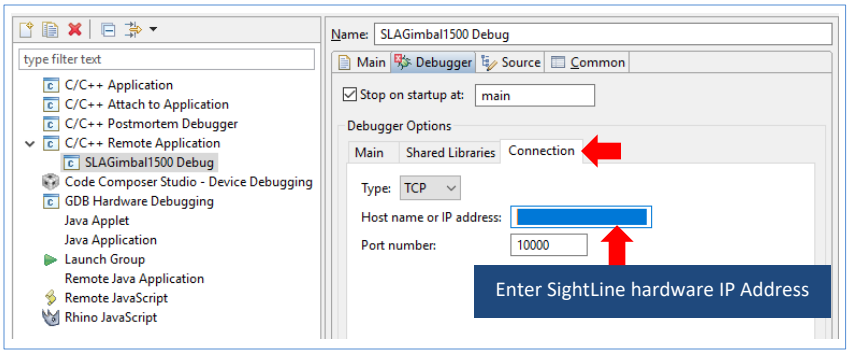


Figure B4: Enter IP Address of SightLine Hardware

10. Start the GDB server on the SightLine hardware:

```
root@sla1500:~#  gdbserver :10000  SLAGimbal_Debug
```

11. Click *Debug* in the *Debug Configurations* dialog to start the debugging process.

When the execution is stopped at a breakpoint, CCS may not find the correct thread in the Debug window (usually at the top left corner). If this occurs, selecting the correct thread in the Debug window (which is in Suspended mode) will enable the Resume and Step functions.

Appendix C - ToDo Message

If a *ToDo* message is displayed, additional steps are required to complete the development environment update. The instructions may be different from in each release. It is not necessary to perform the indicated steps more than once for a given firmware version.

Example *ToDo* message:

```
ToDo: *** run the following commands on the 3000-OEM ***
On the serial Terminal, hold Shift+S to break into u-boot, then power cycle 3000.
At u-boot prompt (SLA3000#)
set NFS server by: set serverip xx.xx.xx.xx
start NFS boot: run nfsboot
When the system has finished booting the Linux Kernel
Logon to 3000 (username=root, no password).
mount -w -o remount /   # To make the filesystem writable.
/etc/init.d/38xx-demo   # To initialize graphics libraries.
```

Enable SMBv1 Support

1. Use the WinKey(⊞)+R to open a *Run* prompt.
2. Enter *appwiz.cpl* to open the Programs and Features control panel app.
3. Select *Turn Windows features on or off* from the sidebar.
4. Expand *SMB 1.0/CIFS File Sharing Support*.
5. Check *SMB 1.0/CIFS Client* and *SMB 1.0/CIFS Automatic Removal*.

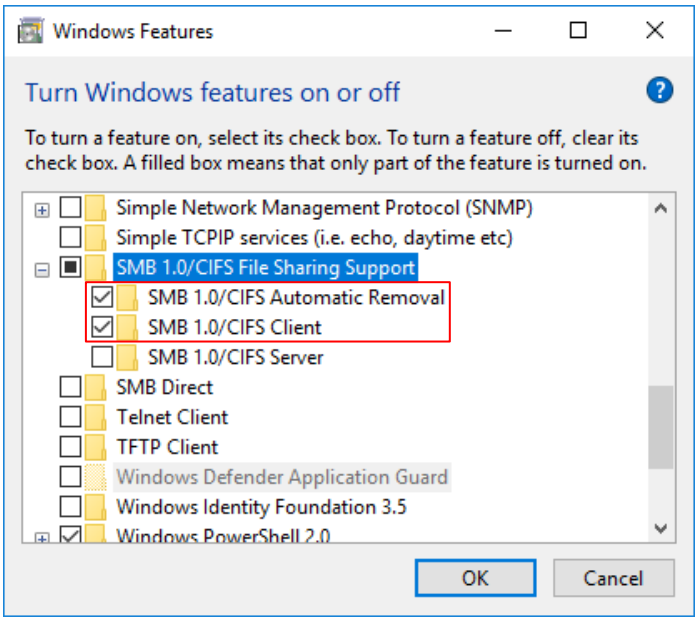


Figure D1: Enabling SMBv1 Support

Appendix D - Configuring Startup Scripts

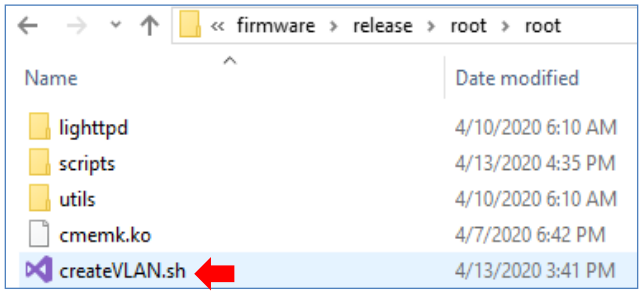
Custom scripts can be loaded to the target OEM hardware. This section explains how to create, edit, and call up startup scripts.

Example:

1. Create a new script file. In the scripts folder go to:

   \SLA-1500 Upgrade Utility\3.01.01\firmware\release\root\root\scripts\createVLAN.sh.

2. Double click on the *createVLAN.sh* script file.



3. Edit script contents.

```
1  ##
2  ## Add a VLAN
3  ##
4  vconfig add eth0 5
5  ifconfig eth0.5 192.168.42.100 netmask 255.255.255.0 broadcast 192.168.42.255 up
```

4. Use the SightLine Upgrade Utility to upgrade the OEM. The new script is now on the target hardware and ready to test.

5. Use Tera Term to establish an SSH session to the target OEM and execute the script.

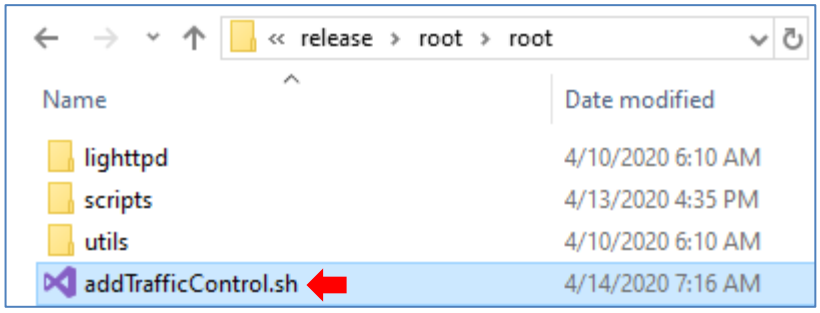
```
root@sla1500:~# sh createVLAN.sh
root@sla1500:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP100> mtu 1500 qdisc pfifo_fast qlen 100
0
    link/ether 0e:00:a0:28:01:37 brd ff:ff:ff:ff:ff:ff
    inet 169.254.1.180/16 brd 169.254.1.255 scope global eth0
3: teql0: <NOARP> mtu 1500 qdisc noop qlen 100
    link/void
4: eth0.5@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
    link/ether 0e:00:a0:28:01:37 brd ff:ff:ff:ff:ff:ff
    inet 192.168.42.100/24 brd 192.168.42.255 scope global eth0.5
root@sla1500:~#
```

6. Call scripts at bootup. The primary scripts for SightLine OEM platforms are as follows:
   - 1500-OEM /etc/rc.d/**rc.local**
   - 3000-OEM /root/home/root/**sla3000_init.sh**
   - 4000-OEM /root/home/slroot/sl/scripts/**sla_init.sh**
7. After manually verifying that the custom script works, edit the startup script to call those custom scripts at bootup.

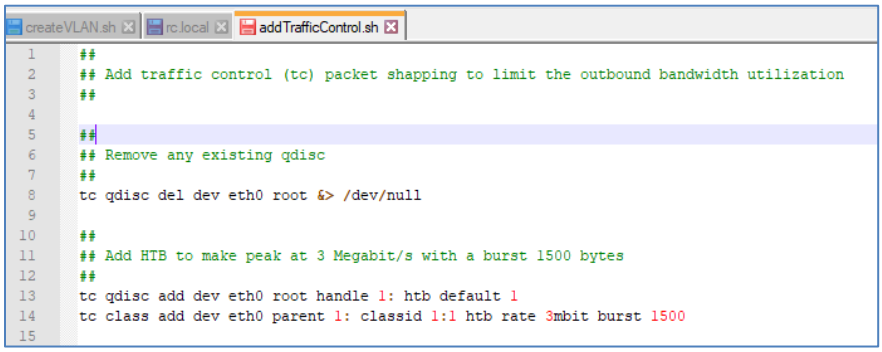
```
#ifup eth0; exit 0   ## Not to start VideoTrack.
./createVLAN.sh

if [ -f VideoTrack1500 ]; then
    if [ -f slStabTrackOMAP.out ]; then
        #./VideoTrack1500 -S0 0 -O -CAPTURE sony 720p &
```

When the firmware is updated the startup script will automatically call the custom script.

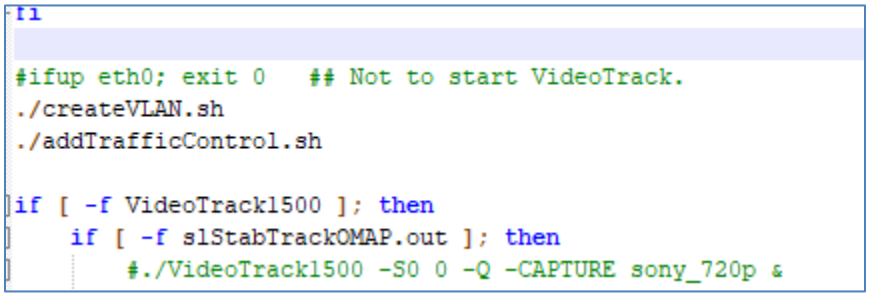
8. To add an additional script:
   a. Double click on the *addTrafficContol.sh* file.



b. Edit the script.



c. Call new script from startup script (*rc.local*).



d. To apply the script, update the firmware of the target OEM.