Six15 HUD

Windows Software Development Kit

Version 2.7.0

12/07/2021

Document Number: 555ES0003-10

Table of Contents

List of Tables

1. Six15 HUD Product

This document is intended for developers creating Heads Up Display (HUD) applications using the Six15 HUD. It communicates all possible inputs and outputs of the Six15 HUD service interface.

1.1 Introduction

This guide describes how a developer can create applications for a Windows Device (the source system) to control the HUD (the target system).

This guide specifies the interface requirements the participating systems must meet. It describes the concept of operations for the interface and defines the message structure and protocols used by the development library.

2. Software Development Kit Contents

2.1 What is Included

The development kit includes the HUD hardware, Windows HUD Service, Documentation and Source Code.

2.2 Six15 HUD Service

The Six15 HUD Service is a standalone service that is installed on Windows devices to provide basic functionality for your HUD device and provide the abstracted interface for programmers to develop custom applications for the HUD device.

See document “555ES0012-03 WINDOWS SERVICE USER GUIDE” for installation instructions.

2.2.1 Service Provided Functionality

The HUD service provides basic functionality such as screen capture and Inertial Measurement Unit (IMU) interface. It also provides the ability to query the HUD’s information and update the HUD’s firmware.

2.2.2 Six15 HUD Service Installation

The HUD service installation is provided via an APK.

Installing can be done with the filesystem tool.

2.3 Development Workstation Setup

Starting development using the Six15 HUD requires that your workstation be setup with Visual Studio (Community Edition is supported). Visual Studio is a freely available development environment that runs on Windows. Using the Visual Studio user guide, install Visual Studio on your workstation.

https://visualstudio.microsoft.com/

You will also need to have the Six15 HUD Service installed on the Windows device where your application will be installed. It is the responsibility of the application to check for the installed service prior to trying to bind to the service via the application. Six15 HUD service installation can be performed on a device as described in Windows Service User Guide. All examples and code have been written using Visual Studio 2019 Professional Edition.

3. Six15 HUD Service Interface

3.1 Assumptions

The device controlling the HUD must run Windows 10 or higher and will control the HUD via the commands made available by the HUD service. Commands will be issued using the service binding between your application and the HUD service which is used to provide an abstraction layer to the hardware. Microsoft .NET is currently supported with examples in C# only.

3.2 Constraints

The following Device constraints are imposed to ensure compatibility and interface reliability.

- Device to run Windows 10.
- Device must support USB HighSpeed.

3.3 Service API

Six15's Windows Service makes interfacing with the HUD easier by abstracting the details of the protocols and physical connections between the host system and the device while also providing some features and capabilities without the need for developing custom applications. Using the service interface provided by .NET remoting interface in your application, commands can be issued to the HUD and information from the HUD can be received and processed.

All commands are sent in an asynchronous fashion unless otherwise noted. All responses from the HUD will be in the form of a response packet containing the command and command ID that it is responding to. Data will be returned as a JSON object for easier parsing by the application.

3.3.1 HUD Service Class

The HUD Service class is a utility class that implements the remoting interface and adds .NET specific calls for ease of use such as sending a bitmap to the HUD. This class will reformat the image to adjust to the HUD display size and send frame to the HUD.

```
HudServiceLib.HudInterface

Example Implementation:

HudInterface hudService = new HudInterface();
```

3.3.2 HUD States

Below is the enumeration of the HUD states. Currently there are three states of the HUD:

1. Not Detected - The HUD is not connected to the PC.
2. Application - The HUD is ready to communicate with an application for sending and receiving data
3. Bootloader - The HUD is in Firmware maintenance mode.

```
public enum HUD_STATE { NotDetected = 0, Application, Bootloader };
```

3.3.3 HUD Remoting Interface

The HUD Remoting Interface to the HUD service is shown below and can be found in the distributed library, “SDK HudServiceLib.IHudInterface”. Each function is described in Section 3.4.

```
.    public interface IHudInterface
    {
        /* properties */
        Size DisplaySize { get; }
        int DisplayBrightness { get; }
        bool DisplayOn { get; }
        UInt32 HUDFirmwareVersion { get; }
        UInt32 ServiceVersion { get; }
        HUD_STATE HudState { get; }
        bool isHudConnected { get; }

        /* system functions */
        bool doHudSoftReset();

        /* display image functions */
        bool clearHudDisplay();
        bool sendBufferToHud(byte[] buffer);

        /* hud events */
        event NotifyConnectionStateCallback NotifyConnectionState;
    }
```

3.3.4 HUD NotifyConnectionState Callback

The HUD Notify Connection State Callback is used as a remoting callback so that your application can be notified when the HUD state changes. This is only required if you are writing your own implementation of the remoting interface to the HUD service. When using the HUD service utility class this is handled for you and is revealed through a delegate method.

```
    public delegate void NotifyConnectionStateCallback(HUD_STATE s);

    public abstract class NotifyConnectionStateCallbackSink : MarshalByRefObject
    {
        // Called by the service to fire the call back to the client
        public void FireNotifyConnectionStateCallback(HUD_STATE s)
        {
            Console.WriteLine("Activating Connection State callback");
            OnNotifyConnectionStateCallback(s);
```

```
        }


        // Client overrides this method to receive the callback events from the service
        protected abstract void OnNotifyConnectionStateCallback(HUD_STATE s);
    }
```

3.4 HUD Service API Definitions

3.4.1 Information and Hardware Functions

The HUD service capabilities are exposed by the functions in the HUD service remoting interface. All property get functions return a result based on their native .NET return type. Functions that perform an action return a Boolean true or false to determine if the call was successful or not.

3.4.1.1 Display Size

Function Prototype: `Size DisplaySize`

Parameters: None

Return: `System.Drawing.Size`

Description:

Used to retrieve the HUD display size.

3.4.1.2 Display Brightness

Function Prototype: `int DisplayBrightness`

Parameters: None

Return: Integer

Description:

Used to retrieve the display brightness setting. Valid responses are one(1)(min brightness) – four(4)(max brightness).

3.4.1.3 Set Display Brightness

Function Prototype: `int setBrightnessLevel(int lvl)`

Parameters: Integer

Return: Integer

Description:

Used to set the display brightness by passing the brightness level desired one(1)(min brightness) – four(4)(max brightness). Brightness levels sent that are less than the range minimum will be set to one(1), and levels greater than the maximum of the range will be set to four(4). The call returns a -1 on a failure while setting the display brightness or else it returns the brightness level.

3.4.1.4 Display On

Function Prototype: `bool DisplayOn`

Parameters: None

Return: boolean

Description:

Used to check if the display is on.

3.4.1.5 HUD Firmware Version

Function Prototype: `UInt32 HUDFirmwareVersion`

Parameters: None

Return: unsigned integer

Description:

Returns the HUD firmware version as an unsigned integer with the following format.

```
[31:24] Major version
[23:16] Minor version
[15:8]  Patch version
[7:0]  release candidate
```

3.4.1.6 HUD Service Version

Function Prototype: `UInt32 ServiceVersion`

Parameters: None

Return: unsigned integer

Description:

Returns the HUD Service version as an unsigned integer with the following format.

```
[31:24] Major version
[23:16] Minor version
[15:8]  Patch version
[7:0]  release candidate
```

3.4.1.7 Get HUD State

Function Prototype: `HUD_STATE HudState`

Parameters: None

Return: HUD_STATE Enumeration

Description:

Used to check if a HUD is connected to the Windows device and what state the device is in. The return values are one of the HUD State enumeration values.

3.4.1.8 Get HUD Connected

Function Prototype: `bool isHudConnected`

Parameters: None

Return: boolean

Description:

Used to check if a HUD is connected to the Windows device. Returns true when a HUD is connected to the Windows USB port.

3.4.1.9 HUD Soft Reset

Function Prototype: `bool doHudSoftReset()`

Parameters: None

Return: boolean

Description:

Perform a soft reset of the device and returns true if the call completes successfully.

3.4.1.10 Clear HUD Display

Function Prototype: `bool clearHudDisplay()`

Parameters: None

Return: boolean

Description:

Clears the HUD display and returns true if the call completes successfully.

3.4.1.11 Send Buffer to HUD

Function Prototype: `bool sendBufferToHud(byte[] buffer)`

Parameters: `byte[]`

Return: boolean

Description:

Send display data to the HUD display. Display data must be a byte array containing a 640x400 image that is JPEG Baseline compressed. Returns true if the frame is sent successfully to the HUD.

3.4.1.12 Notify Connection State

Function Prototype: `event NotifyConnectionStateCallback NotifyConnectionState`

Parameters: None

Return: None

Description:

Connection state .NET event for monitoring state changes of the HUD.

3.5 IMU Functions

On the HUD there is located an IMU that can asynchronously stream sensor data to the connected device in the form of JSON data. This sensor returns accelerometer, gyroscope, magnetometer, quaternion, and temperature data.

3.5.1.1 IMU Data

Function Prototype: `delegate void HudImuDataReceived(string json);`

Parameters: None

Return: None

Description:

IMU sensor data is sent to the HUD at a fixed rate of 33ms when IMU streaming is enabled. This data is sent via the HUD Service callback `HudImuDataReceived` function. Data is in JSON format defined below. All floats are carried out to a maximum of three decimal places except for temperature which is two decimal places.

Accelerometer data is in the form of a float vector consisting of X, Y, Z values signified by the A element.

Gyroscope data is in the form of an array of floats X, Y, Z values signified by the G element.

Magnetometer data is in the form of an array of floats X, Y, Z values signified by the M element. A Quaternion vector is provided signified by the Q element and consists of X, Y, Z, W values.

Temperature is in the format of degrees Celsius and is signified by the T element in the data structure and consists of a single float value.

ASYNC JSON Data:

```
{
	"A":[%.3f,%.3f,%.3f],
	"G":[%.3f,%.3f,%.3f],
	"M":[%.3f,%.3f,%.3f],
	"Q":[%.3f,%.3f,%.3f,%.3f],
	"T":%.2f
}
```

3.5.1.2 IMU Status

Property Prototype: `bool ImuEnabled`

Parameters: None

Return: bool

Description:

Return the status of the IMU data streaming, where true is on and false is off.

3.5.1.3 Start IMU

Property Prototype: `bool ImuEnabled`

Parameters: bool

Return: NONE

Description:

Set the ImuEnabled Property to true to start the IMU data Stream, which is received by the `HudImuDataReceived` delegate function.

3.5.1.4 Stop IMU

Function Prototype: `bool ImuEnabled`

Parameters: bool

Return: NONE

Description:

Set the ImuEnabled Property to false to stop the IMU data Stream.

3.5.1.5 Start User IMU Cal

Function Prototype: `bool startUserImuCal(int imuSensor);`

Parameters: bool

Return: startImuCal(ImuCalType.CAL_USER, ImuCalSensor.forValue((byte)imuSensor));

Description:

Starts calibration functions for system corresponding to the byte value.

3.5.1.6 Get User IMU Cal

Function Prototype: `bool getUserImuCal(int imuSensor)`

Parameters: bool

Return: getImuCal(ImuCalType.CAL_USER, ImuCalSensor.forValue((byte)imuSensor)

Description:

Get IMU calibration.

4. HUD Service Class

4.1 Information and Hardware Functions

The HUD service utility class exposes all the capabilities of the .NET remoting interface described in section 3 without the developer having to worry about managing the remote connection. It also provides higher level functions providing more options for sending image data to the HUD and auto formatting image data to work with the HUD.

4.1.1 Auto Crop

Function Prototype: `bool AutoCropEnabled`

Parameters: None

Return: boolean

Description:

Get or set auto crop, which when enabled will automatically crop the image down to the correct size for the HUD display. By default this is enabled.

4.1.2 Auto Center

Function Prototype: `bool AutoCenterEnabled`

Parameters: None

Return: boolean

Description:

Get or set auto center functionality, which when enabled will automatically center the image in the display if the image is smaller than the HUD display. By default this is enabled.

4.1.3 Send Bitmap to HUD

Function Prototype: `bool sendBitmapToHud(Bitmap bmp)`

Parameters: Bitmap

Return: Boolean

Description:

Send .NET Bitmap object to the HUD display. Saves the user from having to convert this to a compressed JPEG byte array. Returns true when the bitmap is succssfully sent to the display.

4.1.4 Send JPEG from Resource

Function Prototype: `bool sendJpgResourceToHud(string imagename)`

Parameters: String

Return: Boolean

Description:

Send JPEGs saved as embedded resources to the HUD display. User must provide the resource name as a string and the function will return true when the JPEG is sent to the display.

4.1.5 Send JPEG File

Function Prototype: `bool sendJpegToHud(string filePath)`

Parameters: String

Return: Boolean

Description:

Send JPEG files from the file system to the HUD display. User must provide the file path as a string and the function will return true when the file is sent to the display.

4.1.6 Connection Changed Delegate

Function Prototype: `delegate void ConnectionChanged(HUD_STATE state)`

Property: `ConnectionChanged connectionChanged;`

Parameters: None

Return: None

Description:

Using the connection changed delegate connectionChanged property, a user can set a function to respond to state changes of the HUD.

5. Getting Started

5.1 Getting Started Overview

To get started with your first Windows project utilizing the Six15 HUD, you will need to create a new project or use an existing project and add the HUD Library to provide the interfaces and classes required to communicate with the HUD Service. Once the SDK library is added to the project and the Windows HUD service is installed, your application will be able to communicate with the HUD device.

5.2 Creating a new .NET Windows application

In this section we will walk through setting up a .NET Windows application project and adding the Six15 library. Install Visual Studio community edition which is free of charge from:

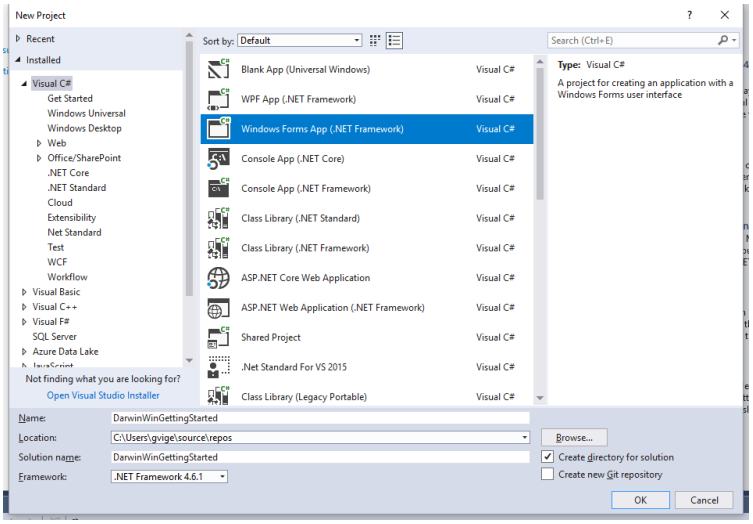
https://visualstudio.microsoft.com/downloads/

Information on installing Visual Studio can be found here:

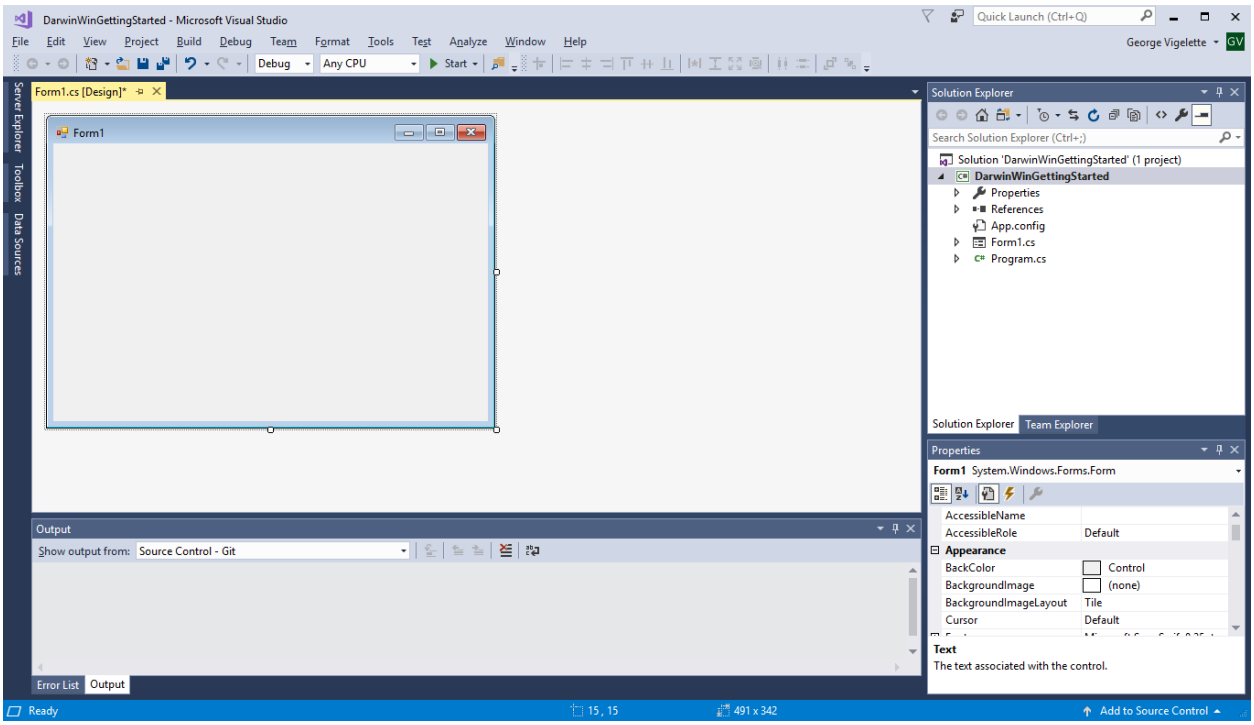
https://docs.microsoft.com/en-us/visualstudio/install/install-visual-studio?view=vs-2017

5.2.1 Create new Visual Studio project

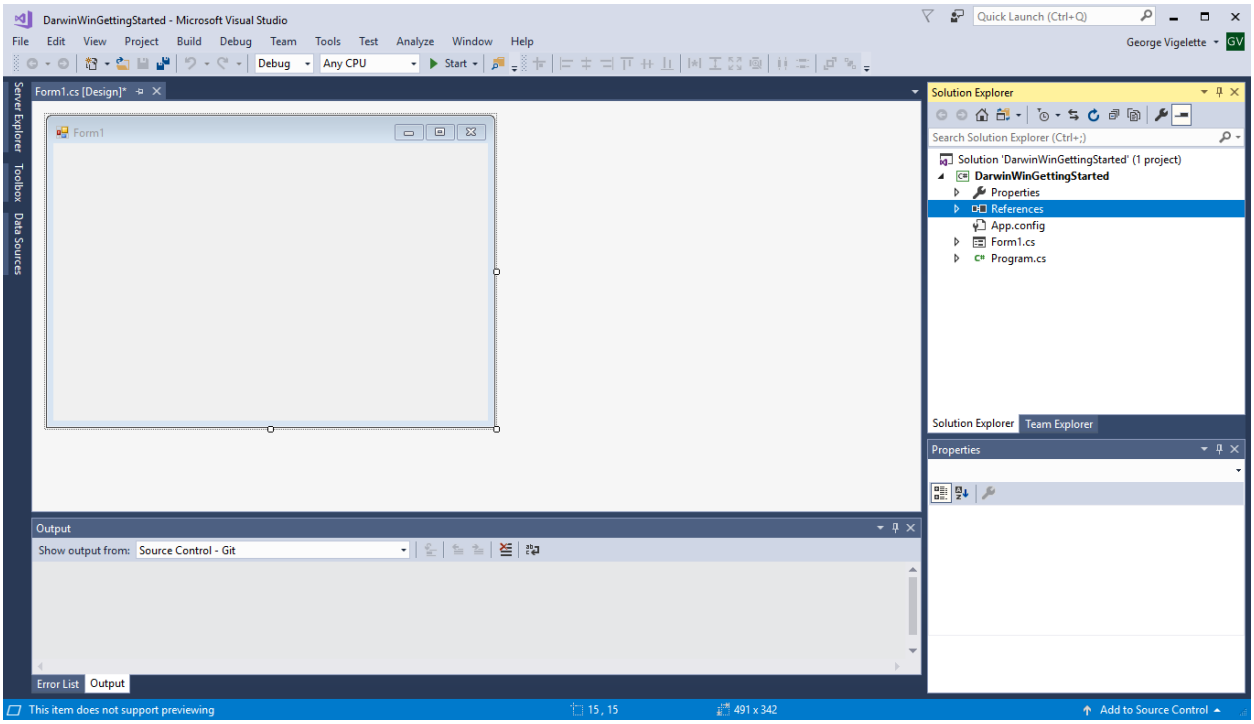
Launch Visual Studio and select File->New->Project when presented with the New Project dialog box. Then select Visual C# and Windows Form APP as shown below. Provide a name for your application and click OK.

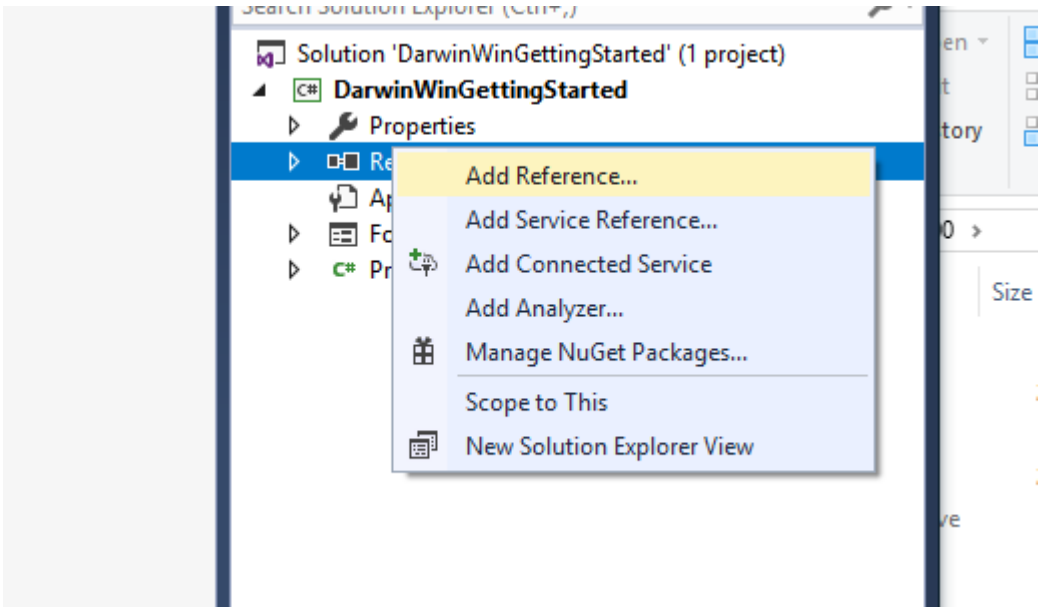


You will be presented with a blank form as seen below.

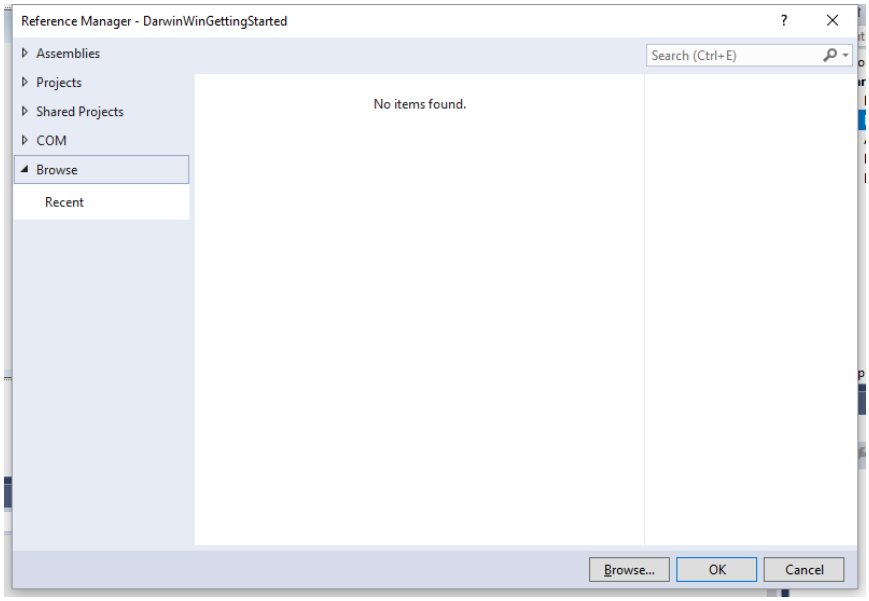


Add the Six15 HUD SDK library as a reference to the project. This can be done by right clicking on References and select Add Reference.

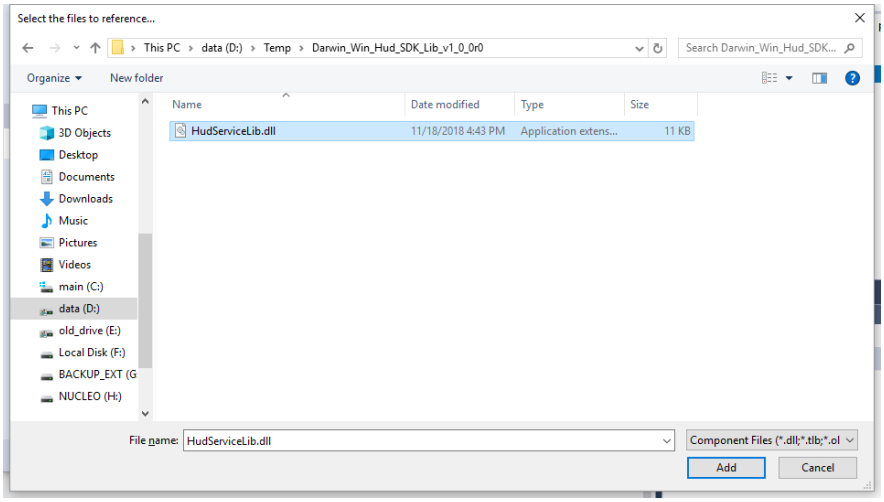


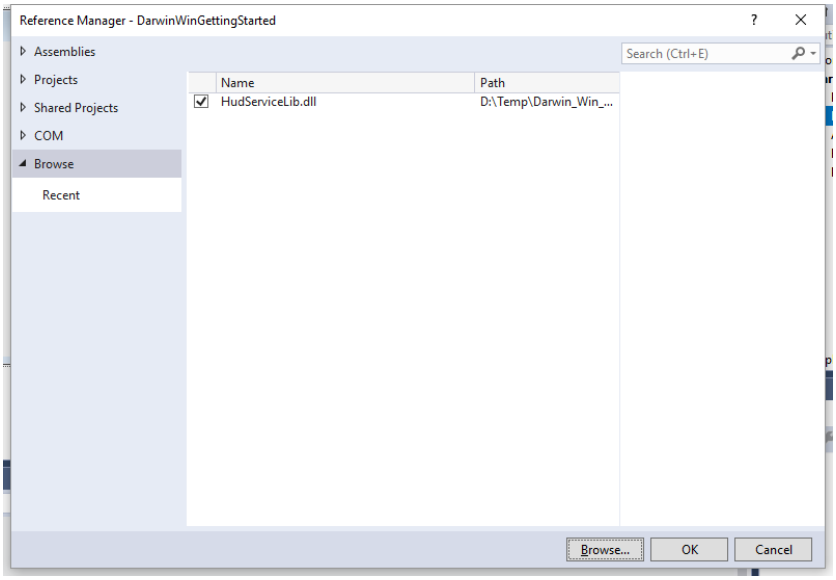


Select Browse from the reference manager and click the Browse button.

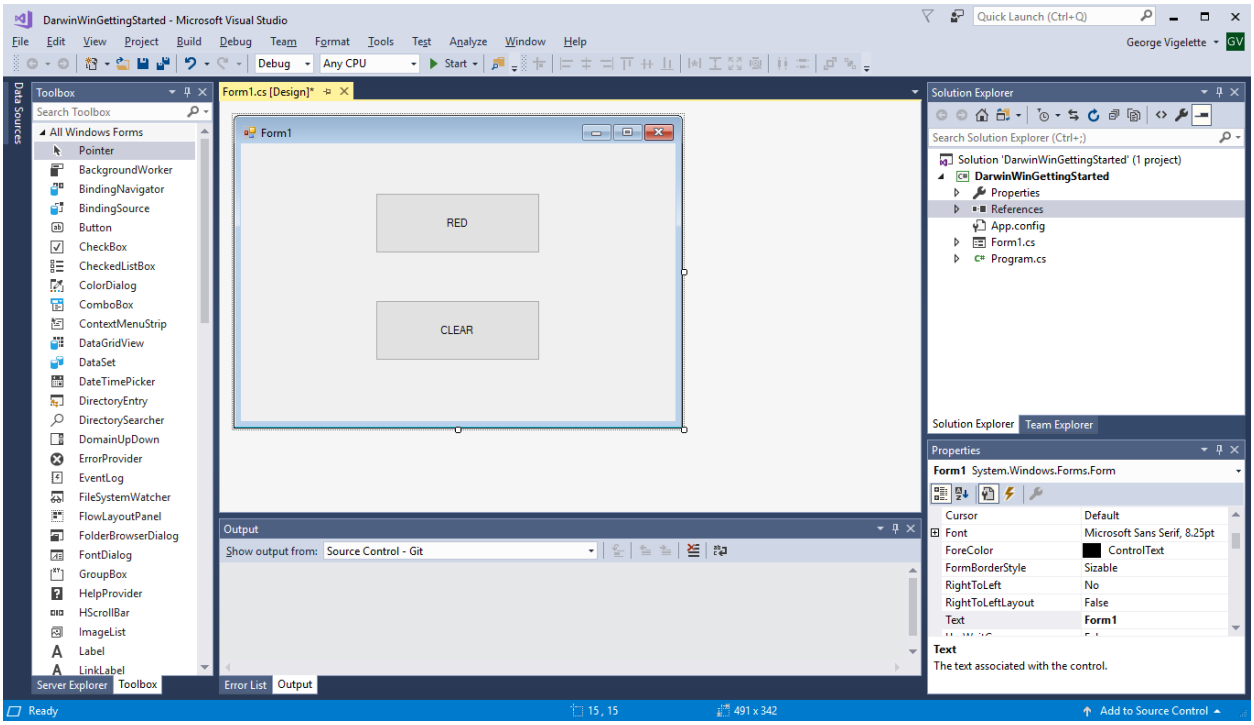


Navigate to the HUD SDK Library that was extracted and select HudServiceLib.dll, then click Add followed by OK to complete.





From the Toolbox, drag two buttons onto the form and label one RED and the other CLEAR as shown below.



Double click the form to get into the form_load function and the source code of our application. We will start by adding some private global variables: one to keep track of the HUD connection state and one to interface with the HUD using the HudInterface class.

```
        private bool bHudConnected = false;
        private static HudInterface hudService;
```

You will need to add the using directive to allow the compiler to find the HudInterface class in the added library.

```
        using HudServiceLib;
```

We will add a connection change event to update our connection status Boolean

```
        void ConnectionChanged(HUD_STATE state)
        {
            bHudConnected = state == HUD_STATE.NotDetected? false:true;
        }
```

Now we will add the start-up code in our form_load function

```
        private void Form1_Load(object sender, EventArgs e)
        {

            // set initial connection state
            bHudConnected = false;

            // Create instance of HUD interface class
            hudService = new HudInterface();

            // register for HUD connection state changes
            hudService.connectionChanged = ConnectionChanged;

            try
            {
                // check connection state on load
                if (hudService.isHudConnected)
                {
                    bHudConnected = true;
                }
                else
                {
                    bHudConnected = false;
                }

            }
            catch (Exception ex)
            {
                // not connected or error
                Console.WriteLine(ex.Message);
                MessageBox.Show("NO Service Detected!");
            }
        }
```

Lets add the clear HUD code to the Clear button's onClick event. Go back to your form and double click the CLEAR button to create the button's onClick event. Then we add the code to check for a connected HUD and then send the ClearHudDisplay command using the HUD Service Class.

```
        private void button2_Click(object sender, EventArgs e)
        {
            // clear button clicked
            if (!bHudConnected)
            {
                MessageBox.Show("No HUD Connected");
                return;
            }

            hudService.clearHudDisplay();
        }
```

Finally, lets add code to the RED button's onClick event to set the display screen to red. Just like with the CLEAR button, double click on the button in your form to create the method and then add the following code.

```
        private void button1_Click(object sender, EventArgs e)
        {
            // red button clicked
            if (!bHudConnected)
            {
                MessageBox.Show("No HUD Connected");
                return;
            }

            // create a bitmap of the same size as the display
            Size dispSize = hudService.DisplaySize;
            Bitmap bmp = new Bitmap(dispSize.Width, dispSize.Height);

            // fill bitmap with red
            Graphics g = Graphics.FromImage(bmp);
            g.Clear(Color.Red);

            // send bitmap to HUD
            if (hudService.sendBitmapToHud(bmp))
            {
                MessageBox.Show("Display should be red on the HUD");

            }
            else
            {
                MessageBox.Show("Failed to send bitmap to display: " +
                hudService.LastErrorMessage);
            }
        }
```

Ensure that the HUD service is installed and connect your HUD to your Windows device. Run your application and click the buttons to change the display according to the button. RED fills the display with red while clear will remove the color.

5.3 Receiving event information from HUD Service

Setting up handlers for events generated by the HUD service allows your application to better handle use cases where the HUD is unplugged or plugged in as well as data sent from the HUD to the application without having to query the information from the HUD.

5.3.1 Delegates for handling remote messages

Below are the delegates exposed to the developer so that the developer can build handlers for handling messages from the HUD.

```
        public delegate void ConnectionChanged(HUD_STATE state);
        public delegate void HudMessageReceived(HUD_CommandBYTE cmd, ushort cmd_id, bool
status, string json);
        public delegate void HudImuDataReceived(string json);
```

5.3.2 Establish Handlers

The developer needs to implement handler functions that reflect the delegate prototype and assign the handler to the delegate in order to receive the messages from the HUD device. This is done after establishing the HUD interface object.

```
            hudService = new HudInterface();

            hudService.connectionChanged = ConnectionChanged;
            hudService.incomingMessage = MessageReceived;
            hudService.incomingImuData = imuDataReceived;
```

5.3.3 Example ConnectionChanged Handler

When the HUD service detects a change in state the HUD State will be sent to the Developers application and the application can handle this accordingly.

```
        static void ConnectionChanged(HUD_STATE state)
        {
            switch (state)
            {
                case HUD_STATE.Application:
                    Console.WriteLine("USB DEVICE CONNECTED");
                    bHudConnected = true;
                    break;
                case HUD_STATE.Bootloader:
                    Console.WriteLine("USB BOOTLOADER CONNECTED");
                    bHudConnected = false;
                    break;
                case HUD_STATE.NotDetected:
                default:
                    Console.WriteLine("NO USB DEVICE CONNECTED");
                    bHudConnected = false;
                    break;
            }
        }
```

5.3.4 Command Byte Values

A command byte enum is provide `HUD_CommandBYTE`. Below are the values associated with the tags in the enumeration for reference. These command bytes are received with the HUD Message received event and determines the type of message that is being presented to your application by the HUD device.

```
HC_STATUS  ((byte)0x00),
HC_VERSIONS ((byte)0x01),
HC_DEV_INFO ((byte)0x02),
HC_DEV_SETTINGS ((byte)0x03),
HC_VERSIONS_EXTRA ((byte)0x04),
HC_MODE_UPD ((byte)0x05),
HC_MODE_QUERY ((byte)0x0A),
HC_MODE_SRST ((byte)0x0F),
HC_DISP_SIZE ((byte)0x20),
HC_DISP_BRT ((byte)0x22),
HC_DISP_ON ((byte)0x23),
HC_DISP_INFO ((byte)0x2F),
HC_CAM_START ((byte)0x30),
HC_CAM_SNAP ((byte)0x35),
HC_CAM_STOP ((byte)0x3A),
HC_CAM_INFO ((byte)0x3F),
HC_CFG_SPEN ((byte)0x40),
HC_CFG_SPDEL ((byte)0x41),
HC_CFG_SPIMAGE ((byte)0x42),
HC_CFG_AUTRESIZE ((byte)0x43),
HC_CFG_BL ((byte)0x44),
HC_CFG_APP ((byte)0x45),
HC_CFG_OEM ((byte)0x46),
HC_CFG_SETTINGS ((byte)0x47),
HC_CFG_CUSTOM ((byte)0x49),
HC_CFG_RESET ((byte)0x4F),
HC_IMU_DATA ((byte)0x50),
HC_IMU_STATUS ((byte)0x51),
HC_IMU_START ((byte)0x52),
HC_IMU_STOP ((byte)0x5F),
HC_AUD_INFO ((byte)0x6F),
HC_DEV_METRICS ((byte)0x80),
HC_DEV_DEBUG_TERMINAL ((byte)0x81),
HC_IMAGE ((byte)0xD0),
HC_JPEG((byte)0xDE),
HC_RAW ((byte)0xDD),
HC_ERROR ((byte)0xEE),
HC_HEART_BEAT ((byte)0xFF);
```

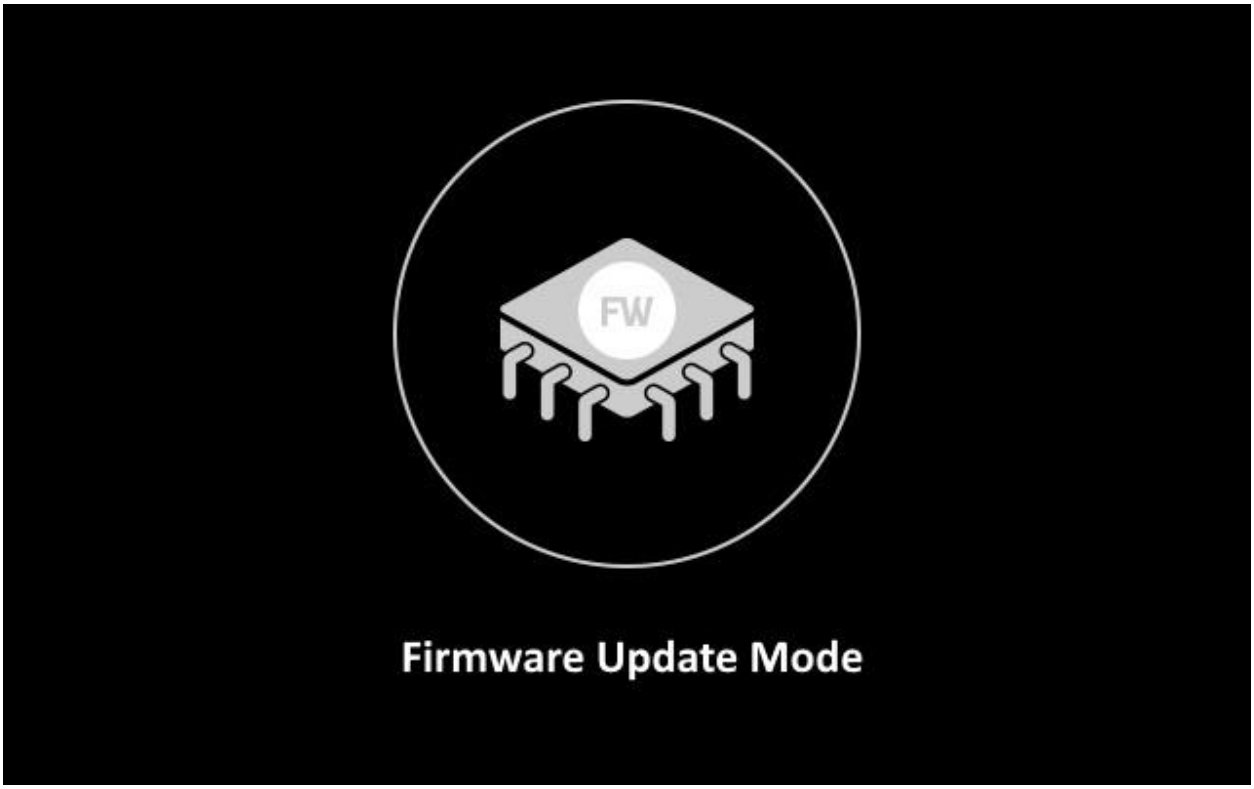
6. Known Issues

None

7. Troubleshooting

7.1 Device Boots into Safemode

Your device may occasionally display an image like the one below. This is because the device has entered bootloader mode.



7.1.1 Powercycle

When this happens, the first step you should take is to power cycle your device. Make sure to close the application connecting the device to your computer before unplugging and reconnected the HUD.

7.1.2 Reflash the firmware

If power cycling does not work, your next step is trying to reflash the firmware. To do this, refer to 555ES0012-03 WINDOWS SERVICE USER GUIDE section 4.3 Firmware Update for instructions on using the Update Firmware function of the Six15 application.

7.1.3 Contact Six15

If all else fails and you are unsure why your device does not appear to be working, please contact Six-15.

Email: Info@six-15.com

8. SDK Changelog

8.1 SDK 2.1 to 2.2

Firmware (5.23):

- Power Usage:
  - Reduced idle power: 425mW
  - Static image display power: 500mW
- Tracker:
  - Added calibration routines
  - Corrected quaternion output
  - Increased rate to 50Hz
- Stability fixes

Windows Service (3.1.1)

- New window/screen/area capture interface
- Added Tracker IMU to API

8.2 SDK 2.2 to 2.3

Firmware (5.30)

- Display:
  - Improved image pipeline to support framerates up to 30 fps
- Microphone:
  - Firmware now restarts when mic enable state is changed (Used to be done in service)
- Stability fixes:
  - Corrected issue: IMU caused reset when not initialized correctly
  - Corrected issue where camera did not work on Zebra TC51
  - Corrected issue where microphone did not work on Android 10 (updated sample rate to 48kHz)

8.3 SDK 2.4 to 2.5

- Tracker (IMU)
  - Improved calibration routine
  - Added calibration progress reporting.
  - Improved filter for calculating pose (roll, pitch, yaw)
  - Samples are output at 60Hz (from 30Hz).
  - **Breaking Change**: magnetometer readings are in units of uT (instead of mG). This is to conform to the Android sensor specification.
  - **Breaking Change**: The orientation of the accel/gyro/magnetometer is changed to adhere to Android standard (see: https://developer.android.com/reference/android/hardware/SensorEvent#values)
- Camera
  - Corrected bug where video locked up on highest resolution.

8.4 SDK 2.5 to SDK 2.6.3

Firmware (5.39):

- Tracker (IMU)
  - Corrected bug where IMU data is corrupted when calibrating some IMUs.
- Microphone
  - Corrected bug where ST1 reboots into bootloader if mic is opened/closed many times.
- Fixed issue where screen would flicker when using screen mirror and camera.
- Fixed issue where image display was sometimes delayed when recovering from low-power mode.

8.5 SDK 2.6.3 to SDK 2.6.5

Firmware (5.41):

- Fixed crash when enabling autofocus

8.6 SDK 2.6.5 to SDK 2.7.0

Firmware (5.42):

- Add display dither functionality (disabled by default)

9. Document Revision History

Table 5 – Document Revision History

Version Number	Date	Author/Owner	Description of Change
1.00	11/18/2018	George Vigelette	Initial Release
2.00	11/01/2019	A Sojda	Updated for 2.00 SDK release
2.01	01/27/2020	A Sojda	Update to 2.01 Release, see section 8.1 for changelog.
2.2	03/04/2020	John Martel	Changes to format, spelling, and grammar. Made stylistic changes for ease of reading. Adjusted hierarchy of information. Rearranged content to flow easier.
2.3	07/06/2020	A Sojda	Update to 2.3.
2.5	3/31/2021	A Sojda	Update to 2.5.0
2.6.3	10/28/2021	A Sojda	Update to 2.6.3
2.6.5	11/17/2021	A Sojda	Update to 2.6.5
2.7.0	12/07/2021	A Sojda	Update to 2.7.0