Wacom STU SDK – Getting Started

Global Signature Development Team

February 2016

Contents

1 Getting Started

Welcome to the Wacom STU SDK. Use this SDK to control all aspects of the Wacom STU-series of signature tablets.

You can:

- Query the tablet for its current settings and make changes to them.
- Send images to the tablet's LCD screen.
- Receive and decode pen stylus data.

The SDK is written in C++11 and the source code is provided. However there are no limitations if you program in a different language. C and Java language bindings (Java relies upon JNI) are available with virtually all the same functionality as the core C++. All this is supported on Windows and Linux platforms. On Windows, there is also a COM library which provides support for any ActiveX-capable language such as .NET Framework (C#), Delphi, and HTML (JavaScript) from within Internet Explorer.

2 Which language to use

2.1 C++

The SDK's native language is C++. This is provided as source code and so can be compiled directly into your program. Note that the source code requires a number of prerequisites:

- Boost library
- Windows Driver Kit
- libusb library

Dependencies on external components such as zlib compression and encryption are optional. These can be pulled in by the developer statically or dynamically according to preference and decisions on third-party licensing models. The source can be used on all supported platforms; it has been tested with Visual Studio 2010 and GCC 4.6.2.

2.2 All other languages

All other language bindings make use of a pre-compiled binary library with a base name of **`wgssSTU`**. On Windows this is **`wgssSTU.dll`**, while on Linux this is **`libwgssSTU.so`**. You will need to ensure there is a binary library provided for your target platform.

2.3 Java

Support is supplied with a package called **`wgssSTU.jar`** which uses the binary library to perform the low-level interaction with the hardware. We have not found a way to safely bundle the binary within the package so the developer is required to keep the files synchronized and run the Java application with the correct settings to locate the binary library. It requires a minimum of Java 1.5, with the **`Tablet`** class requiring Java 1.7.

2.4 C

The language is supported via a single **`wgssSTU.h`** header file to the binary library. The binary library supports linking statically or dynamically directly with functions or through a function table. The interface is a C interpretation of the C++ interface, with C++ exceptions turned back into error codes. Although this is a supported language, we would not typically recommend using this language binding due to the potential for programming errors.

2.5 COM (ActiveX)

Support is available on Windows. The binary library can be registered (using regsvr32.exe) or used without registration with an SxS (side-by-side) manifest (especially if developing a new application). Registration is required for use with Internet Explorer and during the development of .NET applications (when using Visual Studio), but are not necessary for application execution if deployed with a suitable manifest.

2.6 HTML (JavaScript)

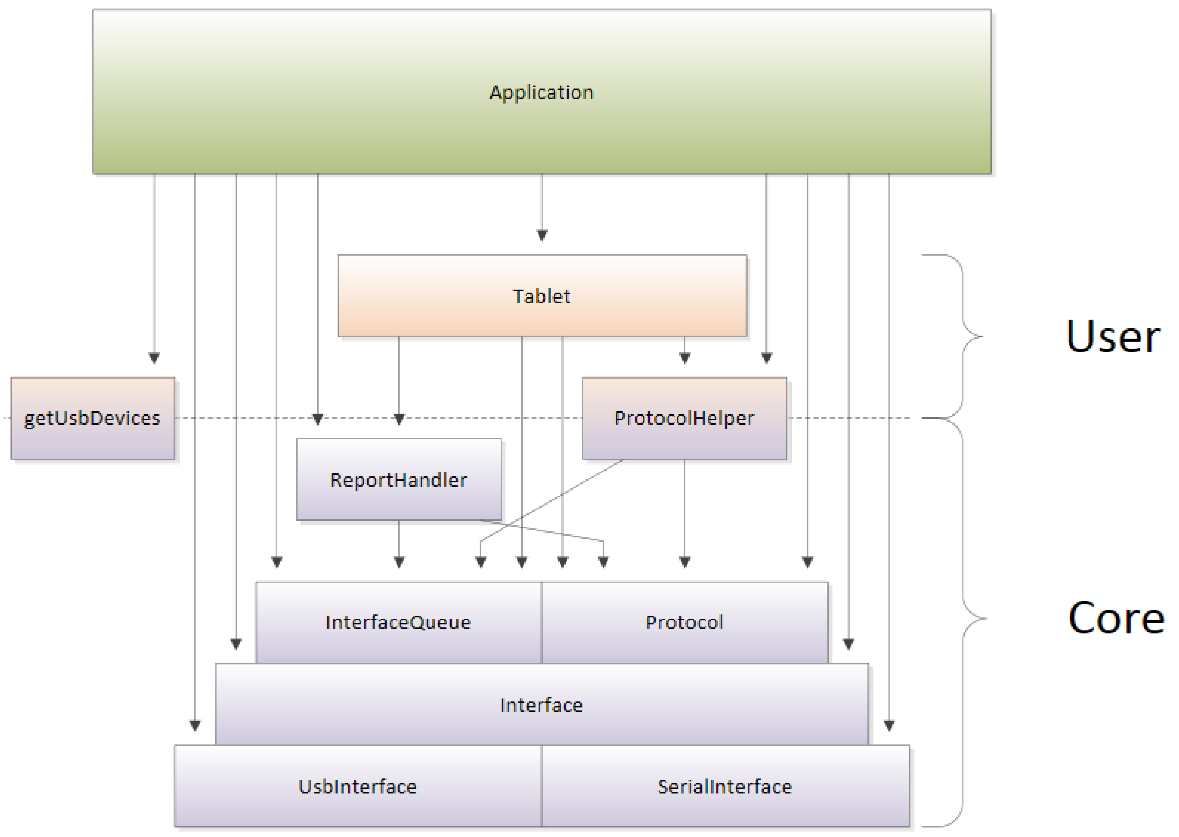
COM/ActiveX support is limited to Internet Explorer while a Netscape plugin is installed to support alternative browsers such as Chrome and Firefox.
Update notice:
Internet Explorer versions newer than IE10 no longer support ActiveX and similarly new versions of Chrome and Firefox will no longer support Netscape plugins using NPAPI.
An alternative browser solution has been developed to resolve these issues:
STU-SigCaptX.
Once released, this package will be included in the STU-SDK as a separate installer. The installation will create a local service and server for use with web based applications. The package gives HTML Javascript indirect access to the functions in the Windows binary library **`wgssSTU.dll`**.

3 Overview

As far as possible, in each language the classes, interfaces and methods are named identically. The SDK has been built in discrete layers and you can pick and choose as you prefer. Generally, all functionality has been made public, including the internal workings so that very fine control can be made if necessary. Logically, the SDK can be split into simpler User APIs and the low-level Core APIs.

Note that this SDK is a low-level hardware programming interface, not a user interface and there are no user interface components within the API. This is one of the key factors which made the API readily ported across languages and platforms.

The diagram below illustrates the key components within the SDK.



Although no encryption code is provided, the API is designed with encryption in mind, so that you can add your own routines very easily.

3.1 The *User* API

The User API is the highest-level API, providing the simplest interface to interact with the tablet.

Use the **`getUsbDevices()`** method to return an array of attached devices, and pass that to the appropriate **`connect()`** method of class **`Tablet`** (you can also connect via a serial port).

The **`Tablet`** class is the primary class that you use, and this provides basic protection of I/O timings and state-changes within the device as well as against calling functionality that the specific tablet may not have. This is especially helpful if encryption is used.

To upload an image, use an appropriate method from within **`ProtocolHelper`** to prepare the image into the native tablet data format before sending it via your tablet object.

The exact mechanism for receiving pen data is language dependent but in all but C++, the class automatically queues and decodes the data for you. In C++ use an **`InterfaceQueue`** from the tablet object and decode incoming data using class **`ReportHandler`**.

If necessary, it is possible to use any of the Core API while using the **`Tablet`** class.

3.2 The *Core* API

Below the **`Tablet`** class, the **`Protocol`** and **`Interface`** classes provide direct access to the workings of the tablet. This is not generally required, but is not hidden from use if any limitation is found within the higher level API. No sanity-checking is performed on the input or output data, which is transferred directly as the developer requests it.

As per the User API, use the **`getUsbDevices()`** method to return an array of attached devices, and pass that to the appropriate **`connect()`** method of **`UsbInterface`**. Serial devices can be connected via the **`SerialInterface`** class.

Then we recommend you use the **`Interface`** class which abstracts which type of port the tablet is connected to once the connection has been established.

You can then use the **`Protocol`** class to communicate to the tablet. In theory you can even craft your own API packets and send them to the device directly. Note that the methods in **`Protocol`** do not check that the tablet is in a valid state to accept the command, nor does it check whether the operation completed successfully on the device - you must do this yourself.

For ease of use, there are some helper functions in **`ProtocolHelper`** that simplify common tasks such as waiting for a command to complete and preparing images to send.

To receive pen data, you must get an **`InterfaceQueue`** object from an **`Interface`** object. This reports incoming raw data, which can then be given to **`ReportHandler`** to decode.

3.3 Windows SDK Installation

Each language has its own sample code and programming reference. Having installed the SDK you will find separate Doc folders containing the full API reference, see index.html for each language:

- C
- COM
- CPP

The SDK includes binaries for 32-bit and 64-bit windows.
The installer registers the 32-bit COM component **`wgssSTU.dll`** for immediate use in the sample code.

3.4 Linux development

- Ensure you have the dependencies installed. This should include: libusb-1.0 and libusb-1.0-devel.
- We have currently tested only on fedora16 i686.
- Allow user-access to your STU tablet. This usually requires adding a udev rule, for example:

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="056a", ATTRS{idProduct}=="00a[0-9|a-f]",
MODE="0666", GROUP="users"
```

- NOTE: Remember to install the **`libwgssSTU.so`**, for example (note paths maybe different):

```
sudo cp libwgssSTU.so.0.0.x /usr/local/lib
sudo /sbin/ldconfig /usr/local/lib
```

NOTE: You may need to adjust your LD_LIBRARY_PATH variable as well.

4 Signature Capture Basics

A signature capture application will typically follow these steps:

- Get available devices
- Connect device
- Display user interface image
- Receive pen data
- Generate signature image

The steps are outlined below with code based on the C# sample application DemoButtons. The full sample can be built and run with Visual Studio 2010 or above, including Visual Studio Express.

4.1 Get available devices

```
wgssSTU.UsbDevices usbDevices = new wgssSTU.UsbDevices();
if (usbDevices.Count != 0)
{
   wgssSTU.IUsbDevice usbDevice = usbDevices[0]; // select a device
```

Note that a number of USB devices can be connected and one must be selected from the list.
If a serial connection is used, a specific COM port must be selected.

4.2 Connect device

```
wgssSTU.Tablet m_tablet = new wgssSTU.Tablet();
wgssSTU.ICapability  m_capability  = m_tablet.getCapability();
wgssSTU.IInformation m_information = m_tablet.getInformation();
    ...
wgssSTU.IErrorCode ec = m_tablet.usbConnect(usbDevice, true);
if (ec.value == 0)
 {
    m_capability  = m_tablet.getCapability();
    m_information = m_tablet.getInformation();
    ...
 }
```

If a serial device is connected use the serial connection function, for example:

```
wgssSTU.IErrorCode ec = m_tablet.serialConnect("COM7",true);
if (ec.value == 0)
 {
    m_capability  = m_tablet.getCapability();
    m_information = m_tablet.getInformation();
    ...
 }
```

Having connected the device its properties can be accessed:

Capability includes:

- screenWidth
- screenHeight

Information includes:

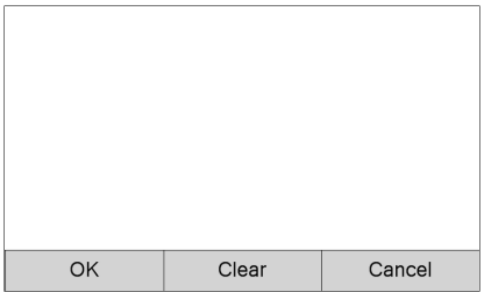
- modelName

4.3 Display user interface image

Create a bitmap image which can be used to guide the user, for example:

```
bitmap = new Bitmap(capability.screenWidth, capability.screenHeight,
         System.Drawing.Imaging.PixelFormat.Format32bppArgb);
Graphics gfx = Graphics.FromImage(bitmap);
gfx.Clear(Color.White);

// gfx commands to draw the image
...
```



The image will normally contain 'buttons' which can be clicked on the STU display. In reality the application will detect the pen position and pressure on the STU and respond accordingly to a 'button' click.

Determine the image encoding mode for the connected STU model. The mode is dependent on:

- Mono or colour
- Colour resolution
- Usb bulk transfer option

```
// Calculate the encodingMode that will be used to update the image
wgssSTU.ProtocolHelper protocolHelper = new wgssSTU.ProtocolHelper();

ushort idP = m_tablet.getProductId()
wgssSTU.encodingFlag encodingFlag =
(wgssSTU.encodingFlag)protocolHelper.simulateEncodingFlag(idP, 0);

bool useColor = false;
if ((encodingFlag & (wgssSTU.encodingFlag.EncodingFlag_16bit |
  wgssSTU.encodingFlag.EncodingFlag_24bit)) != 0)
{
  if (m_tablet.supportsWrite())
    useColor = true;
}
if ((encodingFlag & wgssSTU.encodingFlag.EncodingFlag_24bit) != 0)
{
    m_encodingMode = m_tablet.supportsWrite() ?
      wgssSTU.encodingMode.EncodingMode_24bit_Bulk :
      wgssSTU.encodingMode.EncodingMode_24bit;
}
else if ((encodingFlag & wgssSTU.encodingFlag.EncodingFlag_16bit) != 0)
{
    m_encodingMode = m_tablet.supportsWrite() ?
      wgssSTU.encodingMode.EncodingMode_16bit_Bulk :
      wgssSTU.encodingMode.EncodingMode_16bit;
}
else
{
    // assumes 1bit is available
    m_encodingMode = wgssSTU.encodingMode.EncodingMode_1bit;
}
```

Prepare the bitmap for transfer

```
System.IO.MemoryStream stream = new System.IO.MemoryStream();
bitmap.Save(stream, System.Drawing.Imaging.ImageFormat.Png);
bitmapData = (byte[])protocolHelper.resizeAndFlatten(stream.ToArray(),
    0, 0, (uint)bitmap.Width, (uint)bitmap.Height,
    m_capability.screenWidth, m_capability.screenHeight,
    (byte)m_encodingMode, wgssSTU.Scale.Scale_Fit, 0, 0);
```

Send the bitmap to the STU display:

```
m_tablet.writeImage((byte)m_encodingMode, bitmapData);
```

4.4 Receive pen data

Add delegates to receive pen callback data:

```
m_tablet.onPenData += new
 wgssSTU.ITabletEvents2_onPenDataEventHandler(onPenData);

m_tablet.onGetReportException += new
 wgssSTU.ITabletEvents2_onGetReportExceptionEventHandler(onGetReportException);
```

Enable pen inking on the STU display:

```
M_tablet.setInkingMode(0x01);
```

Receive data as the pen is used on the STU display::

```
void onPenData(wgssSTU.IPenData penData)
{
// Process incoming pen data
```

penData is delivered while the pen is in contact with, or in close proximity to the STU display surface. The data includes the values:

rdy	True if the pen is in proximity with the tablet
sw	True if the pen is in contact with the surface
P	Pen pressure in tablet units
X	X-coordinate of the pen position in tablet units
Y	Y-coordinate of the pen position in tablet units

To use penData the coordinates need to be converted to normalised units, for example:

```
private Point tabletToScreen(wgssSTU.IPenData penData)
  {
  // Screen means LCD screen of the tablet.
  return Point.Round(new PointF(
    (float)penData.x * m_capability.screenWidth / m_capability.tabletMaxX,
    (float)penData.y * m_capability.screenHeight / m_capability.tabletMaxY));
  }
```

The pen coordinates are examined to detect a ‘button’ click:

```
    int btn = 0; // will be +ve if the pen is over a button.
    {
      for (int i = 0; i < m_btns.Length; ++i)
      {
        if (m_btns[i].Bounds.Contains(pt))
        {
          btn = i+1;
          break;
        }
      }
    }
```

The penData values are saved as a list of X/Y/P points which plot the signature:

```
private List<wgssSTU.IPenData> m_penData; // Array of data being stored.

...

m_penData.Add(penData);
```

4.5 Generate signature image

When the OK button has been clicked the signature can be reconstructed from the saved penData points. For example in its simplest form:

```
PointF prev = new PointF();
for (int i = 0; i < m_penData.Count; ++i)
  {
    if (m_penData[i].sw != 0)
    {
      if (!isDown)
      {
        isDown = true;
        prev = tabletToClient(m_penData[i]);
      }
      else
      {
        PointF curr = tabletToClient(m_penData[i]);
        gfx.DrawLine(m_penInk, prev, curr);
        prev = curr;
      }
    }
    else
    {
      if (isDown)
      {
        isDown = false;
      }
    }
  }
```

The sample shows the simplest form of drawing a signature. The image quality can be improved by adding curve smoothing and increasing the width of the line with pressure.