How-To Guide

SigPlus Java App Demo

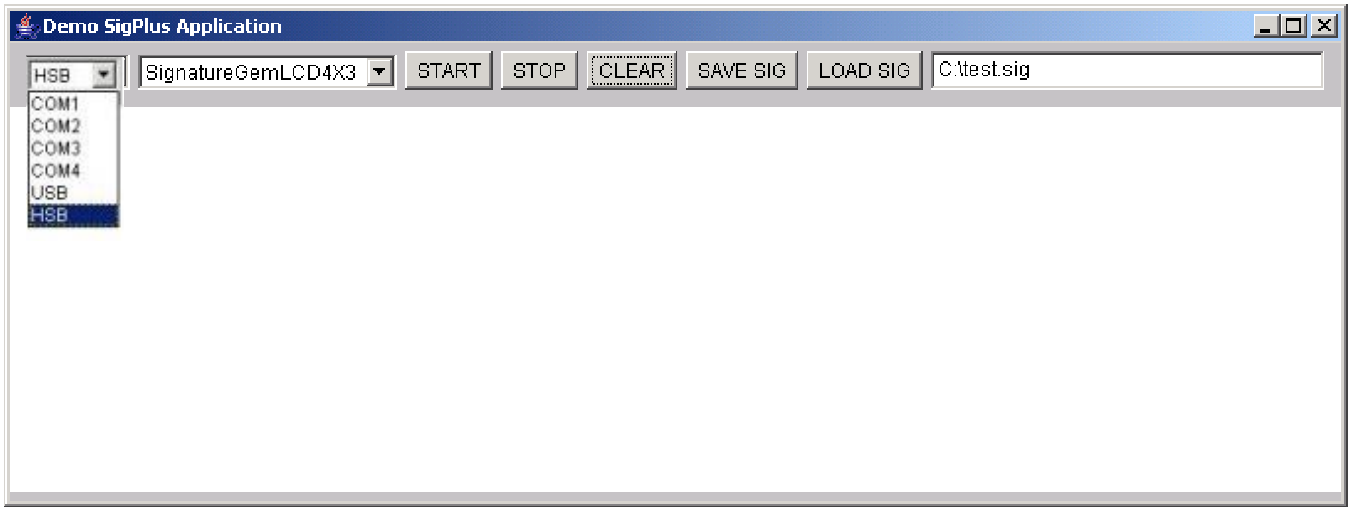
Table of Contents

Overview

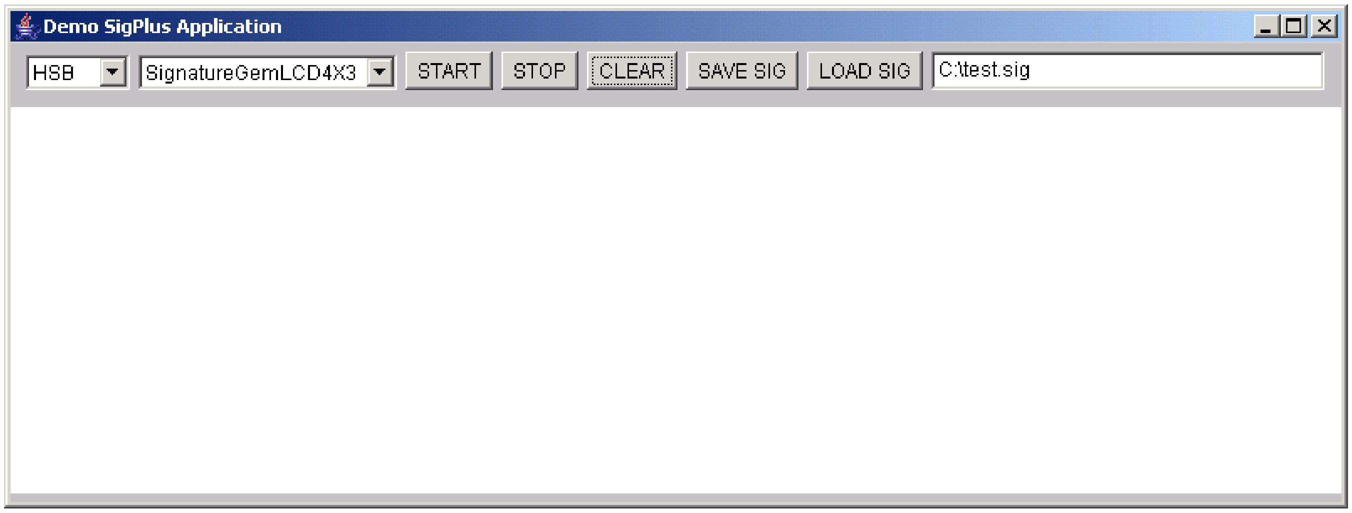
Welcome to the SigPlus App Demo Guide. This will walk you through our demo "SigPlusAppDemo.java", available from Topaz's website.

Download at: www.topazsystems.com/sigplusprojava.html

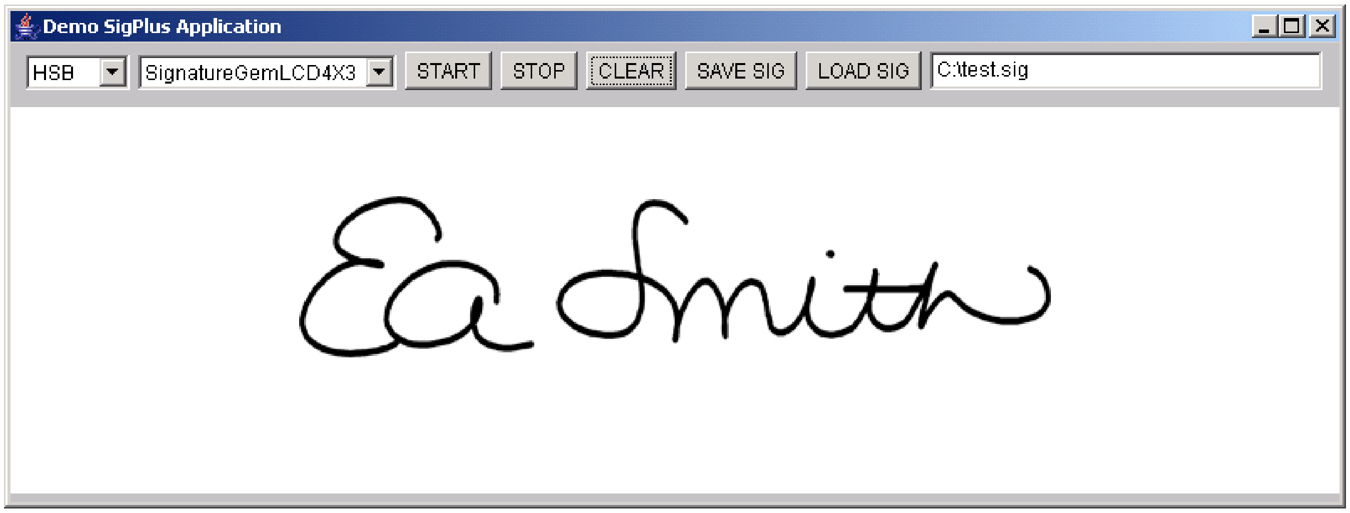
This application will show Java developers how to integrate SigPlus technology into their own applications and to export and import a signature image as a .sig file. Begin by opening "SigPlusAppDemo.java" in a Java runtime environment. Set the connection type to correspond with the connection type you are using. We used an HSB tablet for the purposes of this demo documentation.



Now, ensure that the "Tablet Model" you are using corresponds to the tablet type selected by the demo application. For the purposes of this demo, we used a SignatureGem LCD 4X3.



After configuring the application for use with your signature capture device, click “Start” to activate the tablet, and sign your name. Your signature will be displayed in the window. To erase your signature and start over, push “Clear”. However, if you do push “Clear”, be sure to push “Start” again before resigning. If you would like to turn the pad off, push the “Stop” button. When you are ready to save, click “Save Sig” to save your signature as a .sig file.



Your signature will be saved to the location specified in the text box on the far right of the application. The default location is C:\test.sig.

The Code

Next, we will look over the Java code used to control this application. The code below creates a form for the SigPlusAppDemo using the gridbag class and the panel class. Buttons are defined to later call upon button events.

```
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import com.topaz.sigplus.*;
import javax.comm.*;
import java.io.*;
import javax.swing.*;

public class SigPlusAppDemo extends Frame implements Runnable
        {
        SigPlus         sigObj = null;
        Thread  eventThread;

        public static void main( String Args[] )
                {
                SigPlusAppDemo demo = new SigPlusAppDemo();
                demo.setSize(800,300);
                demo.setVisible(true);
           demo.setBackground(Color.lightGray);
```

```
                }
        public SigPlusAppDemo()
                        {
            GridBagLayout gbl = new GridBagLayout();
                        GridBagConstraints gc = new GridBagConstraints();
                        setLayout(gbl);
                        Panel controlPanel = new Panel();
                        setConstraints(controlPanel, gbl, gc, 0, 0,
                        GridBagConstraints.REMAINDER, 1, 0, 0,
                        GridBagConstraints.CENTER,
                        GridBagConstraints.NONE,0, 0, 0, 0);
                        add(controlPanel, gc);

                        controlPanel.add(connectionChoice);
                        controlPanel.add(connectionTablet);

                        Button startButton = new Button("START");
                        controlPanel.add(startButton);

                        Button stopButton = new Button("STOP");
                        controlPanel.add(stopButton);

                        Button clearButton = new Button("CLEAR");
                        controlPanel.add(clearButton);

                        Button saveSigButton = new Button("SAVE SIG");
                        controlPanel.add(saveSigButton);

                        Button loadSigButton = new Button("LOAD SIG");
                        controlPanel.add(loadSigButton);

                        controlPanel.add(txtPath);

                        Button okButton = new Button("QUIT");
                        controlPanel.add(okButton);
```

The code below initializes the com API. This is necessary even with HSB pads because Java expects to see the com initialized. The com driver class is only necessary while in Windows. It then uses the ClassLoader class to create a new instance of SigPlus.

```
                        initConnection();

                        String drivername = "com.sun.comm.Win32Driver";
                        try
                                {
                                CommDriver driver = (CommDriver)
                                Class.forName(drivername).newInstance();
                                driver.initialize();
                                }
                        catch (Throwable th)
                                {
                                /* Discard it */
                                }
```

```
                    try
                            {
                            ClassLoader cl =
            (com.topaz.sigplus.SigPlus.class).getClassLoader();
                            sigObj = (SigPlus)Beans.instantiate( cl,
            "com.topaz.sigplus.SigPlus" );

                    setConstraints(sigObj, gbl, gc, 0, 1,
                    GridBagConstraints.REMAINDER, 1, 1, 1,
                    GridBagConstraints.CENTER,
                    GridBagConstraints.BOTH, 5, 0, 5, 0);
                    add(sigObj, gc);
                    sigObj.setSize(100,100);
                sigObj.clearTablet();
                    setTitle( "Demo SigPlus Application" );
```

When the “Ok” button, which is titled as the “Quit” button, is pushed, the tablet is turned off and the application is closed.

```
                okButton.addActionListener(new     ActionListener(){
            public void actionPerformed(ActionEvent e){
                sigObj.setTabletState(0);
              System.exit(0);
            }
        });
```

When the “Start” button is pushed, the tablet is activated to accept signatures.

```
        startButton.addActionListener(new     ActionListener(){
            public void actionPerformed(ActionEvent e){
                sigObj.setTabletState(0);
                sigObj.setTabletState(1);
            }
        });
```

When the “Stop” button is pushed, the tablet is turned off and no longer accepts any signatures. However, unlike the “Quit” button, the application still runs.

```
        stopButton.addActionListener(new     ActionListener(){
            public void actionPerformed(ActionEvent e){
                sigObj.setTabletState(0);
            }
        });
```

When the “Clear” button is pushed, the signature is cleared from the signature field.

```
clearButton.addActionListener(new     ActionListener(){
        public void actionPerformed(ActionEvent e){
                sigObj.clearTablet();
         }
});
```

The code below saves the signature to .sig format. The filepath is set to the contents of the text box; the .sig is created, and then saved to the filepath. The signature is encrypted to the string “Sample Encryption Data”.

```
saveSigButton.addActionListener(new     ActionListener(){
        public void actionPerformed(ActionEvent e){
                boolean blnExport=false;
                String path=txtPath.getText();
int pathlength=path.length();
if(pathlength!=0)
{
                sigObj.autoKeyStart();
                sigObj.setAutoKeyData("Sample Encryption Data");
                sigObj.autoKeyFinish();
        sigObj.setEncryptionMode(1);
                blnExport = sigObj.exportSigFile(txtPath.getText());
        if (blnExport==false)
           {
                System.out.println("Error writing SIG file");
           }
        }
        else
        {
           System.out.println("Please type in full path information to save file");
                }
        }
           });
```

The code below imports a signature back into the SigPlusApp java demo. The filepath is taken from the textbox, and the signature displayed in the window. The signature is decrypted using the string “Sample Encryption Data”.

```
loadSigButton.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
                boolean blnImport=false;  String
                path=txtPath.getText();
        int pathlength=path.length();
        if(pathlength!=0)
        {
                sigObj.autoKeyStart();
                        sigObj.setAutoKeyData("Sample Encryption Data");  sigObj.autoKeyFinish();
                sigObj.setEncryptionMode(1);
```

```
                        blnImport = sigObj.importSigFile(txtPath.getText());
                if(blnImport==false)
                        {
                                System.out.println("Error reading SIG file");
                        }
                }
                else
                {
                        System.out.println("Please type in full path information to load file");
                                }
                }
        });
        //txtPath.addTextListener(new TextListener(){
                //public void textValueChanged(TextEvent e){
                //System.out.println(txtPath.getText());
                //}
        //});
```

This section of code sets the tablet model to match the selection from the drop down menu. If you use a SignatureGem LCD 4X3, it must be set to SignatureGem LCD 4X3 New. Otherwise, the choice matches the entry from the drop down list.

```
        connectionTablet.addItemListener(new ItemListener(){  public void
             itemStateChanged(ItemEvent e){

            if(connectionTablet.getSelectedItem() != "SignatureGemLCD4X3"){
               sigObj.setTabletModel(connectionTablet.getSelectedItem());
            }
            else{
          sigObj.setTabletModel("SignatureGemLCD4X3New"); //properly set up LCD4X3
            }
                         }
         });
```

The following code sets the connection type to match the selection from the drop down menu. If you select HSB, it must be set to “HID1”. Otherwise, the choice matches the entry from the drop down list.

```
         connectionChoice.addItemListener(new ItemListener(){  public void
            itemStateChanged(ItemEvent e){

            if(connectionChoice.getSelectedItem() != "HSB"){
                  sigObj.setTabletComPort(connectionChoice.getSelectedItem());
            }
            else{
               sigObj.setTabletComPort("HID1"); //properly set up HSB tablet
            }
                                        }
        });
```

When you prompt the window to close, this ensures the tablet is off. There are three SigPlus event handlers in SigPlus that are currently unimplemented. These are not available for the developer to use.

```
 AddWindowListener( new WindowAdapter()
         {
         public void windowClosing( WindowEvent we )
                         {
                  sigObj.setTabletState( 0 );
                         System.exit( 0 );
                         }
         public void windowClosed( WindowEvent we )
                         {
                         System.exit( 0 );
                         }
         } );
sigObj.addSigPlusListener( new SigPlusListener()
         {
         public void handleTabletTimerEvent( SigPlusEvent0 evt )
                         {
                         }

         public void handleNewTabletData( SigPlusEvent0 evt )
                         {
                         }

         public void handleKeyPadData( SigPlusEvent0 evt )
                         {
                         }
         } );
```

This next section selects the defaults. A thread called eventThread is then created to run the application.

```
                  show(); sigObj.setTabletModel("SignatureGem1X5");
         sigObj.setTabletComPort("COM1");
                  eventThread = new Thread(this);
                  eventThread.start();
                  }
                 catch ( Exception e )
                  {
                  return;
                  }}
   public void run()
      {
      try
           {
           while ( true )
                  {
                 Thread.sleep(100);
                  }
```

```
        }
    catch (InterruptedException e)
        {   } }
```

The code below sets the default for the filepath for saving and loading .sig.

```
    TextField txtPath = new TextField("C:\\test.sig", 30);
```

The code below creates and defines the items of the drop down menu for connection type choices.

```
    Choice connectionChoice = new Choice();   protected String[] connections =
    {
        "COM1",
        "COM2",
        "COM3",
        "COM4",
        "USB",
        "HSB",
    };
```

The code below creates and defines the items in the drop down menu for tablet type choices.

```
Choice connectionTablet = new Choice();   protected String[] tablets =
    {
    "SignatureGem1X5", "SignatureGem4X5",
        "SignatureGemLCD",
        "SignatureGemLCD4X3", "ClipGem",
        "ClipGemLGL",
    };
```

The following code specifies how many choices there will be for the menus above.

```
private void initConnection()
        {
            for(int i = 0; i < connections.length; i++)
            {
                connectionChoice.add(connections[i]);
            }
            for(int i = 0; i < tablets.length; i++)
            {
                connectionTablet.add(tablets[i]);
            }
          }
```

Dependencies and Files Necessary to Run This Demo

Jar files must be included in the CLASSPATH

1. SigPlus2_xx.jar (at the creation of this document the current version was 2_45)
2. Comm.jar (This file will vary depending on OS, you will need the correct java com API for your OS.)

Additional Files

1. (Win) – Win32com.dll must reside in the System32 folder
2. (HSB in Win) – SigUsb.dll must reside in the System32 folder