



Topaz Systems, Inc. Java iText Demo Guide

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065
©1995-2006, all rights reserved, US patent 6,307,955, and pending

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

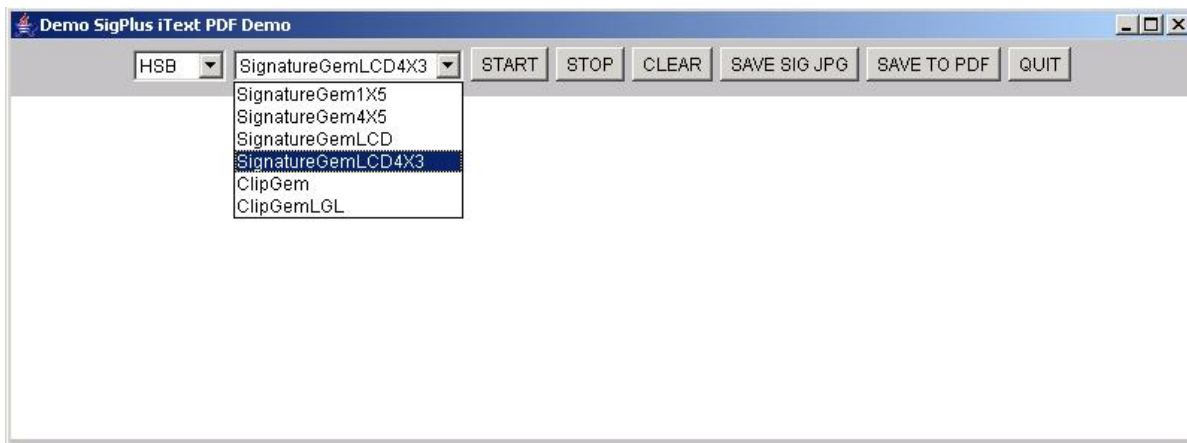
Welcome to the SigPlus Java iText Demonstration Guide. This will walk you through our demo “SigPlus_iTextDemo.java”, available from our website (www.topazsystems.com). To download go to:

http://www.topazsystems.com/software/download/java/sigplusjava_itext_demo.zip

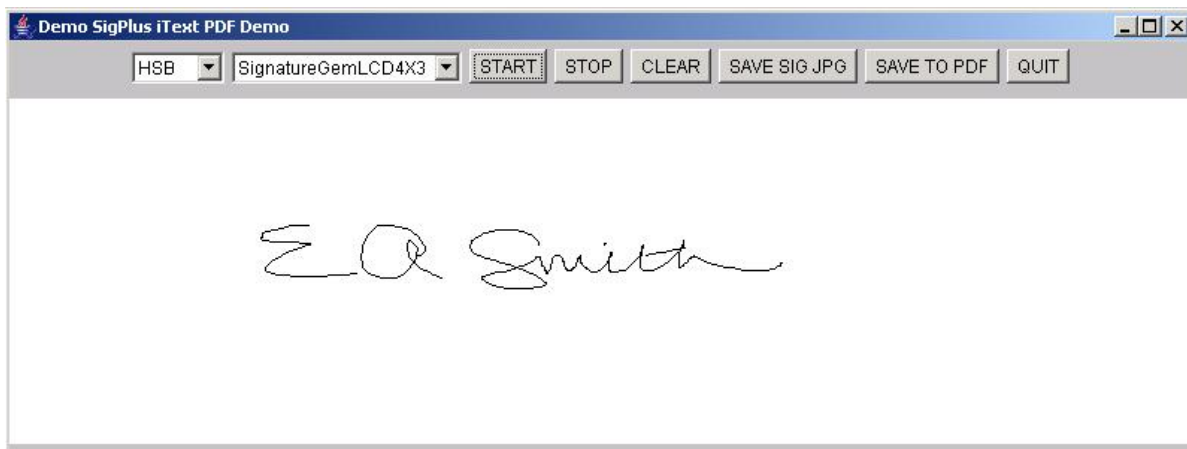
This application will show Java developers how to integrate SigPlus technology into their own applications, and export a signature image into a PDF document. Begin by opening SigPlus_iTextDemo.java in a Java runtime environment. Set the connection type (shown below) to correspond with the connection type you are using. We used an HSB tablet for the purposes of this demo documentation.



Now ensure 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 SignatureGemLCD4X3 tablet (shown below).



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 (shown below). 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 either “Save to JPG” to save your signature as a .JPG image file, or click “Save to PDF” to import your signature image to a PDF document.



If you import the signature image into a PDF the document, it will be saved as “sig.pdf” in the same folder as “SigPlus_iTextDemo.java.” Of course you can change to location and filename in your own application. If you chose to save as a JPG the , it will be saved as “sig.jpg,” and will be stored in your Local Disk (C:). This operation is independent of the PDF demo and can be ignored if you simply want to understand how to write to PDF format. Press “Quit” when you are finished to exit the application.

The Code

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

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

public SigPlus_iTextDemo()
{
    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 savePdfButton = new Button("SAVE TO PDF");
    controlPanel.add(savePdfButton);

    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 drivename = "com.sun.comm.Win32Driver";
try
{
    CommDriver driver = (CommDriver)
Class.forName(drivename).newInstance();
    driver.initialize();
}
```

```

    }
    catch (Throwable th)
    {
        /* Discard it */
    }
    try
    {
        ClassLoader cl =
(com.gemtools.sigplus.SigPlus.class).getClassLoader();
        sigObj = (SigPlus)Beans.instantiate( cl,
"com.gemtools.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 and iText PDF" );
    }
}

```

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 (shown below).

```

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 (shown below). 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 (shown below).

```

clearButton.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        System.out.println(sigObj.getKeyReceipt());
        sigObj.clearTablet();
    }
});

```

When the Save PDF button is pushed a new PDF document is created. This is done through a series of steps. First a new PDF document is created, and the signature is sized to fit the developer's needs. For the sake of the demo we chose an image size of 400x100 pixels, with an ink thickness of four pixels. SigPlus then creates a new buffered image called sigImage. Variables w and h are created to correspond to the height and width of the signature. The RGB values are set to 0 so that the signature will be displayed in black and white only. A new com.lowagie.text.Image instance is created called img1 using the iText.jar file. The GetInstance function then sets the sigImage from SigPlus to img1. Finally the location of the image is specified and placed on the document.

```

savePdfButton.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){

        Document document = new Document();

        try {

            PdfWriter.getInstance(document, new
            FileOutputStream("sig.pdf"));
            document.open();

            sigObj.setTabletState(0);
            sigObj.setImageJustifyMode(5);
            sigObj.setImagePenWidth(4);
            sigObj.setImageXSize(400);
            sigObj.setImageYSize(100);
            BufferedImage sigImage = sigObj.sigImage();
            int w = sigImage.getWidth(null);
            int h = sigImage.getHeight(null);
            int[] pixels = new int[(w * h) * 2];
            sigImage.setRGB(0, 0, 0, 0, pixels, 0, 0);
            com.lowagie.text.Image img1 =
            com.lowagie.text.Image.getInstance(sigImage, null, true);
            img1.setAbsolutePosition(40, 400);
            document.add(img1);

        }
        catch(DocumentException de) {
            System.err.println(de.getMessage());
        }
        catch(IOException ioe) {
            System.err.println(ioe.getMessage());
        }

        document.close();

    }
});

```

The code below saves the signature to JPG format. First the tablet is turned off, and the image is formatted. Variable W and H are assigned respectively to the Width and Height of the image. The filepath is set to the "C:Sig1.jpg", the Jpeg is created, and then saved to the filepath.

```

saveJpgButton.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){

```

```

        try {
            sigObj.setTabletState(0);
            sigObj.setImageJustifyMode(5);
            sigObj.setImagePenWidth(10);
            sigObj.setImageXSize(1000);
            sigObj.setImageYSize(350);
            BufferedImage sigImage = sigObj.sigImage();
            int w = sigImage.getWidth(null);
            int h = sigImage.getHeight(null);
            int[] pixels = new int[(w * h) * 2];

            sigImage.setRGB(0, 0, 0, 0, pixels, 0, 0);
            FileOutputStream fos = new
            FileOutputStream("c:\\sig.jpg");

            JPEGImageEncoder jpeg =
            JPEGCodec.createJPEGEncoder(fos);
            jpeg.encode(sigImage);
            fos.close();

        }
        catch (Throwable th) {
            th.printStackTrace();
        }
    }
});

```

This section of code sets the tablet model to match the selection from the drop down menu. If you use a SignatureGemLCD4X3, it must be set to SignatureGemLCD4X3New. 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");
                }
            }
        });

```

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

```

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

        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]);
        }
    }

```

Because this demo will not create a biometric or an encrypted signature but instead creates a .jpeg image, the document is not legally binding. This is a demo only and should be used as a blueprint for creating your own signed PDF documents.

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.)
3. iText.jar (available at <http://www.lowagie.com/iText/>
 - a. Please be sure to read the licensing agreement(s) from the lowagie web site to be sure your application meets the licensing requirements set forth there before you begin using the api.

Additional Files

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

For more information regarding products and support, please visit the Topaz web site at <http://www.topazsystems.com>