Samsung Electronics

S Pen SDK 2.2 Tutorial

Contents

S Pen SDK 2.2 Tutorial

This document provides SDK usage instructions and example code for assisting developers in using the S Pen SDK as easily as possible. For more information on "how to" with each API provided in the S Pen SDK, please refer to the API Reference document included in the SDK.

This document introduces the S Pen SDK and demonstrates the API using example code. This document does not include information about building the basic Android development environment and settings.

1. Introduction to the S Pen SDK

1-1. What is the S Pen SDK?

The S Pen SDK provides libraries in an Android environment so that developers can develop various applications using S Pen.

1-2. Composition of the S Pen SDK

The S Pen SDK consists of the following packages.

- Documentation: S Pen SDK-related JavaDoc (API Reference documents).
- Library: S Pen SDK Library.
- Examples: Example code written based on the S Pen SDK.

2. How to setup the S Pen SDK

2-1. Requirements

Using the S Pen SDK requires an Android development environment. To create an Android development environment, the following components must be installed first. For information about downloading and installing these components, please refer to the related links.

- JDK: Java SE Development Kit
  http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Android SDK
  http://developer.android.com/sdk/index.html
- IDE: Eclipse
  http://www.eclipse.org/downloads/

2-2. Adding S Pen Library to an Android Project

1. First, download the S Pen SDK from http://developer.samsung.com/kr/android/spen.sdk.
2. Create a new Android project in Eclipse.
3. Right-click on the new project and create a new folder called **libs.**
4. Locate the **libspen22.jar** file in the **libs** folder in the S Pen SDK folder that was extracted.
5. Copy the **libspen22.jar** file that you located in step 4 to the **libs** folder you created in step 3.
6. Include the native library files, which were used to support features such as signature recognition,
   and image processing (image filters), to the project.
   Create a folder called **armeabi** by repeating step 3 and copy the native library files (*.so) from the **libs/armeabi** folder from the installation path of the SDK to the folder you just created.
7. In Eclipse, add the **libspen22.jar** file to the build path.
8. Click **OK**. When the Properties dialog box appears, click **OK** again.
   The **libspen22.jar** file is now added to the project.

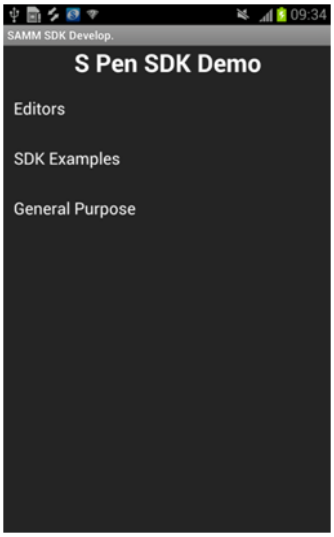
3. Features provided by the S Pen SDK

3-1. Composition of example program and the first example code

Unlike the previous versions that provided separate example programs for each feature, the S Pen SDK 2.2 provides one example program that includes multiple features.

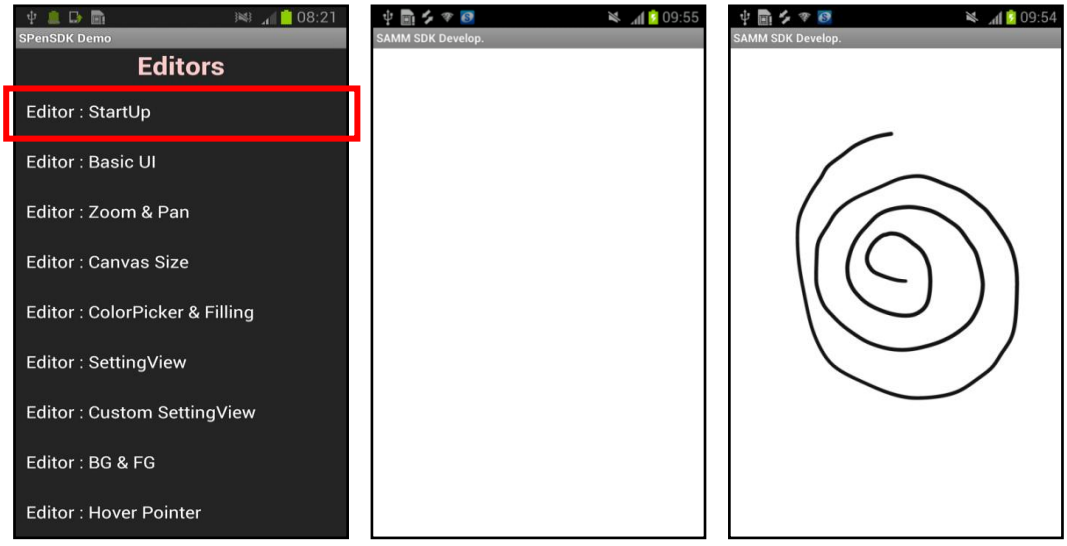
The new example program consists of three main menu items ("Editors", "SDK Examples", "General Purpose").

Each menu contains examples of each feature in the submenu.



[Figure 1] Start screen of the example program (main menu)

By analyzing the examples in "Editor: StartUp", which is the first item in the Editors menu, you can see the basic usage of the APIs.



[Figure 2] Execution screen of the first example

The first example, "Editor: StartUp", is an example that only provides the drawing function without any UI.
It is the most basic example for explaining the most basic functions of SCanvasView, which is a core component of the S Pen SDK.
By looking at the results of running this example, you can test the basic drawing feature provided by the SCanvasView class in the S Pen SDK.
For more information about other API's, please refer to the API Reference document included in the SDK.

3-2. SCanvasView Class

The SCanvasView class inherits from the View class and serves as a canvas in the drawing process. If you designate an SCanvasView object in an activity, the user can start drawing with a pen using basic line color, thickness, and opacity values.
Furthermore, the user can easily save and open drawings that he or she has drawn on the canvas.

3-2-1. SCanvasView Class Initialization

To use the SCanvasView class, you must first create a suitable layout for the SCanvasView object as shown in the example below.

Here is the layout used in the first example.

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android=¡±http://schemas.android.com/apk/res/android¡±
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="match_parent">
        <RelativeLayout
                        android:id="@+id/canvas_container"
                        android:layout_width="match_parent"
                        android:layout_height="match_parent"
                        android:layout_gravity="center">
        </RelativeLayout>
</LinearLayout>
```

[Code 1] editor_startup.xml

To show the drawing area of an SCanvasView on the screen, you must first load and designate the SCanvasView from the layout using the findViewByID method.
Here is the code for the first example.

```
@Override
public void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);

        setContentView(R.layout.editor_startup);

        mContext = this;

        mCanvasContainer = (RelativeLayout) findViewById(R.id.canvas_container);
        mSCanvas = new SCanvasView(mContext);
        mCanvasContainer.addView(mSCanvas);
        ...
```

[Code 2] SPen_Example_StartUp.java

As you can see in the example code, an SCanvasView from the S Pen SDK has been added in certain layout areas.
Just by entering the code above, you can use the basic drawing features provided by the S Pen SDK.

[Figure 3] SCanvasView drawing screen

3-2-2. Changing Modes in the SCanvasView Class

The SCanvasView class provides Pen, Eraser and Text Mode.

The Pen Mode is used to draw on the screen and the Eraser Mode is to erase the drawings on the screen. The Text Mode is used when the user enters text on the screen.

When initialized, the default value in SCanvasView class is Pen Mode.

As you can see in the code below, mode switching can be done easily by using the setCanvasMode method.

```
mSCanvas.setCanvasMode(SCanvasConstants.SCANVAS_MODE_INPUT_PEN);      // Pen Mode
mSCanvas.setCanvasMode(SCanvasConstants.SCANVAS_MODE_INPUT_ERASER); // Eraser Mode
mSCanvas.setCanvasMode(SCanvasConstants.SCANVAS_MODE_INPUT_TEXT);     // Text Mode
```

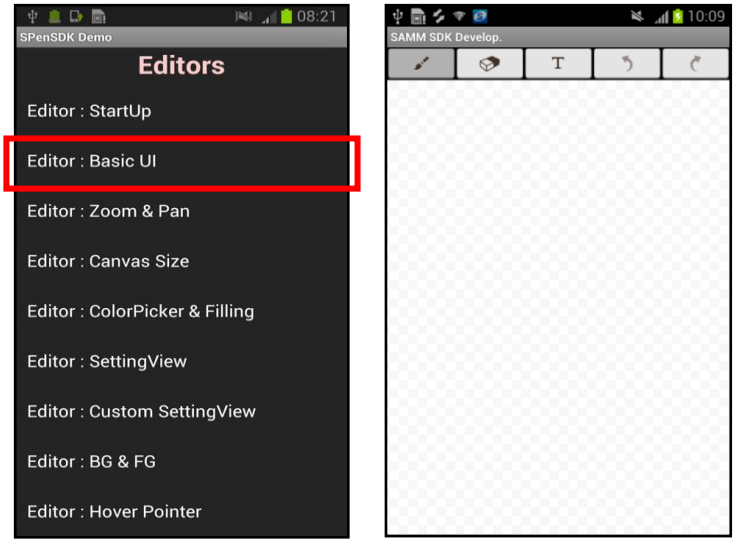
[Code 2] Switch to Pen, Eraser or Text Mode

Also, you can check the current mode value of the canvas by using the getCanvasMode method.

```
If ( mSCanvas.getCanvasMode()==SCanvasConstants.SCANVAS_MODE_INPUT_PEN )
{
    // Processed when the current mode of the SCanvasView is Pen Mode
}
```

[Code 4] Checking the current mode

3-2-3. Applying UI (Pen, Eraser, Text) to SCanvasView



[Figure 4] Run screen of the second example

The run screen of the second example, "Editor: Basic UI", consists of buttons on the top of the screen and a lower canvas area. The buttons on the top perform the following functions.

- **Pen** – Switches the input mode to Pen Mode or runs a setting pop-up window in Pen Mode.
- **Eraser** – Switches the input mode to Eraser Mode or runs a setting pop-up window in Eraser Mode.
- **Text** – Switches the input mode to Text Mode or runs a setting pop-up window in Text Mode.
- **Undo** – Cancels the last draw command and restores the previous state.
- **Redo** – Restores the last draw command that was undone.

Here is the layout used in the second example.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="match_parent">

        <LinearLayout
                android:id="@+id/tool_menu"
                android:orientation="horizontal"
                android:layout_gravity="top|right"
                android:layout_width="match_parent"
                android:layout_height="wrap_content">
                <ImageView
                        android:id="@+id/penBtn"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:layout_margin="1dip"
                        android:layout_weight="1"
                        android:background="@drawable/selector_tool_bg"
                        android:src="@drawable/selector_pen" />
                <ImageView
                        android:id="@+id/eraseBtn"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:layout_margin="1dip"
                        android:layout_weight="1"
                        android:background="@drawable/selector_tool_bg"
                        android:src="@drawable/selector_eraser" />
```

```
        <ImageView
            android:id="@+id/textBtn"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="1dip"
            android:layout_weight="1"
            android:background="@drawable/selector_tool_bg"
            android:src="@drawable/selector_text" />
        <ImageView
            android:id="@+id/undoBtn"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="1dip"
            android:layout_weight="1"
            android:background="@drawable/selector_tool_bg"
            android:src="@drawable/selector_undo" />
        <ImageView
            android:id="@+id/redoBtn"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="1dip"
            android:layout_weight="1"
            android:background="@drawable/selector_tool_bg"
            android:src="@drawable/selector_redo" />
    </LinearLayout>

    <FrameLayout
        android:id="@+id/layout_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="center">

        <RelativeLayout
            android:id="@+id/canvas_container"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center">
```

```
                    <ImageView
                            android:id="@+id/canvas_default_background"
                            android:layout_width="match_parent"
                            android:layout_height="match_parent"
                            android:src="@drawable/scanvas_bg"
                            android:scaleType="centerCrop"/>
                </RelativeLayout>
        </FrameLayout>
</LinearLayout>
```

[Code 5] editor_basic_ui.xml

The Pen, Eraser, or Text Settings pop-up window, which appears when each button is pressed, can be managed by using with SCanvasView methods.

Additionally, layout elements modified for different languages can be defined and registered in the settings pop-up window. When doing so, you can use a HashMap to set the required resources.

```
mLayoutContainer = (FrameLayout) findViewById(R.id.layout_container);

...

// Resource map
HashMap<String,Integer> settingResourceMapInt = new HashMap<String, Integer>();
// Layout
settingResourceMapInt.put( SCanvasConstants.LAYOUT_PEN_SPINNER, R.layout.mspinner );
settingResourceMapInt.put( SCanvasConstants.LAYOUT_TEXT_SPINNER, R.layout.mspinnertext );
settingResourceMapInt.put( SCanvasConstants.LAYOUT_TEXT_SPINNER_TABLET,
                            R.layout.mspinnertext_tablet);

// Supported languages by country
settingResourceMapInt.put( SCanvasConstants.LOCALE_PEN_SETTING_TITLE,
                            R.string.pen_settings );
...
settingResourceMapInt.put( SCanvasConstants.LOCALE_ERASER_SETTING_TITLE,
                            R.string.eraser_settings);
settingResourceMapInt.put( SCanvasConstants. LOCALE_ERASER_SETTING_CLEARALL,
                            R.string.clear_all);
```

```
...
settingResourceMapInt.put( SCanvasConstants.LOCALE_TEXT_SETTING_TITLE,
                           R.string.text_settings );
settingResourceMapInt.put( SCanvasConstants.LOCALE_TEXT_SETTING_TAB_FONT,
                           R.string.text_settings_tab_font);
...

// User-added font support
HashMap<String,String> settingResourceMapString = new HashMap<String, String>();
settingResourceMapString.put( SCanvasConstants.USER_FONT_PATH1, "fonts/chococooky.ttf" );
settingResourceMapString.put( SCanvasConstants.USER_FONT_PATH2, "fonts/rosemary.ttf" );
```

[Code 6] Initialization for SettingView

To reflect the setting values that have been changed in the settings window onto an SCanvasView, the SCanvasView must be connected to the settings window.
You can easily create a SettingView by using the createSettingView method of the SCanvasView as shown below.

```
...

mSCanvas.createSettingView( mLayoutContainer,
                            settingResourceMapInt,
                            settingResourceMapString );
...
```

[Code 7] Connecting Settings Pop-up Window to SCanvasView

The method of the SCanvasView is used to display the settings pop-up window on the screen. The showSettingView method can be used to close the pop-up by setting it to false.

```
// Displaying a Pen Settings pop-up window.
mSCanvas.toggleShowSettingView(SCanvasConstants.SCANVAS_SETTINGVIEW_PEN);

// Closing the Pen Settings pop-up window.
mSCanvas.showSettingView(SCanvasConstants.SCANVAS_SETTINGVIEW_PEN, false);
```

[Code 8] Displaying and closing a Settings pop-up window

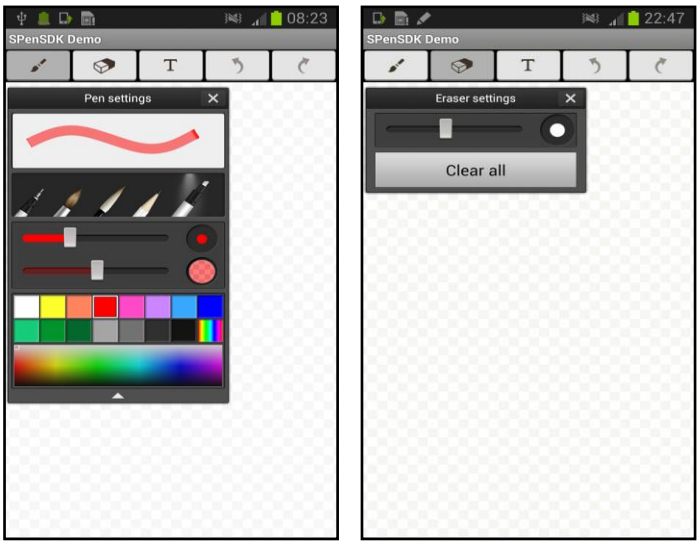
You can use the isSettingViewVisible method, as shown below, to check whether each Settings Pop-up window is active on the screen or not. When using this method, use the parameters that correspond to the mode you wish to check.

```
// Check if the Pen Settings pop-up window is displayed.
If ( mSCanvas.isSettingViewVisible( SCanvasConstants.SCANVAS_SETTINGVIEW_PEN ) )
{
}
// Check if the Eraser Settings pop-up window is displayed.
If ( mSCanvas.isSettingViewVisible( SCanvasConstants.SCANVAS_SETTINGVIEW_ERASER PEN ) )
{
}
// Check if the Text Settings pop-up window is displayed.
If ( mSCanvas.isSettingViewVisible( SCanvasConstants.SCANVAS_SETTINGVIEW_TEXT ) )
{
}
```

[Code 9] Check the status of Pen, Eraser, and Text Settings pop-up windows

In the Pen Settings pop-up window, the user can choose one of the five pen types (Solid, Normal brush, Chinese brush, Pencil, Highlighter) and adjust the pen color, thickness, and opacity. How the selected pen looks can be previewed in the preview area at the top of the pop-up window.

In the Eraser Settings pop-up window, you can adjust the thickness of the eraser. You can erase all the drawings on the screen by pressing the "Clear All" button.



[Figure 5] Pen and Eraser Settings pop-up windows

In the Text Settings pop-up window, the user can easily change the font, font size, style, and color of the text.

[Figure 6] Text Settings pop-up window

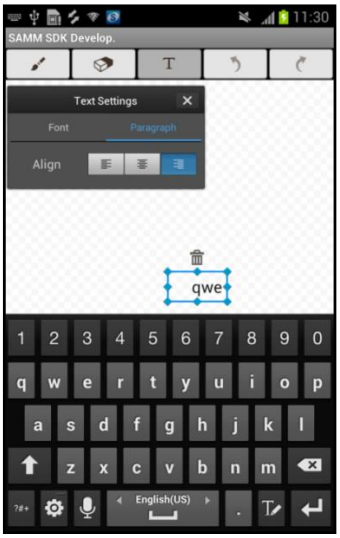
The following code can be used to display Pen, Eraser, and Text Settings pop-up windows on the screen.

```
// Displaying a Pen Settings pop-up window if the current mode is Pen Mode.
if(mSCanvas.getCanvasMode()==SCanvasConstants.SCANVAS_MODE_INPUT_PEN)
{
    mSCanvas.toggleShowSettingView(SCanvasConstants.SCANVAS_SETTINGVIEW_PEN);
}

// Displaying an Eraser Settings pop-up window if the current mode is Eraser Mode.
if(mSCanvas.getCanvasMode()==SCanvasConstants.SCANVAS_MODE_INPUT_ERASER)
{
    mSCanvas.toggleShowSettingView(SCanvasConstants.SCANVAS_SETTINGVIEW_ERASER);
}

// Displaying a Text Settings pop-up window if the current mode is Text Mode.
if(mSCanvas.getCanvasMode()==SCanvasConstants.SCANVAS_MODE_INPUT_TEXT)
{
    mSCanvas.toggleShowSettingView(SCanvasConstants.SCANVAS_SETTINGVIEW_TEXT);
}
```

[Code 10] Checking the current mode and showing pop-up window

Basically, the settings values adjusted in each settings pop-up window are reflected in the drawing in SCanvasView without any additional code.

For example, if you touch an empty drawing area in Text Mode, an adjustable text box appears on the screen. You can adjust the alignment of the text by changing the settings values (align left/middle/right) in the Text Settings pop-up window.
By using the "Clear all" feature in the Eraser Settings pop-up window, you can delete everything on an SCanvasView.



[Figure 7] Applying settings values from Text Settings Pop-up window

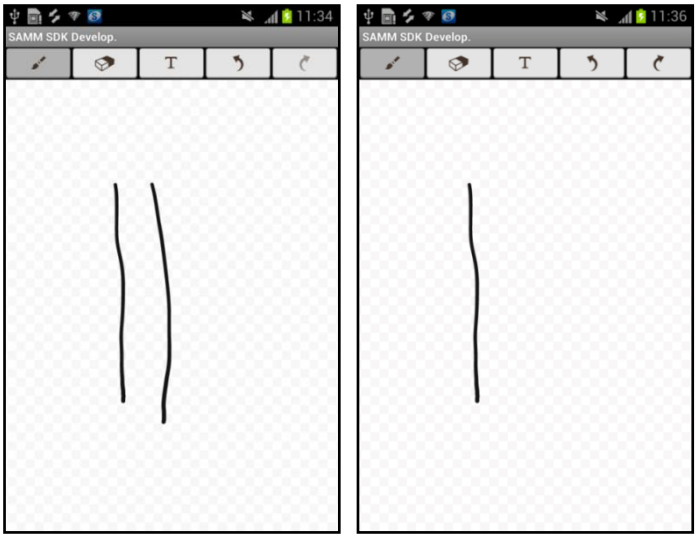
3-2-4. Undo/Redo

You can undo and redo operations by using simple methods in the SCanvasView class.

```
mSCanvas.undo( );   // Undo

mSCanvas.redo( );   // Redo
```

[Code 11] Undo/Redo

The undo and redo history features in the example "Editor: Basic UI" are run by using the undo and redo methods in an SCanvasView when the corresponding button is pressed.
The maximum number of undo and redo operations is 30.



[Figure 8] Undoing an operation

You can check whether the undo and redo history features are available by using the isUndoable method and isRedoable method.
In the code below, the corresponding methods are used to activate or deactivate the undo and redo buttons.

```
// Check whether undo can be performed and activate or deactivate the Undo button
mUndoBtn.setEnabled( mSCanvas.isUndoable( ) );
// Check whether redo can be performed and activate or deactivate the Redo button
mRedoBtn.setEnabled( mSCanvas.isRedoable( ) );
```

[Code 12] Change the button status depending on whether undo and redo can be performed

Furthermore, you can handle other variables that occur in the drawing process by registering HistoryUpdateListener in the SCanvasView class. Then, you can use this listener to check whether undo and redo can be performed while drawing is in progress.

```
// Register listener in SCanvasView.
mSCanvas.setHistoryUpdateListener(new HistoryUpdateListener() {
        @Override
        public void onHistoryChanged(boolean undoable, boolean redoable) {
                mUndoBtn.setEnabled( undoable );   // Check whether undo can be performed.
                mRedoBtn.setEnabled( redoable );    // Check whether redo can be performed.
        }
});
```

[Code 13] Use HistoryUpdateListener to check whether undo and redo can be performed

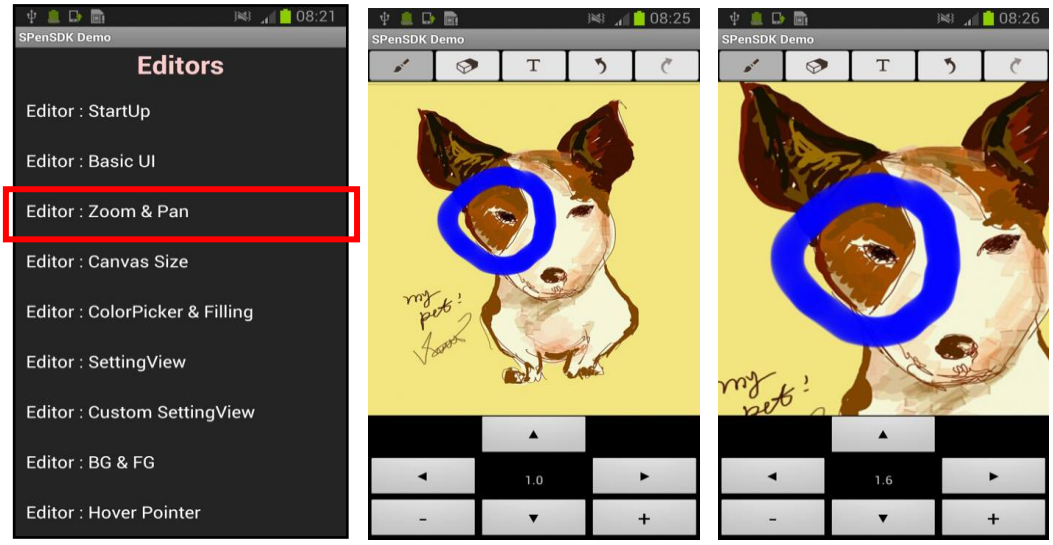
3-2-5. Zoom In / Zoom Out

The SCanvasView drawing area can be zoomed in or out by up to 50 times from the original size.

By using the "Editor: Zoom & Pan" code provided in the SDK, you can use zooming and see how it works.

```
mSCanvas.setCanvasZoomScale( 2.0f );   // Zoom to 200% of original size.
mSCanvas.setCanvasZoomScale( 1.0f );   // Restore to original size.
```

[Code 14] Zoom in and zoom out



[Figure 9] Zooming in

3-2-6. Panning

SCanvasView can be used pan the view, which moves the current drawing area according to x- and y-axis coordinates.

You can refer to the example code below to see how panning works.

```
mSCanvas.panBySCanvas(0, 30);        // Move 30 pixels toward the top of the canvas.
mSCanvas.panBySCanvas(0, -30);       // Move 30 pixels toward the bottom of the canvas.
mSCanvas.panBySCanvas(30, 0);        // Move 30 pixels toward the left of the canvas.
mSCanvas.panBySCanvas(-30, 0);       // Move 30 pixels toward the right of the canvas.
```

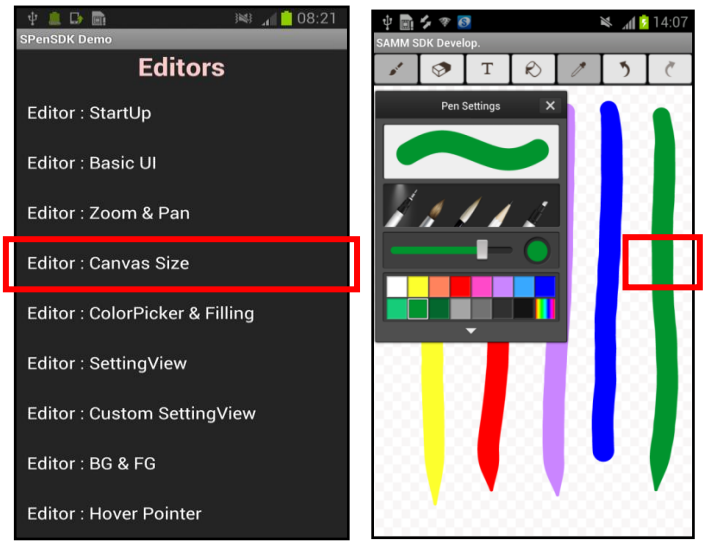
[Code 35] Panning the drawing area

Keep in mind that panning works by pixel units regardless of the current zoom status of the screen.

For example, when you pan 1 pixel at the 200% zoom level, it does not pan by 2 pixels, but rather pans by only 1 pixel.

3-2-7. ColorPicker & Filling

The ColorPickerColorChangeListener listener provided in the S Pen SDK captures color information from the coordinates of the current drawing area where a touch event occurs.
To use this listener, it must be registered by using the setColorPickerColorChangeListener method of SCanvasView.
The ColorPickerColorChangeListener contains the onColorPickerColorChanged method.
It is called when a touch event occurs and returns the color information from the coordinates.



[Figure 10] Screen showing color information being captured

The example code below shows how to do this.

```
mSCanvas.setColorPickerColorChangeListener(new ColorPickerColorChangeListener(){
        @Override
        public void onColorPickerColorChanged(int nColor) {
            // Sends back the captured color information in the nColor variable.
        }
};

mSCanvas.setColorPickerMode ( true );
```

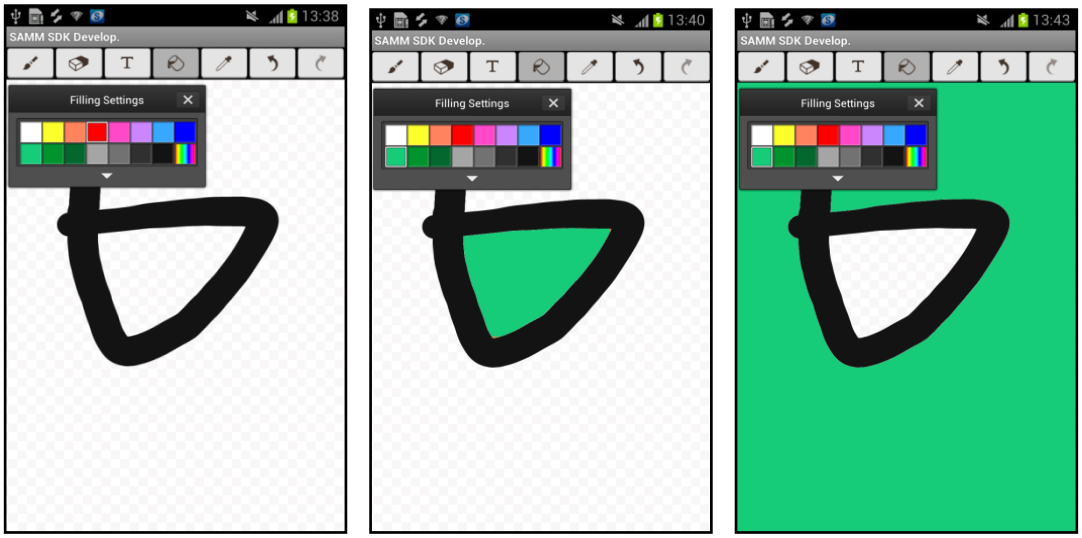
[Code 4] Examples of color information extraction using ColorPickerColorChangeListener

In S Pen SDK version 2.2, a new feature for filling a closed area with color while drawing in SCanvasView's drawing area has been added.
To use this feature, the current canvas mode value must be switched to SCANVAS_MODE_INPUT_FILLING as shown in the code snippet below.

```
mSCanvas.setCanvasMode( SCanvasConstants.SCANVAS_MODE_INPUT_FILLING );
```

[Code 5] Switching to Color Fill Mode

In Color Fill Mode, when you tap on a closed space, the space is filled with the currently selected color.



[Figure 11] Color Fill Mode screen

By using the code below, you can use the ColorPicker function when the SCanvasView is in Pen Input, Text Input, or Color Fill Mode, and reflect the picked color in the current mode.

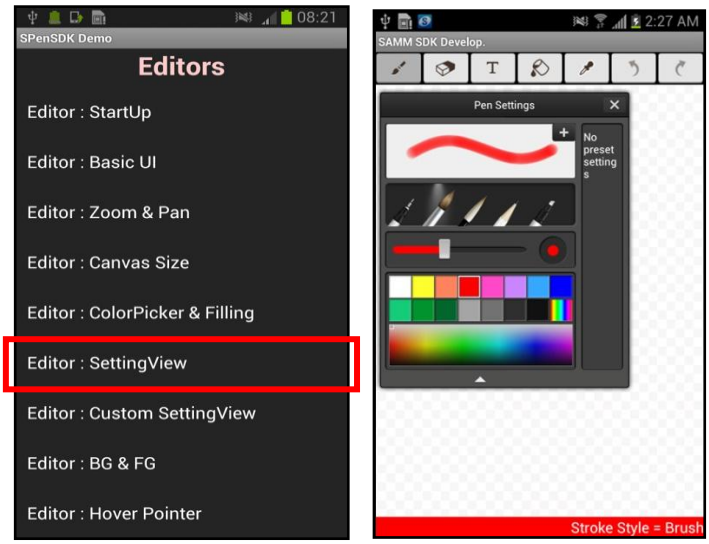
```
mSCanvas.setColorPickerColorChangeListener( new    ColorPickerColorChangeListener()
{
        @Override
        public void onColorPickerColorChanged(int nColor)
        {
                int nCurMode = mSCanvas.getCanvasMode();
                // When Selecting Pen and activating ColorPicker mode.
                if(nCurMode==SCanvasConstants.SCANVAS_MODE_INPUT_PEN)
                {
```

```
                    SettingStrokeInfo strokeInfo = mSCanvas.getSettingViewStrokeInfo();

                    if(strokeInfo != null)
                    {
                        strokeInfo.setStrokeColor(nColor);
                        mSCanvas.setSettingViewStrokeInfo(strokeInfo);
                    }
                }
                else if(nCurMode==SCanvasConstants.SCANVAS_MODE_INPUT_ERASER)
                {
                    // Setting it to do nothing.
                }
                else if(nCurMode==SCanvasConstants.SCANVAS_MODE_INPUT_TEXT)
                {
                    // When Selecting Text and activating ColorPicker mode.
                    SettingTextInfo textInfo = mSCanvas.getSettingViewTextInfo();

                    if(textInfo != null)
                    {
                        textInfo.setTextColor(nColor);
                        mSCanvas.setSettingViewTextInfo(textInfo);
                    }
                }
                else if(nCurMode==SCanvasConstants.SCANVAS_MODE_INPUT_FILLING)
                {
                    // When Selecting Filling and activating ColorPicker mode.
                    SettingFillingInfo fillingInfo = mSCanvas.getSettingViewFillingInfo();

                    if(fillingInfo != null)
                    {
                        fillingInfo.setFillingColor( nColor );
                        mSCanvas.setSettingViewFillingInfo( fillingInfo );
                    }
                }
            }
} );
```

[Code 68] Applying picked color information to a different mode

3-2-8. Settings Information Listener and Mini Settings Window

The S pen SDK provides various classes, such as SettingTextInfo, SettingStrokeInfo, SettingFillingInfo, and listeners, such as SettingTextChangeListener,SettingStrokeChangeListener, SettingFillingChangeListener, for retrieving and transferring settings from Pen, Eraser, Text Settings pop-up windows.



[Figure 12] Run screen of Setting View

The SettingStrokeInfo class manages parameters related to Pen settings. You can read or write parameter values that contain Pen settings information, such as Pen type, thickness, color, and opacity.

To use SettingStrokeInfo class, you can create a class manually by using the class generator or use the current SettingStrokeInfo class object, which is automatically generated and connected by SCanvasView.

```
SettingStrokeInfo   strokeInfo = mSCanvas.getSettingViewStrokeInfo();
```

[Code 19] Initializing the SettingStrokeInfo class

By using the SettingStrokeInfo class object, which is set for developer's purposes with the setSettingStrokeInfo method of SCanvasView as shown below, it can be used as a substitute for an existing SettingStrokeInfo, which is currently connected to the Pen Settings pop-up window of the canvas.

```
mSCanvas.setSettingViewStrokeInfo( strokeInfo );
```

[Code 20] Replacing an existing SettingStrokeInfo object

While drawing, various changes in Pen and Eraser settings can be checked by using the SettingStrokeChangeListener.

This listener can be registered by using the setSettingStrokeChangeListener method in an SCanvasView.

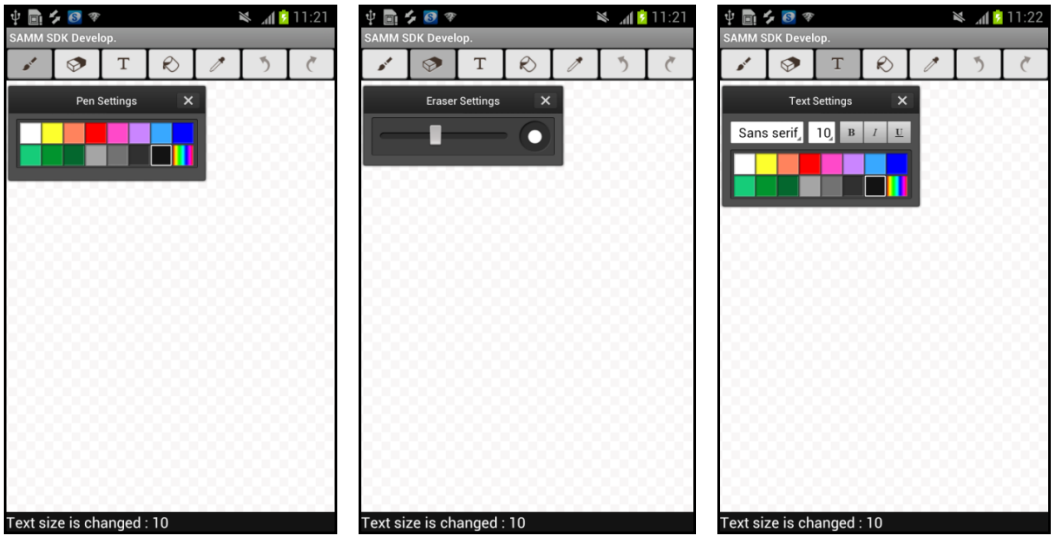
You can override the methods provided by this listener to define actions to perform when property values, such as pen type, thickness, color, and opacity, are changed.

```
mSCanvas.setSettingStrokeChangeListener(new SettingStrokeChangeListener()
{
        @Override
        public void onClearAll(boolean bClearAllCompleted)
        {
            // Called when the "Clear all" button is pressed in an Eraser Settings window.
        }

        @Override
        public void onEraserWidthChanged(int eraserWidth)
        {
            // Called when the Eraser width changes.
        }

        @Override
        public void onStrokeColorChanged(int strokeColor)
        {
            // Called when stroke color has been changed.
        }

        @Override
        public void onStrokeStyleChanged(int strokeStyle)
        {
            // Called when stroke style (type) changes.
            if (strokeStyle == SObjectStroke.SAMM_STROKE_STYLE_PENCIL)
            else if (strokeStyle == SObjectStroke.SAMM_STROKE_STYLE_BRUSH)
            else if (strokeStyle == SObjectStroke.SAMM_STROKE_STYLE_CHINESE_BRUSH)
```

```
                else if (strokeStyle == SObjectStroke.SAMM_STROKE_STYLE_CRAYON)
                else if (strokeStyle == SObjectStroke.SAMM_STROKE_STYLE_MARKER)
                else if (strokeStyle == SObjectStroke.SAMM_STROKE_STYLE_ERASER)
            }
            @Override
            public void onStrokeWidthChanged(int strokeWidth)
            {
                // Called when stroke thickness changes.
            }

            @Override
            public void onStrokeAlphaChanged(int strokeAlpha)
            {
                // Called when stroke opacity (alpha) changes.
            }
);
```

[Code 21] Examples of SettingStrokeChangeListener settings

In S Pen SDK 2.2, a Mini Settings pop-up window has also been added as shown below.



[Figure 13] Mini Settings pop-up window

To show the Mini Settings pop-up window, you must use the setSettingViewSizeOption method of the SCanvasView and use the SCanvasConstants.SCANVAS_SETTINGVIEW_SIZE_MINI enumeration, which is the factor value that corresponds to the Mini Settings pop-up window, to adjust the size of the pop-up window.

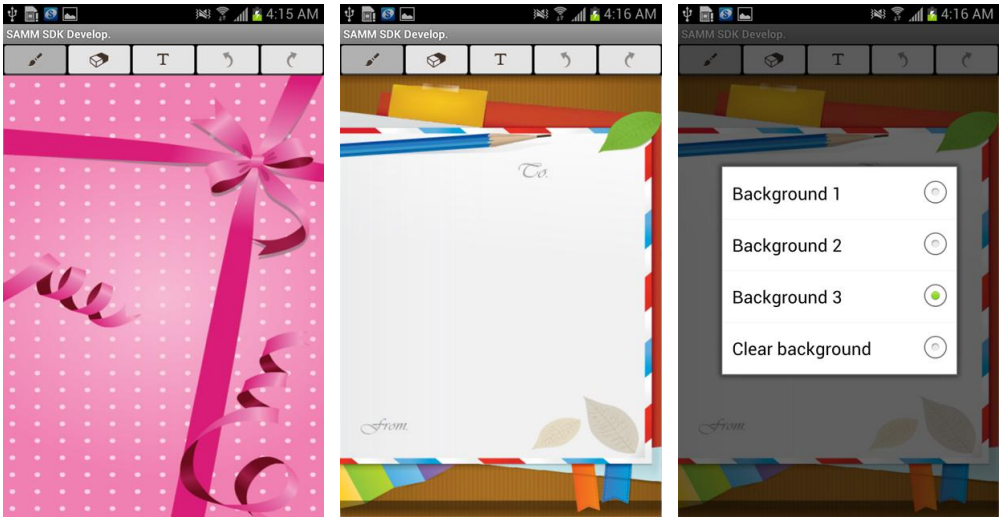
```
// Adjust size.
mSCanvas.setSettingViewSizeOption( SCanvasConstants.SCANVAS_SETTINGVIEW_PEN,
                                   SCanvasConstants.SCANVAS_SETTINGVIEW_SIZE_MINI );

// Displaying a Pen Settings pop-up window.
mSCanvas.toggleShowSettingView(SCanvasConstants.SCANVAS_SETTINGVIEW_PEN);
```

[Code 22] Adjusting SettingView pop-up window size to Mini

3-2-9. Setting Background Images

You can easily set a background image for the drawing area by using the setBGImage method of the SCanvasView.

You can set an image as the background for the drawing area by using a Bitmap object or the path to an image file (JPG/PNG) as the parameter.



[Code 14] Changing background images

```
Bitmap bm = null;
Bitmap bm = BitmapFactory.decodeResource( mContext.getResources(), R.drawable.letter_bg );
mSCanvas.setBGImage( bm );
```

[Code 23] Setting background images

It is recommended that the size of the image matches the size of the SCanvasView. Otherwise, the image may seem distorted as it is stretched or reduced to match the size of the SCanvasView.

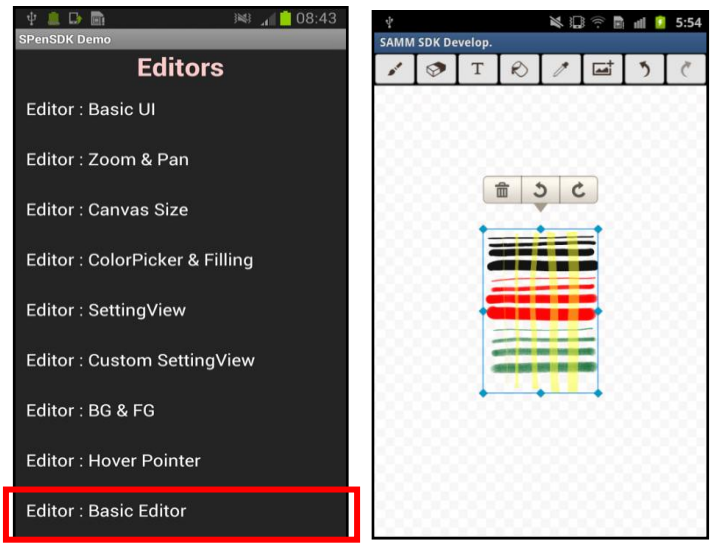
3-2-10. Adding Images to a Canvas Area

SCanvasView can also add image objects that inherit from SObject to the drawing area.
To do so, you must create an SObjectImage object and set it as a variable in the insertSAMMImage method.

The first factor in the insertSAMMImage method is the image object you wish to insert, and the second factor is the flag that decides whether the image is displayed in the selected state after being inserted on the canvas.

```
RectF rectF = getDefaultImageRect(imagePath);

// Create an image object.
SObjectImage sImageObject = new SObjectImage();      // Create an SObjectImage object.
sImageObject.setRect( rectF );                       // Select the size of the object to insert.
sImageObject.setImagePath( imagePath );              // Designate an image file path.
// Set the image object.
if( mSCanvas.insertSAMMImage(sImageObject, true) ) {
...
}
```

[Code 24] Adding an image object to the drawing area



[Figure 15] Run screen for inserting an image

The SObjectUpdateListener listener is called when an image or text object is added, deleted, selected, or modified.
This listener is included in the com.samsung.spensdk.applistener package.

```
mSCanvas.setSObjectUpdateListener( new SObjectUpdateListener() ) {
        // Called when an object (stroke, image, text) input is completed.
        @Override
        public void onSObjectInserted(SObject sObject, boolean byUndo, boolean byRedo) {
            ...
        }
        // Called when an object (stroke, image, text) is deleted.
        @Override
        public void onSObjectDeleted( SObject sObject,
                                      boolean byUndo, boolean byRedo,
                                      boolean bFreeMemory ){
        ...
        }
        // Called when an object (stroke, image, text) changes.
        @Override
        public void onSObjectChanged( SObject sObject,
                                       boolean byUndo, boolean byRedo){
        ...
        }
        // Called when stroke input is completed to decide
        // whether to cancel or maintain the input.
        @Override
        public boolean onSObjectStrokeInserting(SObjectStroke sObjectStroke) {
        ...
        }
        // Called when an object selection (stroke, image, text) changes.
        @Override
        public void onSObjectSelected(SObject sObject, boolean bSelected) {
        ...
        }
        // Called when an object (stroke, image, text) list is deleted.
        @Override
        Public void onSObjectClearAll(boolean bFreeMemory){
        ...
        }
};
```

[Code 25] Using the SObjectUpdateListener listener

When an object that inherits from SObject, including an image object, is added to the drawing area and selected, the copy, cut, and paste clipboard actions can be used. First, check if there is an SObject object that was copied or cut onto the clipboard using the isClipboardSObjectListExist method of SCanvasView. Next, you can use the getClipboardSObjectListType method to check the type of the SObject on the clipboard. (SObject.SOBJECT_LIST_TYPE_IMAGE for image objects.)
You can use the copySelectedSObjectList and cutSelectedSObjectList methods to insert an SObject onto the clipboard and use the pasteClipboardSObjectList method to paste the SObject onto the drawing area.

```
// Copy an SObject onto the clipboard.
void copySelectedObject(){
        if(!mSCanvas.copySelectedSObjectList(bResetClipboard)) {
        }
        Else {
        }
}

// Cut an SObject onto the clipboard.
void cutSelectedObject() {
        if(!mSCanvas.cutSelectedSObjectList(bResetClipboard)) {
        }
        else {
        }
}

// Initialize the clipboard.
void clearClipboardObject() {
    mSCanvas.clearClipboardSObjectList();
}

// Paste an SObject list from the clipboard to the canvas.
void pasteClipboardObject(int nEventPositionX, int nEventPositionY) {
     if(!mSCanvas.pasteClipboardSObjectList(bSelectObject, nEventPositionX, nEventPositionY)) {
     }
}
```

[Code 26] Clipboard initialization and copy/paste/cut command execution

3-3. SAMM Library

SAMM is an abbreviation for **S**amsung **A**nimated **M**ultimedia **M**essage. It refers to the common data format that Samsung has defined for creating messages, memos, and notes using writing, text and various multimedia elements.

The SAMM library is included in SCanvasView. It can play the contents written on the canvas in an animation as well as maintain compatibility with data created in other applications that use the SAMM library (included in the SDK).

The types of data that can be handled by the SAMM library are as follows.

- **Object Data** - Writing data that can exist in the form of an object, such as writing (stroke), images, text, and videos (undo or redo data)
- **Settings Data** - Data that is used generally without any relationship to any specific object, such as a title, tag, attachment file, favorite, background color, and background music

For example, a stroke (writing) object has defined style values, such as pencil, crayon, and highlight. An image object has defined style values, such as normal image, stamp, and animated icon.
By using these style values defined in the SAMM library, you can maintain data compatibility between applications.

If you want to use a specific style value that is not defined, you can use extra, or customized, data.

3-3-1. Initialization and Closing

Each object in the SAMM library, such as writing, images, and text, has its own location (area). This area is set relative to the screen size or the writing area. In other words, the screen size, or the writing area, must be set initially so that the location of object data can be saved with its relative position inside the screen.

Therefore, the SCanvasView-related initialization must be performed in the SCanvasInitializeListener, which is called when the size of the writing area is decided. (An error occurs when initialized in the onCreate method or other listeners.)

```
// Register a listener in SCanvasView.
mSCanvas.setSCanvasInitializeListener( new SCanvasInitializeListener()
{
        @Override
        public void onInitialized()
        {
            // SCanvasView initialization.
            ...
        }
} );
```

[Code 27] SCanvasView SAMM library Initialization

Also, temporary files and data used to save or load a file to and from the library using the closeSCanvasView method must be deleted when closing the program.

```
// Call closeSCanvasView () when the SAMM library is no longer used.
@Override
protected void onDestroy()
{
        super.onDestroy();
        // Release SCanvasView resources
        if( !mSCanvas.closeSCanvasView() )
            Log.e(TAG, "Fail to close SCanvasView");
}
```

[Code 28] Closing the SAMM library

3-3-2. Basic Animation Function

When using the SCanvasView class, actions on the canvas, such as drawings, input text and images, are saved internally in a data format defined by the SAMM library. They can be played in the View screen as an animation later on.

To use animation, you must first change the animation mode status value to "Enable" using the SCanvasView method. This is because the initial setting for the animation mode in an SCanvasView is disabled. Once the animation mode is enabled, drawing actions can no longer be performed and only animation actions can be performed. Drawing actions can be performed again when the animation mode is disabled.

Before playing an animation, the background music option can be set using SOptionSCanvas. The selected options can be applied to the canvas using the setOption(SOptionSCanvas

canvasOption) method. The options include background music on or off, background music repeat, and background music volume.

```
SOptionSCanvas canvasOption = new SOptionSCanvas();
canvasOption.mPlayOption. setInvisibleBGImageAnimationOption( false );

// Background music playback options setting.
canvasOption.mPlayOption. setPlayBGAudioOption (true);

// Background music repeat options setting.
canvasOption.mPlayOption.setRepeatBGAudioOption(false);

// Stop background music when the animation ends options setting.
canvasOption.mPlayOption.setStopBGAudioOption( true );

// Background music volume options setting.
if( !canvasOption.mPlayOption.setBGAudioVolume( 1.0f   ) )
      return;

// Animation speed option setting.
if( !canvasOption.mPlayOption.setAnimationSpeed( SOptionPlay.ANIMATION_SPEED_AUTO ) )
   return;

// Apply options to SCanvasView.
if( !mSCanvas.setOption( canvasOption ) )
   return;
```

[Code 29] Setting and applying background music options

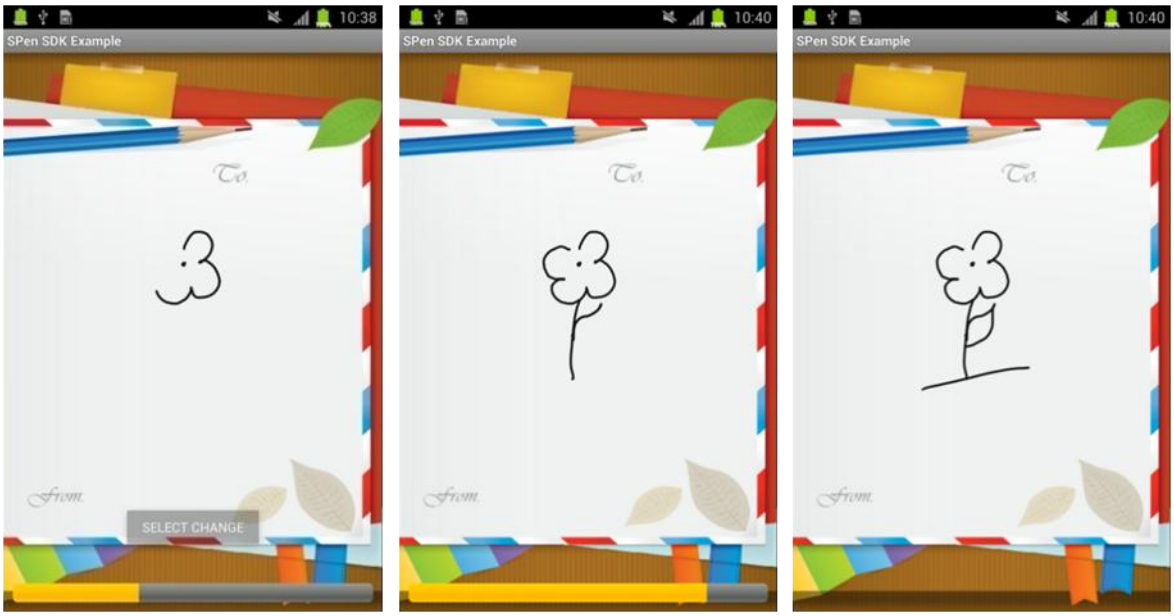
For more information about these options, refer to the SOptionPlay and SOptionSAMM classes in the API Reference document included in the SDK.

Now, you can play the animation saved in SCanvasView using the code below.

```
// Start animation playback.
mSCanvas.doAnimationStart( );
```

[Code 30] Start animation playback

The example included in the SDK uses a separate activity to show the animation. The figure below shows each step of the animation playback screen in the example application.



[Figure 16] Animation playback screen

The progress of animation playback can be acquired using the getAnimationState() method. As shown below, additional actions can be applied depending on the progress.

```
// Start or stop animation playback.
void animationPlayOrPause( )
{
	int   nAnimationState = mSCanvas.getAnimationState( );
	if( nAnimationState==SAMMLibConstants.ANIMATION_STATE_ON_STOP ) {
		SOptionSCanvas canvasOption = new SOptionSCanvas();
		setPlayOption(canvasOption);
		mSCanvas.doAnimationStart();
	}
	else if( nAnimationState==SAMMLibConstants.ANIMATION_STATE_ON_PAUSED ) {
		mSCanvas.doAnimationResume ( );
	}
	else if( nAnimationState==SAMMLibConstants.ANIMATION_STATE_ON_RUNNING ) {
		mSCanvas.doAnimationPause( );
	}
}
```

[Code 31] Checking animation playback progress

When the animation playback has completed, or the progress has changed, the AnimationProcessListener listener is called. By using the onChangeProgress method of the listener, you can get the animation playback complete status or current progress as an integer

between 0 and 100.

This listener is included in the com.samsung.spensdk.applistener package.

```
mSCanvas.setAnimationProcessListener ( playCompleteListener );

AnimationProcessListener   playCompleteListener = new AnimationProcessListener() {
        @Override
        public void onPlayComplete() {
                ...
        }

        @Override
        public void onChangeProgress( int nProgress ) {
                ...
        }
};
```

[Code 32] Checking animation playback progress using AnimationProcessListener

3-3-3. Setting Canvas Background Data

By using the S Pen SDK and the SAMM library, you can use various non-image data as background data.

The examples below show that string-format memo data as well as various properties, such as background color and background music, can be added.

```
SDataPageMemo pageMemo = new SDataPageMemo();
pageMemo.setText( tmpStr );
mSCanvas.setPageMemo( pageMemo, 0 );
```

[Code 33] Adding a page memo

You can designate the background color as shown below.

```
// Set the background color to white.
nSetColor = 0xFFFFFFFF;
mSCanvas.setBGColor( nSetColor );
```

[Code 34] Setting the background color

Also, you can set background music.

```
// Set the background music.
mSCanvas.setBGAudioFile( strBGAudioFileName );

// Remove the background music.
mSCanvas.clearBGAudio();
```

[Code 35] Setting background music

3-3-4. Identifiers for Data Compatibility

The SAMM library provides an API for setting the Application ID so that you can identify if a file is a SAMM file created by your application or another application.

This Application ID shows which application created the SAMM file. It consists of the application name, major and minor version, and patch name.

For example, if the Application ID is "MyMemo 1.2 rev", the application is called "MyMemo", its major version is 1, minor version is 2, and the patch name is "rev".

Keep in mind that when saving data as a SAMM file using the S Pen SDK, an error occurs if the Application ID is not set.

Also, the Application ID of the current application can be checked by using the getAppID method of SCanvasView.

```
private String APPLICATION_ID_NAME = "SDK Sample Application";
private int APPLICATION_ID_VERSION_MAJOR = 2;
private int APPLICATION_ID_VERSION_MINOR = 2;
private String APPLICATION_ID_VERSION_PATCHNAME = "Debug";
...
mSCanvas.setSCanvasInitializeListener(new SCanvasInitializeListener() {
        @Override
        public void onInitialized() {
                ...
                // Application ID setting.
                If( !mSCanvas.setAppID( APPLICATION_ID_NAME,
                                        APPLICATION_ID_VERSION_MAJOR,
                                        APPLICATION_ID_VERSION_MINOR,
                                        APPLICATION_ID_VERSION_PATCHNAME)) {
                }
}
```

[Code 36] Application ID setting

When loading a SAMM file created by another application, a different Application ID is sent back. You can use it to check if the data provided by SAMM is compatible with your application or not.

3-3-5. Entering Additional Information

In SCanvasView, you can enter various types of additional information in addition to drawing information, such as string data and tags.

```
// Adding tags.
mSCanvas.addTag( "MyTag" );

// Reading tags.
String[] tagArray = mSCanvas.getTags();

// Deleting tags.
mSCanvas.removeTag( "MyTag" );

// Setting the title.
mSCanvas.setTitle(title);

// Setting extra data.
mSCanvas.putExtra( "MyExtraDataKey", "MyExtraData" );

// Reading extra data.
String   extra
         =   mSCanvas.getStringExtra(( "MyExtraDataKey", null   );
```

[Code 37] Adding basic information provided in SDK

Extra data can be useful when you want to save arbitrary, customized data that is incompatible with other applications.

Hypertext data, such as a URL, can be saved using the setHypertext method, and can be retrieved using the getHyperText method.

You can also save latitude and longitude information using the setGeoTag method, and the information can be retrieved using the getGeoTagLatitude and getGeoTagLongitude methods.

Also, you can attach external files as shown below.

```
SDataAttachFile attachData = new SDataAttachFile();
attachData.setFileData( strFileName, "S Pen Example Selected File" );
```

[Code 38] Attaching a file

You can use the getAttachedFileData method to check the list of all files attached in the current canvas.

```
// Retrieve the number of files attached.
int   nAttachFileNum = mSCanvas.getAttachedFileNum();

for( int i=0; i<nAttachFileNum; i++ )
{
        SDataAttachFile attachData = mSCanvas.getAttachedFileData( i );
        if( attachData == null )
        {
            // When there is no attachment.
            return;
        }
        String   strPath = attachData.getFilePath();
        String   strDescription = attachData.getFileDescription();
        BitmapDrawable   icon = new BitmapDrawable( getResources(),
                                                    attachData.getFileIconBitmap() );
}
```

[Code 39] Retrieving the list of attached files

3-3-6. Saving and Loading

To save the drawing in the current screen or load external images to the screen, you can use the getCanvasBitmap, getData, and setData methods, which are provided in the SCanvasView class.

Also, the ExampleUtils class, used in the following example code, can be used to save drawings into a file. You can see how it is done by referring to the first example.

In the figure below, the file is categorized as an image or a background when saved or loaded. When loaded as an image, you can use the eraser while drawing. When loaded as a background, the eraser cannot be used.

```
// Saving
public boolean saveCanvasImage( boolean bSaveOnlyForegroundImage ) {
        Bitmap bmCanvas = mSCanvas.getCanvasBitmap( bSaveOnlyForegroundImage );

        if (!(mFolder.exists()))
          if(!mFolder.mkdirs()) return false;

        String savePath = mFolder.getPath() + '/'
              + ExampleUtils.getUniqueFilename(mFolder, "image", SAVED_FILE_EXTENSION);

        return saveBitmapPNG(savePath, bmCanvas);
}
// Loading
private boolean loadCanvasImage(String fileName, boolean loadAsForegroundImage) {
{
        String loadPath = mFolder.getPath( ) + "/" + fileName;
        byte[] buffer = ExampleUtils.readBytedata(loadPath);
        if(loadAsForegroundImage)  {
                Bitmap bmForeground = BitmapFactory.decodeFile(loadPath);

                If ( bmForeground == null ) return false;

                int nWidth = mSCanvas.getWidth();
                int nHeight = mSCanvas.getHeight();
                bmForeground = Bitmap.createScaledBitmap(bmForeground,
                                                         nWidth, nHeight,
                                                         true);
                return mSCanvas.setClearImageBitmap(bmForeground);
        }
        else {
                return mSCanvas.setBGImagePath(loadPath);
        }
}
```

[Code 40] Saving and loading

You can use the saveSAMMFile method to save the visible drawing area and writing data. And you can use the loadSAMMFile method to extract the saved writing data and update it on the screen.

The PNG and JPG image formats are supported when saving in the SAMM data format. When saving only data, the proprietary AMS format is used.

```
private File mFolder = null;
...
File sdcard_path = Environment.getExternalStorageDirectory();

mFolder =    new File(sdcard_path, DEFAULT_APP_IMAGEDATA_DIRECTORY);
...
private boolean saveSAMMFile()
{
        String savePath = mFolder.getPath() + '/'
                + ExampleUtils.getUniqueFilename( mFolder,
                                                  "SAMM",
                                                  SAVED_FILE_EXTENSION );

        return mSCanvas.saveSAMMFile(savePath);
}
```

[Code 41] Saving using SAMM Library

```
private boolean loadSAMMFile(String fileName)
{
        String loadPath = mFolder.getPath() + '/' + fileName;

        return mSCanvas.loadSAMMFile(loadPath, true, true);
}
```

[Code 42] Loading using SAMM Library

3-3-7. FileProcessListener Listener

The FileProcessListener listener provides two methods, shown below, to detect the file loading status.

- onChangeProgress(int nProgress) – Called when the loading progress rate changes
- onLoadComplete(boolean bLoadResult) – Called when loading completes

This listener is included in the com.samsung.spensdk.applistener package.

```
// Registering a listener.

mSCanvas.setFileProcessListener( new FileProcessListener()
{
        // Method called back for changes in the loading progress.

         @Override
         public void onChangeProgress(int nProgress)
         {
           // Progress rate is checked with nProgress as a number between 0~100.
           ...
         }

         @Override
         public void onLoadComplete(boolean bLoadResult)
         {
           ...
         }
};
```

[Code 43] Using FileProcessListener

3-3-8. Canvas Data Resizing

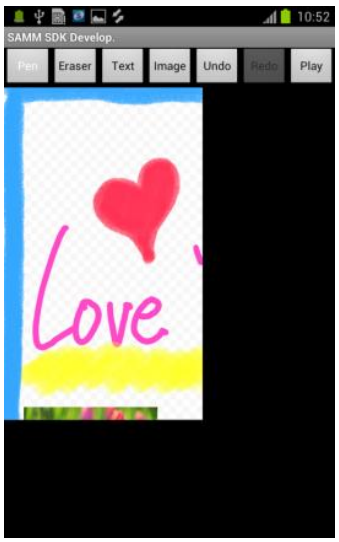
When SCanvasView data created on a WXGA (resolution 1280x800) screen is loaded on a smaller WVGA (resolution 800x480) screen, the canvas size and alignment can be adjusted using a method provided in SCanvasview.

You need to apply SOptionSAMM constant values that provide various options for saving and loading SAMM files.

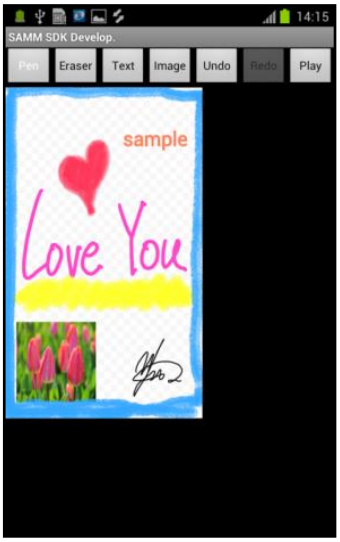
```
// Canvas resizing options.
SOptionSAMM.SAMM_LOAD_OPTION_CONVERT_SIZE_FITTOSIZE - Fit to current canvas size.
SOptionSAMM.SAMM_LOAD_OPTION_CONVERT_SIZE_FITINSIDE - Fit inside current canvas.
SOptionSAMM.SAMM_LOAD_OPTION_CONVERT_SIZE_ORIGINAL - Maintain original canvas size.

// Canvas alignment options.
SAMMLibConstants.SAMM_ALIGN_NORMAL     - Align to left/top.
SAMMLibConstants.SAMM_ALIGN_CENTER     - Align to center.
SAMMLibConstants.SAMM_ALIGN_OPPOSITE   - Align to right/bottom.
```

[Code 44] Canvas resizing and alignment options

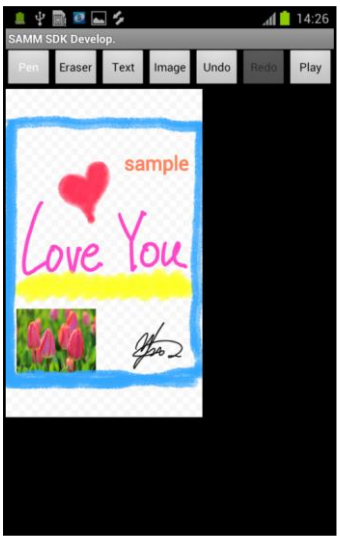
Use the SOptionSCanvas class and methods to set a new size and alignment option for the SCanvasView.



[Figure 17] Maintain original size



Fit to current canvas size



Fit to current canvas size + Align to center

```
SOptionSCanvas canvasOption = mSCanvas.getOption();

canvasOption.mSaveOption.setConvertCanvasSizeOption
                        ( SOptionSAMM.SAMM_LOAD_OPTION_CONVERT_SIZE_FITTOSIZE );
canvasOption.mSaveOption.setConvertCanvasHorizontalAlignOption
                        ( SAMMLibConstants.SAMM_ALIGN_CENTER );
canvasOption.mSaveOption.setConvertCanvasVerticalAlignOption
                        ( SAMMLibConstants.SAMM_ALIGN_NORMAL );

// Option Settings.
mSCanvas.setOption(canvasOption);

// Load a SAMM file.
mSCanvas.loadSAMMFile( ... );
```

[Code 45] Applying canvas size and horizontal/vertical alignment options

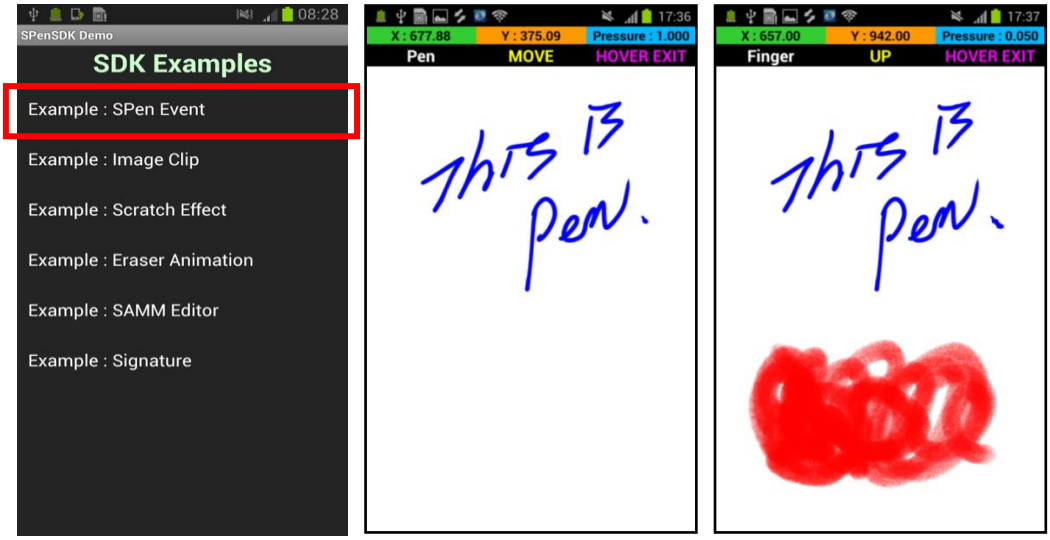
3-4. Support for Handling S Pen Events

In the S Pen SDK, there are listeners and methods for handling S Pen events and information values. Instead of the OnTouchListener, the S Pen SDK uses SPenTouchListener to handle S Pen events.

The S Pen actions that can be handled by the S Pen SDK consist of touch input from a pen or finger, and erasers when using the S Pen with an eraser attached.
Also, checking the status of the up and down buttons on the side of the S Pen is also supported.

In Android Ice Cream Sandwich or later versions, the S Pen SDK supports hovering events which can check the location of the S Pen within a certain distance without an actual touch on the screen from the S Pen.
Related events can be handled by using setSPenHoverListener.

Also, a method for recognizing when the S Pen is taken out of the device has been added in S Pen SDK 2.2.



[Figure 18] Identifying and handling S Pen input and finger input events

3-4-1. SPenTouchListener Listener

You can use the SPenTouchListener by first registering the listener with the canvas and overriding methods within the listener to handle touch events.

The methods provided within the listener generally use View objects that are created after a touch event and the MotionEvent object that contains the touch event information to be delivered as parameters. (For more details regarding the MotionEvent object, please refer to the Android developer website.)

This listener is included in the com.samsung.spensdk.applistener package.

The table below shows the methods that can be checked using the listener.

boolean onTouchFinger(View, MotionEvent)	Called when the drawing area is touched with a finger.
boolean onTouchPen(View, MotionEvent)	Called when the drawing area is touched with an S Pen.
boolean onTouchPenEraser(View, MotionEvent)	Called when the drawing area is touched with an S Pen while in eraser mode.
void onTouchButtonDown(View, MotionEvent)	Called when the drawing area is touched with an S Pen with the side button pressed (down).
void onTouchButtonUp(View, MotionEvent)	Called when the drawing area is pressed with the side button released (up).

The example code below shows how to register listeners to handle various events created by an S Pen in an SCanvasView, and how to use the methods responsible for handling each event.

```
SCanvasView mSCanvas = (SCanvasView)findViewById(R.id.canvas_view);

mSCanvas.setSPenTouchListener( new SPenTouchListener() {
    @Override
    public boolean onTouchFinger(View view, MotionEvent event) {
        mSCanvas.setPenSetting( SObjectStroke.SAMM_STROKE_STYLE_CRAYON, 50, Color.RED);
    }
    @Override
    public boolean onTouchPen(View view, MotionEvent event) {
        mSCanvas.setPenSetting( SObjectStroke.SAMM_STROKE_STYLE_PENCIL, 20, Color.Blue );
    }
    @Override
    public boolean onTouchPenEraser(View view, MotionEvent event) {
        mSCanvas.setPenSetting( SObjectStroke.SAMM_DEFAULT_MAX_ERASERSIZE);
    }
    @Override
    public void onTouchButtonDown(View view, MotionEvent event) {
        //
    }

    @Override
    public void onTouchButtonUp(View vuiew, MotionEvent event) {
        //
    }
});
```

[Code 46] Examples of SPenTouchListener listeners

The example code above simply shows how the Pen settings for a touch event are changed.

Please keep in mind that onTouchButtonDown and onTouchButtonUp methods only function properly in Android 4.x Ice Cream Sandwich or later.

3-4-2. SPenHoverListener Listener

In Android Ice Cream Sandwich or later versions, support for hovering events was added to support StyluS Pens.

In the S Pen SDK, you can easily handle hover events by using the SPenHoverListener listener. First, you must create and register an instance of the listener, override the listener's internal methods that correspond to each event, and register the listener with the SCanvasView.

This listener is included in the com.samsung.spensdk.applistener package.

The table below shows the methods that can be checked using the listener.

boolean onHover(View, MotionEvent)	Called when a hover event occurs as the S Pen reaches a certain distance from the screen without touching the screen surface.
void onHoverButtonDown(View, MotionEvent)	Called when a hover event occurs as the S Pen reaches a certain distance from the screen without touching the screen surface with the side button of the S Pen pressed (down).
void onHoverButtonUp(View, MotionEvent)	Called when a hover event occurs as the S Pen reaches a certain distance from the screen without touching the screen surface with the side button of the S Pen released (switched from down to up).

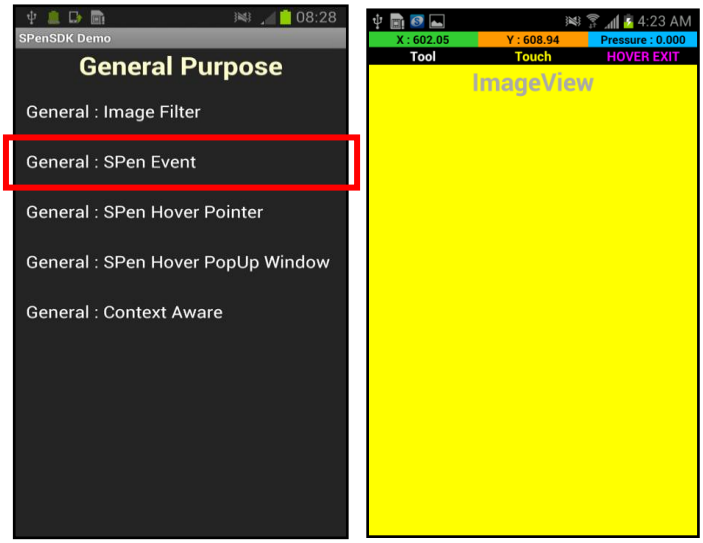
The example code below shows how to use the methods introduced above.

```
Context mContext = this;
SCanvasView mSCanvas = (SCanvasView)findViewById(R.id.canvas_view);

mSCanvas.setSPenHoverListener(new SPenHoverListener()
{
   @Override
   public boolean onHover(View view, MotionEvent event) {
   }

   @Override
   public void onHoverButtonDown(View view, MotionEvent event) {
   }

   @Override
   public void onHoverButtonUp(View view, MotionEvent event) {
        // Show S Pen-Settings pop-up window when the side button on the S Pen is pressed.
        mSCanvas.toggleShowSettingView( SCanvasConstants.SCANVAS_SETTINGVIEW_PEN );
   }
} );
```

[Code 47] Handling hover events using the SPenHoverListener listener

The above example shows that actions such as calling a pop-up window can be done by listening to the hover event and pressing a side button. Keep in mind that the SPenHoverListener and related methods used in the example above only function properly in Android Ice Cream Sandwich or later.

Also, S Pen SDK 2.2 also includes the SPenEventLibrary class and methods to allow use in other views besides an SCanvasView.
You can find out more from the "General: S Pen Event" example included in the SDK.



[Figure 19] Handling S Pen-related events in a normal View instead of an SCanvasView

```
Public class SPen_Example_SPenEventGeneral extends Activity
{
    // Library class for handling S Pen events.
    private SPenEventLibrary mSPenEventLibrary;

    // Another type of View that is not SCanvasView-based.
    private ImageView mImageView;

    ¡¦
    mImageView = (ImageView) findViewById(R.id.image_view);
    ¡¦

    mSPenEventLibrary = new SPenEventLibrary();
```

```
    // Register listeners in another type of View that is not SCanvasView-based.
    mSPenEventLibrary.setSPenTouchListener( mImageView, new SPenTouchListener()
    {
        @Override
        public boolean onTouchFinger(View view, MotionEvent event) {
        }
        @Override
        public boolean onTouchPen(View view, MotionEvent event) {
        }
        @Override
        public boolean onTouchPenEraser(View view, MotionEvent event) {
        }
        @Override
        public void onTouchButtonDown(View view, MotionEvent event) {
        }
        @Override
        public void onTouchButtonUp(View view, MotionEvent event) {
        }
    } );

}
```

[Code 48] S Pen event handling method in normal view

```
mSPenEventLibrary.setSPenHoverListener(mImageView, new SPenHoverListener()
{
        @Override
        public boolean onHover(View view, MotionEvent event) {
        }

        @Override
        public void onHoverButtonDown(View view, MotionEvent event) {
        }

        @Override
        public void onHoverButtonUp(View view, MotionEvent event) {
        }
}   );
```

[Code 49] Hover event handling method in normal view

3-4-3. SPenTouchListener Listener

The SPenDetachmentListener listener and related methods, which are newly added to S Pen SDK 2.2, register and unregister listeners that are called when an S Pen is inserted into or taken out of the device.

```
mSPenEventLibrary.registerSPenDetachmentListener( mContext, new SPenDetachmentListener()
{
        @Override
        public void onSPenDetached(boolean bDetached)
        {
                if( bDetached )
                    Toast.makeText( mContext, " SPen Detached",
                                    Toast.LENGTH_SHORT ).show();
                else
                    Toast.makeText(mContext, "S Pen Inserted", Toast.LENGTH_SHORT).show();
        }
} );
```

[Code 50] Handling events when pen is inserted or taken out of the device

After using this listener, the unregisterSPenDetachmentListener method must be used to free memory when the program (Activity) is closed.

```
@Override
protected void onDestroy()
{
        // Unregister an SPenDetachment listener.
        mSPenEventLibrary.unregisterSPenDetachmentListener(mContext);

        super.onDestroy();
}
```

[Code 51] Unregistering an SPenDetachment listener when the program closes

Please keep in mind that the related listeners and methods only function properly on certain devices, including the Galaxy Note 10.1.

3-5. Image Filter

The S Pen SDK provides various image filters for easily handling image processing tasks. You can apply various filters provided in the SDK, such as sepia, pastel, and cartoon effects to the target image.

You can use either one of the two methods below to implement this feature.

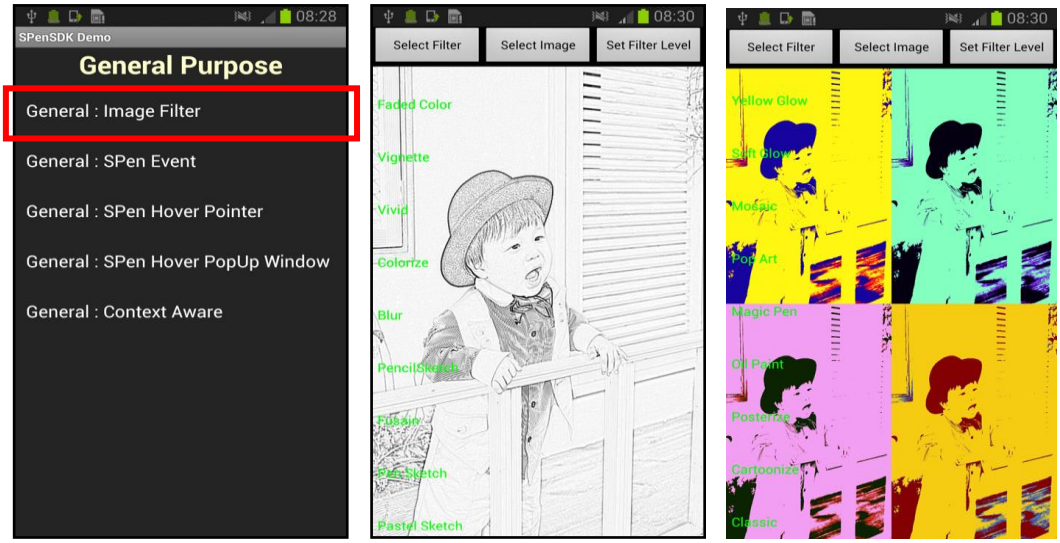
```
boolean result = SPenImageFilter.filterImage( bitmap, filter, level );
```

[Code 52] Apply filters to an image object

```
Bitmap filteredBitmap = SPenImageFilter.filterImageCopy( sourceBitmap, filter, level );
```

[Code 53] Apply filters to a copy of the image object

Note that the original image may be altered when applying the filter values to the image without creating a copy of the image object.



[Figure 20] Various image filter demonstrations

The example code below shows the image filters provided in the S Pen SDK and how to apply them to copies of image objects with a random number for the filter level.

```
final int imageOperationByIndex[] = {
                    SPenImageFilterConstants.FILTER_ORIGINAL,
                    SPenImageFilterConstants.FILTER_GRAY,
                    SPenImageFilterConstants.FILTER_SEPIA,
                    SPenImageFilterConstants.FILTER_NAGATIVE,
                    SPenImageFilterConstants.FILTER_BRIGHT,
                    SPenImageFilterConstants.FILTER_DARK,
                    SPenImageFilterConstants.FILTER_VINTAGE,
                    SPenImageFilterConstants.FILTER_OLDPHOTO,
                    SPenImageFilterConstants.FILTER_FADEDCOLOR,
                    SPenImageFilterConstants.FILTER_VIGNETTE,
                    SPenImageFilterConstants.FILTER_VIVID,
                    SPenImageFilterConstants.FILTER_COLORIZE,
                    SPenImageFilterConstants.FILTER_BLUR,
                    SPenImageFilterConstants.FILTER_PENCILSKETCH,
                    SPenImageFilterConstants.FILTER_FUSAIN,
                    SPenImageFilterConstants.FILTER_PENSKETCH,
                    SPenImageFilterConstants.FILTER_PASTELSKETCH,
                    SPenImageFilterConstants.FILTER_COLORSKETCH,
                    SPenImageFilterConstants.FILTER_PENCILPASTELSKETCH,
                    SPenImageFilterConstants.FILTER_PENCILCOLORSKETCH,
                    SPenImageFilterConstants.FILTER_RETRO,
                    SPenImageFilterConstants.FILTER_SUNSHINE,
                    SPenImageFilterConstants.FILTER_DOWNLIGHT,
                    SPenImageFilterConstants.FILTER_BLUEWASH,
                    SPenImageFilterConstants.FILTER_NOSTALGIA,
                    SPenImageFilterConstants.FILTER_YELLOWGLOW,
                    SPenImageFilterConstants.FILTER_SOFTGLOW,
                    SPenImageFilterConstants.FILTER_MOSAIC,
                    SPenImageFilterConstants.FILTER_POPART,
                    SPenImageFilterConstants.FILTER_MAGICPEN,
                    SPenImageFilterConstants.FILTER_OILPAINT,
                    SPenImageFilterConstants.FILTER_POSTERIZE,
                    SPenImageFilterConstants.FILTER_CARTOONIZE,
                    SPenImageFilterConstants.FILTER_CLASSIC                    };

final int levels[] = { SPenImageFilterConstants.FILTER_LEVEL_VERYSMALL,
                     SPenImageFilterConstants.FILTER_LEVEL_SMALL,
                     SPenImageFilterConstants.FILTER_LEVEL_MEDIUM,
                     SPenImageFilterConstants.FILTER_LEVEL_LARGE,
                     SPenImageFilterConstants.FILTER_LEVEL_VERYLARGE     };

mSCanvas.setBitmap(    SPenImageFilter.filterImageCopy
                ( mSCanvas.getBitmap(true),
                imageOperationByIndex[ new Random().nextInt( imageOperationByIndex.length ) ],
                levels[new Random().nextInt(levels.length)]),
                false    );
```

[Code 54] Example of image processing using image filters

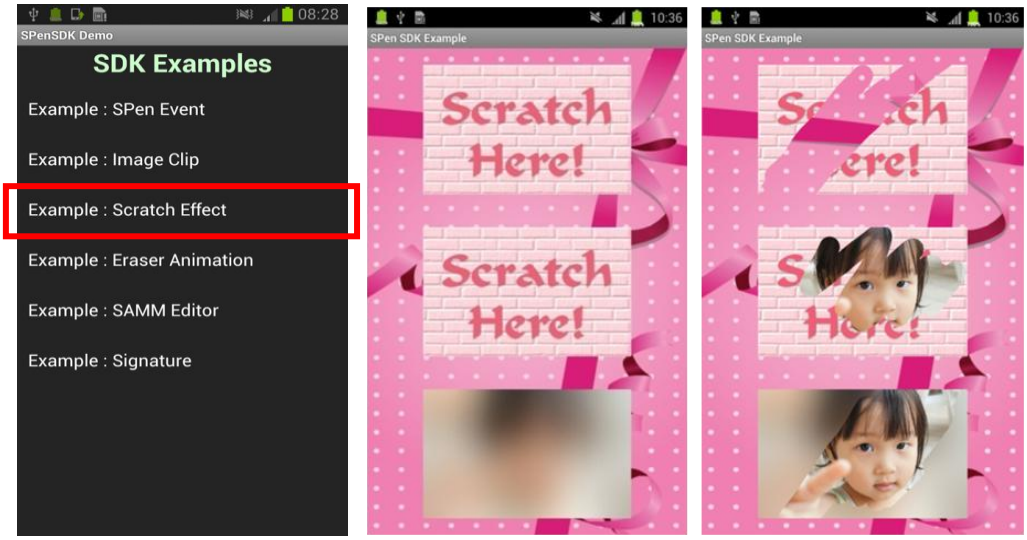
Also, the applicable range has been expanded in S Pen SDK 2.2 to allow independent use of filters in views other than an SCanvasView.
You can find out more about the related features in the "General: ImageFilter" example included in the SDK.

3-6. Scratch Effect

You can also use the S Pen SDK to easily create scratch effects, similar to a scratch-off lottery ticket.
To create this effect, the final image must be hidden by a cover image.
You can set the cover image by using the setClearImageBitmap method.
You can apply various filters on the cover image to create various effects.
You can find out more about the scratch effect in the "Example: Scratch Effect" example included in the SDK.

```
SCanvasInitializeListener scanvasInitializeListener1 = new SCanvasInitializeListener()
{
    @Override
    public void onInitialized()
    {
        mSCanvas1.setCanvasZoomEnable(false);
        //
        mSCanvas1.setCanvasMode(SCanvasConstants.SCANVAS_MODE_INPUT_ERASER);
        //
        mSCanvas1.setEraserStrokeSetting(SObjectStroke.SAMM_DEFAULT_MAX_ERASERSIZE);

        // Set the final image.
        Bitmap bmScratch = BitmapFactory.decodeResource( getResources(),
                                                         R.drawable.scratch_pattern );
        //
        mSCanvas1.setClearImageBitmap( bmScratch,
                                       SPenImageFilterConstants.FILTER_ORIGINAL,
                                       SPenImageFilterConstants.FILTER_LEVEL_MEDIUM );
    }
};
```

[Code 55] Applying a scratch effect



[Figure 21] Scratch effect demonstration

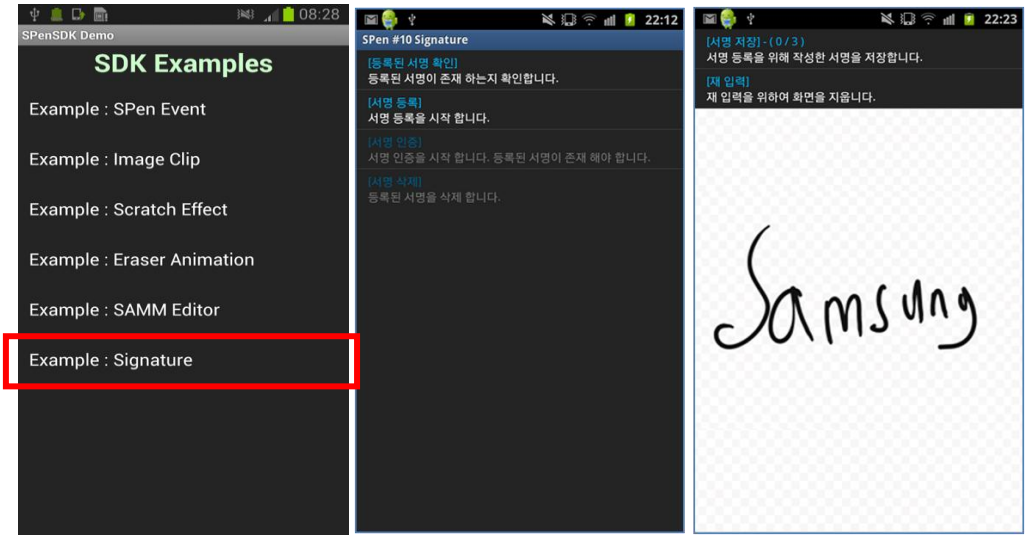
3-7. Signature Recognition

In the S Pen SDK, a signature recognition engine is embedded within SCanvasView to provide signature recognition functionality. Related methods are provided in the SDK.

Signature recognition compares the signature (or drawing) the user inputs on the canvas to the registered signature (or drawing) to determine if the signature by the user matches the registered signature.

Related methods are described in the table below.

boolean openSignatureEngine();	Initializes and opens the signature recognition engine.
boolean closeSignatureEngine();	Closes the signature recognition engine.
int registerSignature();	Adds a new signature to the internal list and returns the registered order number.
boolean isSignatureRegistrationCompleted()	Checks if 3 signature samples have been registered for the signature recognition engine and returns true or false.
static boolean isSignatureExist (Context context)	Returns whether a registered signature exists or not. (Returns true or false after checking the availability of the signature recognition in the current activity.)
boolean verifySignature(int verificationLevel);	Compares the input in the canvas with the registered signature and returns true or false. Takes one of three recognition accuracy levels (low/med/high) as an input parameter. SCanvasConstants.SIGNATURE_VERIFICATION_LEVEL_LOW SCanvasConstants.SIGNATURE_VERIFICATION_LEVEL_MEDIUM SCanvasConstants.SIGNATURE_VERIFICATION_LEVEL_HIGH
static boolean unregisterSignature (Context context)	Unregisters the currently registered signature.



[Figure 22] Example showing the signature recognition feature

The following example code shows how to register a new signature, compares the signature to the registered signature information, and receives the recognition results. It also shows how to delete a registered signature.

```
@Override
public void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);

        ...

        SCanvasInitializeListener   scanvasInitializeListener
                                    = new SCanvasInitializeListener()
        {
                @Override
                public void onInitialized()
                {
                        // Start Signature Mode.
                        mSCanvas.openSignatureEngine();
                        // Disable zooming in and out.
                        mSCanvas.setCanvasZoomEnable( false );
                 }
        };
}
```

```
@Override
public void onBackPressed()
{
	mSCanvas.closeSignatureEngine();
	super.onBackPressed();
}
```

[Code 56] Initializes the signature recognition engine

```
private int mResult = 0;
...
mResult = mSCanvas.registerSignature();
...
if ( mSCanvas.isSignatureRegistrationCompleted() && mResult > 0 )
{
	//registration finished, do what you will
}
else
{
	if (nResult == 1) {
		//first signature sample registered
	} else if (nResult == 2) {
		//second signature sample registered
	} else {
		// Signature registration error
	}
}
```

[Code 57] Register the signature on the current screen

```
if ( SCanvasView.isSignatureExist( SPen_Example_Signature.this ) ) {
	 // When the signature exists.
}
else {
	 // When the signature does not exist.
}
```

[Code 58] Check whether the signature is registered

```
int mVerificationLevel = SCanvasConstants.SIGNATURE_VERIFICATION_LEVEL_MEDIUM;

if ( mSCanvas.verifySignature( mVerificationLevel ) )
{
        // When the signature matches.
}
else
{
        // When the signature does not match.
}
```

[Code 59] Check if the current signature on the canvas matches the registered signature

Deleting a registered signature is done by using the unregisterSignature method of SCanvasview.

```
if ( SCanvasView.unregisterSignature( SPen_Example_Signature.this ) )
{
    Toast.makeText( getApplicationContext(),
                    "Signatures cleared", Toast.LENGTH_SHORT).show();
}
else
{
    Toast.makeText( getApplicationContext(),
                    "No signatures to clear", Toast.LENGTH_SHORT).show();
}
```

[Code 60] Delete the registered signature