Pen SDK Light

Programming Guide

Version 4.1.2

Table of Contents

1. Overview

Pen SDK Light allows you to develop applications that use handwritten inputs. It uses a S pen, finger, or other kinds of virtual pens to provide faster and more precise user input. This means that Pen SDK Light offers a richer set of features than existing input tools. Because it senses the pressure underneath its tip, Pen SDK Light makes it feel more like you are actually writing or drawing on the device.

Pen SDK Light provides functions for verifying if the Spen is activated, identifying event coordinates, sensing the pressure, verifying if the side button is pressed, processing hover events and more for your application.

You can use Pen SDK Light to:

- draw using a finger and/or S pen
- set user preferences for pens and erasers.
- edit and save input stroke objects as a file
- manage history for undo and redo commands

1.1. Basic Knowledge

The Pen SDK Light motion events include touch events and hover events. Touch events occur when a S pen touches the screen and hover events occur when a S pen is within a certain range of the screen.

The SpenSimpleSurfaceView class, which inherits from Android SurfaceView, processes finger and S pen inputs to express data on the viewport. Pen SDK Light saves the objects drawn on an SpenSimpleSurfaceView instance in SpenPageDoc, with multiple SpenPageDocs making an SpenNoteDoc file.

The following figure shows the relationship between SpenPageDoc and SpenSimpleSurfaceView.

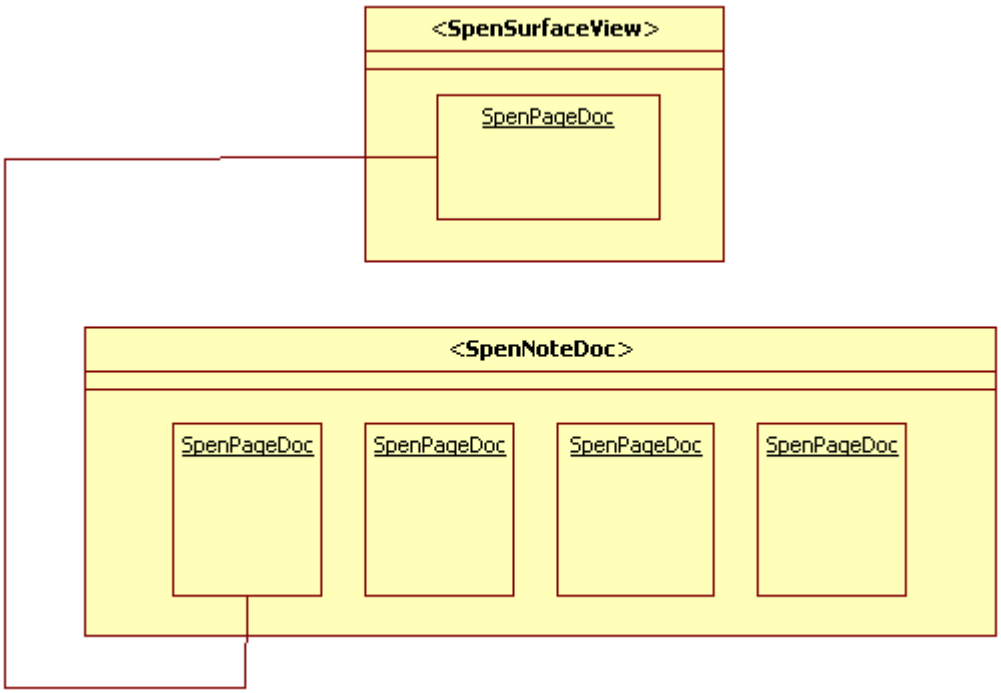


Figure 1: Relationship between SpenPageDoc and SpenSimpleSurfaceView

1.2. Architecture

The following figure shows the Pen SDK Light architecture.

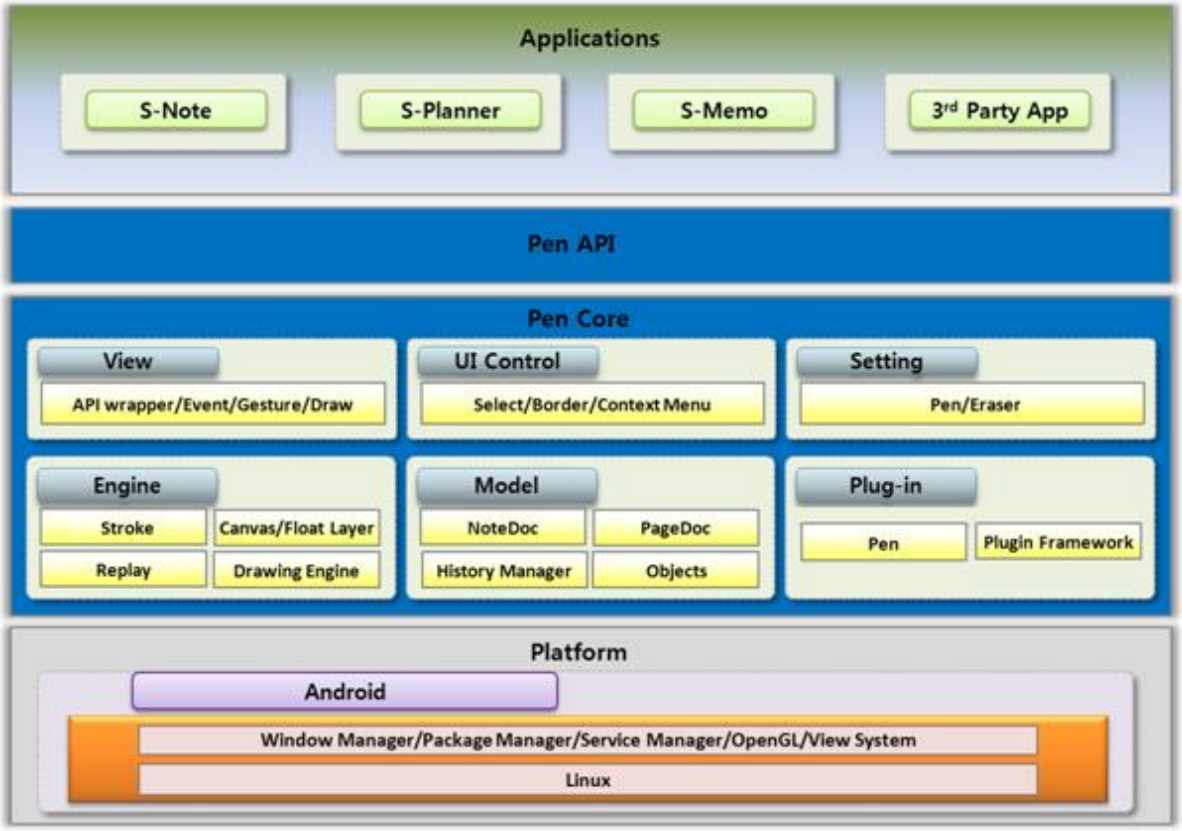


Figure 2: Pen SDK Light architecture

The architecture consists of:

- **Applications:** One or more applications that use Pen SDK Light .
- **View:** Pen SDK Light 's components for managing user input on the viewport.
- **UI Control:** Pen SDK Light 's controls for objects on the viewport (scale, rotate, move, and select.)
- **Setting:** Pen SDK Light 's components for managing user preferences for pens and erasers.
- **Model:** Pen SDK Light 's components for adding, deleting, and saving data and for history management.
- **Plug-in:** Plug-ins for extending Pen SDK Light.

1.3. Package Diagram

The following figure shows the Pen SDK Light packages and classes that you can use in your application.

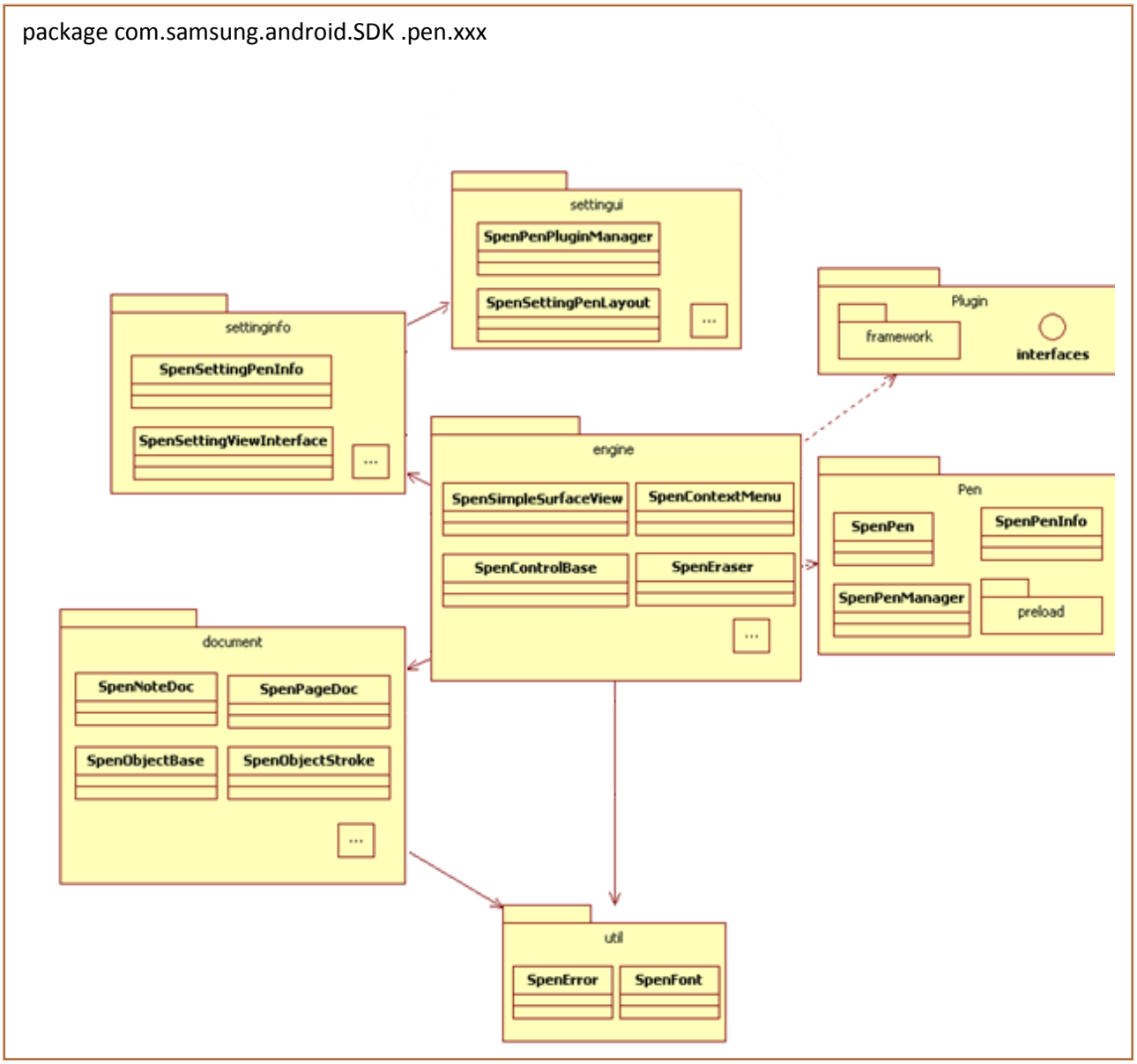


Figure 3: Pen SDK Light packages and classes

The Pen SDK Light packages and classes include:

- **SpenNoteDoc:** Manages SpenPageDocs.Corresponds to an SPD file.
- **SpenPageDoc:** Manages the Metadata, objects and layers of a page, which corresponds to a page in an SPD file.
- **SpenObjectStroke:** Manages user strokes. Each instance corresponds to a user stroke.
- **SpenSimpleSurfaceView:** Expresses data on the viewport and manages touch events and layers.
- **SpenPen:** Manages the pen strokes based on the user preferences.
- **SpenControlBase:** Provides the UI controls for scaling, rotating, moving, and selecting objects.
- **settingui:** Provides the UI controls for the pen settings View.
- **settinginfo:** Contains data on the pen settings.

Replace xxx in the image above with document, engine, pen, Plugin, recognition, settingui, settinginfo or util to get the full package name in question.

1.4. Supported Platforms

Android 4.0 (Ice Cream Sandwich API Level 14) or above support Pen SDK Light.

1.5. Supported Features

Pen SDK Light supports the following features:

- Processing S pen touch events and hover events, sensing S pen pressure and checking if the side button is pressed.
- Zoom in and out and pan on the viewport.
- Managing user preferences for pens and erasers.
- Managing input objects and maintaining their states.
- Selecting, scaling, moving objects.
- Managing the history of an input object.
- Adding third party templates.

1.6. Components

- Components
    - pen-v4.1.x.aar
    - sdk-v1.0.0.jar
- Imported Pen SDK Light:
    - com.samsung.android.SDK .pen

1.7. Importing Libraries

To import Pen SDK Light libraries to the application project:

Add the pen-v4.1.x.aar and sdk-v1.0.0.jar files to the libs folder.

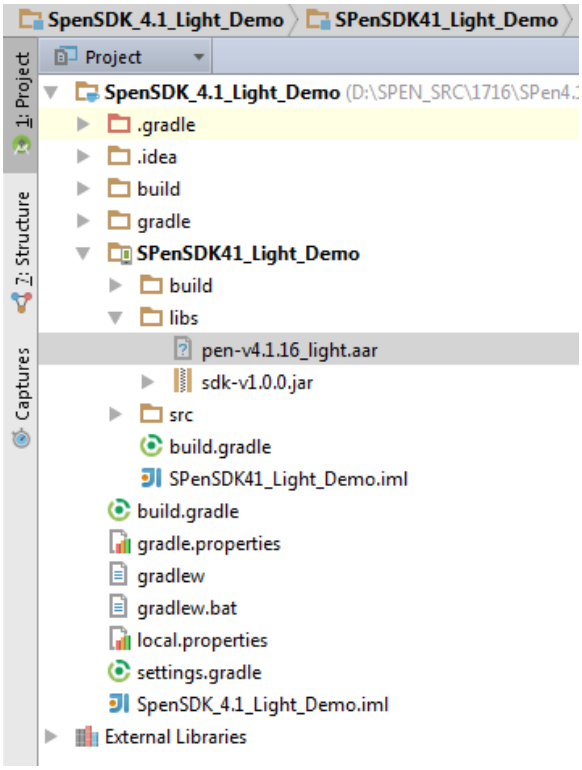


Figure 4: libs folder in Android Studio

Add lib to your build.gradle to link to Android Dependences

```
repositories {
    flatDir {
        dirs 'libs'
    }
}

dependencies {
    compile files('libs/sdk-v1.0.0.jar')
    compile(name:'pen-v4.1.2_full', ext:'aar');
}
```

Add the following permission to your Android manifest file to access the Pen SDK Light external storage.

```
<uses-permissionandroid:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Select Android 4.0 (Ice Cream Sandwich) or higher as a Project Build Target in your project properties.

The following permission has to be specified in the AndroidManifest.xml fileto initialize Pen SDK Light.

```
<uses-permissionandroid:name=
"com.samsung.android.providers.context.permission.WRITE_USE_APP_FEATURE_SURVEY"/>
```

If you don’t add the permission,

- Android 4.4.2 (KitKat) and above: SecurityException is thrown and your application doesn’t work.
- Prior to Android 4.4.2 (KitKat): No exception. And the application works properly.

2. HelloPen

HelloPen is a simple program that:

- gets input from a S pen
- expresses drawing on the viewport

```
public class PenSample1_1_HelloPen extends Activity {

    private Context mContext;
    private SpenNoteDoc mSpenNoteDoc;
    private SpenPageDoc mSpenPageDoc;
    private SpenSimpleSurfaceView mSpenSimpleSurfaceView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hello_pen);
        mContext = this;

        // Initialize Spen
        boolean isSpenFeatureEnabled = false;
        Spen spenPackage = new Spen();
        try {
            spenPackage.initialize(this);
            isSpenFeatureEnabled = spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
        } catch (SSDK UnsupportedException e) {
            if( processUnsupportedException(e) == true) {
                return;
            }
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Spen.",
                Toast.LENGTH_SHORT).show();
            e1.printStackTrace();
            finish();
        }

        // Create Spen View
        RelativeLayout spenViewLayout =
            (RelativeLayout) findViewById(R.id.spenViewLayout);
        mSpenSimpleSurfaceView = new SpenSimpleSurfaceView(mContext);
        if (mSpenSimpleSurfaceView == null) {
            Toast.makeText(mContext, "Cannot create new SpenView.",
                Toast.LENGTH_SHORT).show();
            finish();
        }
        spenViewLayout.addView(mSpenSimpleSurfaceView);

        // Get the dimension of the device screen.
        Display display = getWindowManager().getDefaultDisplay();
        Rect rect = new Rect();
        display.getRectSize(rect);
        // Create SpenNoteDoc
        try {
            mSpenNoteDoc =
                new SpenNoteDoc(mContext, rect.width(), rect.height());
```

```
        } catch (IOException e) {
            Toast.makeText(mContext, "Cannot create new NoteDoc.",
                Toast.LENGTH_SHORT).show();
            e.printStackTrace();
            finish();
        } catch (Exception e) {
            e.printStackTrace();
            finish();
        }
        // Add a Page to NoteDoc, get an instance, and set it to the member variable.
        mSpenPageDoc = mSpenNoteDoc.appendPage();
        mSpenPageDoc.setBackgroundColor(0xFFD6E6F5);
        mSpenPageDoc.clearHistory();
        // Set PageDoc to View.
        mSpenSimpleSurfaceView.setPageDoc(mSpenPageDoc, true);

        if(isSpenFeatureEnabled == false) {
            mSpenSimpleSurfaceView.setToolTypeAction(SpenSimpleSurfaceView.TOOL_FINGER,
SpenSimpleSurfaceView.ACTION_STROKE);
            Toast.makeText(mContext,
                "Device does not support Spen. \n You can draw stroke by finger.",
                Toast.LENGTH_SHORT).show();
        }
    }

    protected void onDestroy() {
        super.onDestroy();

        if (mSpenSimpleSurfaceView != null) {
            mSpenSimpleSurfaceView.close();
            mSpenSimpleSurfaceView = null;
        }

        if(mSpenNoteDoc != null) {
            try {
                mSpenNoteDoc.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            mSpenNoteDoc = null;
        }
    };
}
```

3. Using the Spen Class

The Spen class provides the following methods:

- `initialize()` initializes Pen SDK Light. You need to initialize Pen SDK Light before you can use it. If the device does not support S Pen, SSDK UnsupportedException is thrown.
- `getVersionCode()` returns the Pen SDK Light version number as an integer.
- `getVersionName()` returns the Pen SDK Light version name as a string.
- `isFeatureEnabled()`checks if a Pen SDK Light feature is available on the device.

```
boolean isSpenFeatureEnabled = false;
Spen spenPackage = new Spen();
try {
       spenPackage.initialize(this);
       isSpenFeatureEnabled = spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
} catch (SSDK UnsupportedException e) {
int eType = e.getType();
if (eType == SSDK UnsupportedException.VENDOR_NOT_SUPPORTED) {
// The device is not a Samsung device.
       } else if (eType == SSDK UnsupportedException.DEVICE_NOT_SUPPORTED) {
// The device does not support Pen.
       } else if (eType == SSDK UnsupportedException.LIBRARY_NOT_INSTALLED) {
// SPen SDK Light 4.1 apk is not installed on the device.
       } else if (eType
                     == SSDK UnsupportedException.LIBRARY_UPDATE_IS_REQUIRED) {
// The Pen library or SPen SDK Light 4.1 apk requires to be updated.
       } else if (eType
                     == SSDK UnsupportedException.LIBRARY_UPDATE_IS_RECOMMENDED) {
// It is recommended that the Pen library or SPen SDK Light 4.1 apk isupdated to the
// latest version as possible.
       }
} catch (Exception e) {
          Toast.makeText(this, "Cannot initialize Pen.",
                   Toast.LENGTH_SHORT).show();
          finish();
}

int versionCode = spenPackage.getVersionCode();
String versionName = spenPackage.getVersionName();
```

For more information, see onCreate() in PenSample1_1_HelloPen.java.

3.1. Using the initialize() method

The `Spen.initialize()` method:

- initializes Pen SDK Light
- checks if the device is a Samsung device
- checks if the Samsung device supports Pen SDK Light

- checks if Pen SDK Light 4.1 apk are installed on the device

```
void initialize(Context context) throws SSDK UnsupportedException
```

If initializing Pen SDK Light is failed, the initialize() method throws an SSDK UnsupportedException exception. To find out the reason for the exception, check the exception message.

3.2. Handling SSDK UnsupportedException

If an SSDK UnsupportedException exception is thrown, check the exception message type using SSDK UnsupportedException.getType().

The following five types of exception messages are defined in the Spen class:

- VENDOR_NOT_SUPPORTED: The device is not a Samsung device.
- DEVICE_NOT_SUPPORTED: The device does not support Pen SDK Light.
- LIBRARY_NOT_INSTALLED: SPen SDK Light 4.1 apk is not installed on the device.
- LIBRARY_UPDATE_IS_REQUIRED: A necessary update for the Pen SDK Light library or SPen SDK Light 4.1 apk. If the library or apk is not updated, the user cannot use the application.
- LIBRARY_UPDATE_IS_RECOMMENDED: A recommendationto updatethe Pen SDK Light library or SPen SDK Light 4.1 apk, but it is not mandatory. The user can use the application without updating them.

3.3. Checking the Availability of Pen SDK Light Features

You can check if a S Pen feature is supported on the device withthe isFeatureEnabled() method. The feature typesare defined in the Spen class. Pass the feature type as a parameter when calling the isFeatureEnabled() method. The method returns a boolean value that indicates the support for the feature on the device.

```
boolean isFeatureEnabled(int type)
```

To check if the device supports the use of S pen:

1. Call Spen.isFeatureEnabled(Spen.DEVICE_PEN).
2. If the method returns false (which means S pen are not supported), call SpenSimpleSurfaceView.setToolTypeAction()and set TOOL_FINGER to ACTION_STROKE to enable input with user fingers.

```
if(spenPackage.isFeatureEnabled(Spen.DEVICE_PEN) == false) {
mSpenSimpleSurfaceView.setToolTypeAction(SpenSimpleSurfaceView.TOOL_FINGER,
                SpenSimpleSurfaceView.ACTION_STROKE);
    Toast.makeText(mContext,
```

```
"Device does not support S pen. \n You can draw strokeswith yourfinger",
    Toast.LENGTH_SHORT).show();
}
```

3.4. Supporting Devices

Pen SDK Light supports both of 32 and 64 bit execution environments. Even SDK supports both of them, because it can't know what environment will be used , you need to inform the environment to SDK .

The execution environment is determined by your application's configuration.

There are 3 possible options :

- Android application without any native library
- Android application with only 32bit native library
- Android application withboth of 32 and 64bit native library

3.4.1. Android application without any native library

If it does not have any native library, the execution environment is determined by underlying system. If the system is 32 bit, application will be run on 32 bit execution environment. If it 64 bit, the execution environment will be 64 bit.

In this case, just use below default initialize method.

```
void initialize(Context context) throws SSDK UnsupportedException
```

3.4.2. Android application contains only 32bit native library

If an application contains native library and the library is built with 32 bit mode, the application will be run on 32 bit execution environment even undelying system is 64 bit.

In this case, the application has to call initialize method with "isForce32BitMode = true".

```
void initialize(Context context, int maxCacheSize, int createMode, boolean isForce32BitMode)throws SSDK
UnsupportedException
```

Note

Pen SDK Light libraries including 32/64bit .so files. In this case 64 bit .so files is unnecessary. For saving memory of application, modify build.gradle file in Android Studio as below:
`packagingOptions {`
`  exclude 'lib/arm64-v8a/libgnustl_shared.so'`
`  exclude 'lib/arm64-v8a/libSPenBase.so'`
`  exclude 'lib/arm64-v8a/libSPenBeautify.so'`
`  exclude 'lib/arm64-v8a/libSPenBrush.so'`
`  exclude 'lib/arm64-v8a/libSPenChineseBrush.so'`
`  exclude 'lib/arm64-v8a/libSPenEngine.so'`

Note

```
    exclude 'lib/arm64-v8a/libSPenFountainPen.so'
    exclude 'lib/arm64-v8a/libSPenInit.so'
    exclude 'lib/arm64-v8a/libSPenInkPen.so'
    exclude 'lib/arm64-v8a/libSPenMagicPen.so'
    exclude 'lib/arm64-v8a/libSPenMarker.so'
    exclude 'lib/arm64-v8a/libSPenModel.so'
    exclude 'lib/arm64-v8a/libSPenMontblancCalligraphyPen.so'
    exclude 'lib/arm64-v8a/libSPenMontblancFountainPen.so'
    exclude 'lib/arm64-v8a/libSPenObliquePen.so'
    exclude 'lib/arm64-v8a/libSPenPencil.so'
    exclude 'lib/arm64-v8a/libSPenPluginFW.so'
}
```

3.4.3. Android application contains 32bit/64 bit native library

If an application contains native library and the library contains both of 32 and 64 bit mode, you also don't need care what environment will be used like the applicationwithout any native library case.

Just use belowdefault initialize method.

```
void initialize(Context context) throws SSDK UnsupportedException
```

3.4.4. Prompt to install or update Pen SDK Light

If initializing Pen SDK Light is failed, the `initialize()` method throws an SSDK UnsupportedException exception. Application should display a message that prompts the user to install or update `SPen SDK Light 4.1` apk and open the website to download the package by following Uri:

```
Uri uri = Uri.parse("market://details?id="+ Spen.getSpenPackageName());
```

4. Using Pen SDK Light

4.1. Using Pen SDK Light Views

SpenSimpleSurfaceView, which inherits from Android SurfaceView, processes finger gestures or S pen input to express drawings on the viewport.

SpenSimpleSurfaceView is the view component in the model-view-controller paradigm, and it generates a representation of the object data on the viewport. SpenSimpleSurfaceView provides controls for scaling, rotating, moving, and selecting objects.

SpenSimpleSurfaceView and SpenSettingPenLayout combine to provide methods for managing user preferences for font, font size, and font color; the size, color, or type of the pen tool; the size of the eraser tool; and options for objects.

4.1.1. Drawing on the Screen

The following simple application creates an SpenSimpleSurfaceView instance on the viewport, which allows you to draw with a S pen.

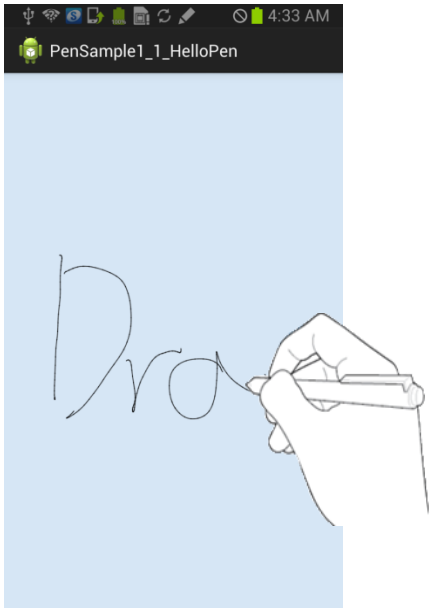


Figure 5: Basic drawing

```
public class PenSample1_1_HelloPen extends Activity {

    private Context mContext;
    private SpenNoteDoc mSpenNoteDoc;
    private SpenPageDoc mSpenPageDoc;
```

```
    private SpenSimpleSurfaceView mSpenSimpleSurfaceView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hello_pen);
        mContext = this;

        // Initialize Spen
        boolean isSpenFeatureEnabled = false;
        Spen spenPackage = new Spen();
        try {
            spenPackage.initialize(this);
            isSpenFeatureEnabled = spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
        } catch (SSDK UnsupportedException e) {
            if( processUnsupportedException(e) == true) {
                return;
            }
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Spen.",
                Toast.LENGTH_SHORT).show();
            e1.printStackTrace();
            finish();
        }

        // Create Spen View
        RelativeLayout spenViewLayout =
            (RelativeLayout) findViewById(R.id.spenViewLayout);
        mSpenSimpleSurfaceView = new SpenSimpleSurfaceView(mContext);
        if (mSpenSimpleSurfaceView == null) {
            Toast.makeText(mContext, "Cannot create new SpenView.",
                Toast.LENGTH_SHORT).show();
            finish();
        }
        spenViewLayout.addView(mSpenSimpleSurfaceView);

        // Get the dimension of the device screen.
        Display display = getWindowManager().getDefaultDisplay();
        Rect rect = new Rect();
        display.getRectSize(rect);
        // Create SpenNoteDoc
        try {
            mSpenNoteDoc =
                new SpenNoteDoc(mContext, rect.width(), rect.height());
        } catch (IOException e) {
            Toast.makeText(mContext, "Cannot create new NoteDoc.",
                Toast.LENGTH_SHORT).show();
            e.printStackTrace();
            finish();
        } catch (Exception e) {
            e.printStackTrace();
            finish();
        }
        // Add a Page to NoteDoc, get an instance, and set it to the member variable.
        mSpenPageDoc = mSpenNoteDoc.appendPage();
        mSpenPageDoc.setBackgroundColor(0xFFD6E6F5);
        mSpenPageDoc.clearHistory();
        // Set PageDoc to View.
        mSpenSimpleSurfaceView.setPageDoc(mSpenPageDoc, true);
```

```
        if(isSpenFeatureEnabled == false) {
            mSpenSimpleSurfaceView.setToolTypeAction(SpenSimpleSurfaceView.TOOL_FINGER,
SpenSimpleSurfaceView.ACTION_STROKE);
            Toast.makeText(mContext,
                "Device does not support Spen. \n You can draw stroke by finger.",
                Toast.LENGTH_SHORT).show();
        }
       }

    private boolean processUnsupportedException(SSDK UnsupportedException e) {

        e.printStackTrace();
        int errType = e.getType();
        // If the device is not a Samsung device or if the device does not support Pen.
        if (errType == SSDK UnsupportedException.VENDOR_NOT_SUPPORTED
                || errType == SSDK UnsupportedException.DEVICE_NOT_SUPPORTED) {
            Toast.makeText(mContext, "This device does not support Spen.",
                    Toast.LENGTH_SHORT).show();
            finish();
        }
        else if (errType == SSDK UnsupportedException.LIBRARY_NOT_INSTALLED) {
            // If SPen SDK Light  APK is not installed.
            showAlertDialog( "You need to install additional Spen software"
                +" to use this application."
                + "You will be taken to the installation screen."
                + "Restart this application after the software has been installed."
                , true);
        } else if (errType
                == SSDK UnsupportedException.LIBRARY_UPDATE_IS_REQUIRED) {
            // SPen SDK Light  APK must be updated.
            showAlertDialog( "You need to update your Spen software "
                + "to use this application."
                + " You will be taken to the installation screen."
                + " Restart this application after the software has been updated."
                , true);
        } else if (errType
                == SSDK UnsupportedException.LIBRARY_UPDATE_IS_RECOMMENDED) {
            // Update of SPen SDK Light  APK to an available new version is
recommended.
            showAlertDialog( "We recommend that you update your Spen software"
                +" before using this application."
                + " You will be taken to the installation screen."
                + " Restart this application after the software has been updated."
                , false);
            return false;
        }
        return true;
       }

    private void showAlertDialog(String msg, final boolean closeActivity) {

        AlertDialog.Builder dlg = new AlertDialog.Builder(mContext);
        dlg.setIcon(getResources().getDrawable(
            android.R.drawable.ic_dialog_alert));
        dlg.setTitle("Upgrade Notification")
            .setMessage(msg)
            .setPositiveButton(android.R.string.yes,
                new DialogInterface.OnClickListener() {
```

```
                    @Override
                    public void onClick(
                        DialogInterface dialog, int which) {
                        // Go to the market website and install/update APK.
                        Uri uri = Uri.parse("market://details?id=" +
Spen.getSpenPackageName());
                        Intent intent = new Intent(Intent.ACTION_VIEW, uri);
                        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
                            | Intent.FLAG_ACTIVITY_CLEAR_TASK);
                        startActivity(intent);

                        dialog.dismiss();
                        finish();
                    }
                })
            .setNegativeButton(android.R.string.no,
                new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(
                        DialogInterface dialog, int which) {
                        if(closeActivity == true) {
                            // Terminate the activity if APK is not installed.
                            finish();
                        }
                        dialog.dismiss();
                    }
            })
            .setOnCancelListener(new DialogInterface.OnCancelListener() {
                @Override
                public void onCancel(DialogInterface dialog) {
                    if(closeActivity == true) {
                        // Terminate the activity if APK is not installed.
                        finish();
                    }
                }
            })
            .show();
        dlg = null;
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();

        if (mSpenSimpleSurfaceView != null) {
            mSpenSimpleSurfaceView.close();
            mSpenSimpleSurfaceView = null;
        }

        if(mSpenNoteDoc != null) {
            try {
                mSpenNoteDoc.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            mSpenNoteDoc = null;
        }
    };
}
```

For more information, see PenSample1_1_HelloPen.java.

The following sections provide more details on the steps involved in drawing on the screen.

4.1.1.1 Initializing Pen SDK Light

To use Pen SDK Light, you must initialize it as shown below before using it.

```
Spen spenPackage = new Spen();
try {
    spenPackage.initialize(this);
    isSpenFeatureEnabled = spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
} catch (SSDK UnsupportedException e) {
    if (SDK Utils.processUnsupportedException(this, e) == true) {
        return;
    }
} catch (Exception e1) {
    Toast.makeText(mContext, "Cannot initialize Spen.", Toast.LENGTH_SHORT).show();
    e1.printStackTrace();
    finish();
}
```

Pen SDK Light runs only on Samsung devices. The `Spen.initialize()` method throws an SSDK UnsupportedException exception on other devices. Handle the SSDK UnsupportedException exception as shown in the sample code below.

If the device is not a Samsung device or if the device is a Samsung device that does not support S pen:

- Display a message that the device does not support Pen SDK Light.
- Call `finish()` to close the application.

If `SPen SDK Light 4.1 apk` is not installed or if it is not the latest version on the device:

- Display a message that prompts the user to install or update `SPen SDK Light 4.1 apk` and open the website to download the package.

```
private boolean processUnsupportedException(SSDK UnsupportedException e) {

    e.printStackTrace();
    int errType = e.getType();
    // If the device is not a Samsung device or if the device does not support Pen.
    if (errType == SSDK UnsupportedException.VENDOR_NOT_SUPPORTED
            || errType == SSDK UnsupportedException.DEVICE_NOT_SUPPORTED) {
        Toast.makeText(mContext, "This device does not support Spen.",
                Toast.LENGTH_SHORT).show();
        finish();
    }
    else if (errType == SSDK UnsupportedException.LIBRARY_NOT_INSTALLED) {
        // If SPen SDK Light  APK is not installed.
        showAlertDialog( "You need to install additional Spen software"
            +" to use this application."
            + "You will be taken to the installation screen."
```

```
            + "Restart this application after the software has been installed."
            , true);
    } else if (errType
            == SSDK UnsupportedException.LIBRARY_UPDATE_IS_REQUIRED) {
        // SPen SDK Light  APK must be updated.
        showAlertDialog( "You need to update your Spen software "
            + "to use this application."
            + " You will be taken to the installation screen."
            + " Restart this application after the software has been updated."
            , true);
    } else if (errType
            == SSDK UnsupportedException.LIBRARY_UPDATE_IS_RECOMMENDED) {
        // Update of SPen SDK Light  APK to an available new version is recommended.
        showAlertDialog( "We recommend that you update your Spen software"
            +" before using this application."
            + " You will be taken to the installation screen."
            + " Restart this application after the software has been updated."
            , false);
        return false;
    }
    return true;
}


private void showAlertDialog(String msg, final boolean closeActivity) {

    AlertDialog.Builder dlg = new AlertDialog.Builder(mContext);
    dlg.setIcon(getResources().getDrawable(
        android.R.drawable.ic_dialog_alert));
    dlg.setTitle("Upgrade Notification")
        .setMessage(msg)
        .setPositiveButton(android.R.string.yes,
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(
                    DialogInterface dialog, int which) {
                    // Go to the market website and install/update APK.
                    Uri uri = Uri.parse("market://details?id=" +
Spen.getSpenPackageName());
                    Intent intent = new Intent(Intent.ACTION_VIEW, uri);
                    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
                        | Intent.FLAG_ACTIVITY_CLEAR_TASK);
                    startActivity(intent);

                    dialog.dismiss();
                    finish();
                }
            })
        .setNegativeButton(android.R.string.no,
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(
                    DialogInterface dialog, int which) {
                    if(closeActivity == true) {
                        // Terminate the activity if APK is not installed.
                        finish();
                    }
                    dialog.dismiss();
                }
```

```
        })
        .setOnCancelListener(new DialogInterface.OnCancelListener() {
            @Override
            public void onCancel(DialogInterface dialog) {
                if(closeActivity == true) {
                    // Terminate the activity if APK is not installed.
                    finish();
                }
            }
        })
        .show();
    dlg = null;
}
```

4.1.1.2 Checking the Availability of S pen

To check if the device supports S pen:

1. Call `Spen.isFeatureEnabled(Spen.DEVICE_PEN)`.

If the method returns false, call `SpenSimpleSurfaceView.setToolTypeAction()`and set TOOL_FINGER to ACTION_STROKE to enable users to use their finger for input.

```
boolean isSpenFeatureEnabled = false;

        .........

isSpenFeatureEnabled = spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);

        .........

if(isSpenFeatureEnabled == false) {
mSpenSimpleSurfaceView.setToolTypeAction(SpenSimpleSurfaceView.TOOL_FINGER,
                SpenSimpleSurfaceView.ACTION_STROKE);
        Toast.makeText(mContext,
"Device does not support S pen. \n You can draw strokeswith your finger",
                Toast.LENGTH_SHORT).show();
}
```

4.1.1.3 Creating SpenSimpleSurfaceView

To createa drawing container in your application:

1. Create an SpenSimpleSurfaceView instance by passing your application Context to the SpenSimpleSurfaceView constructor.

2. Add the SpenSimpleSurfaceView instance to the main layout.

```
RelativeLayout spenViewLayout =(RelativeLayout) findViewById(R.id.spenViewLayout);
mSpenSimpleSurfaceView = new SpenSimpleSurfaceView(mContext);
if (mSpenSimpleSurfaceView == null) {
      finish();
```

```
}
spenViewLayout.addView(mSpenSimpleSurfaceView);
```

4.1.1.4 Connecting SpenPageDoc to SpenSimpleSurfaceView

To create a container to save input data from the SpenSimpleSurfaceView instance:

1. Create an SpenNoteDoc instance by passing your application Context and the width and height of the SpenSimpleSurfaceView instance to the SpenNoteDoc constructor. If Pen SDK Light fails to create a cache directory, an IOException is thrown.
2. Call `SpenPageDoc.setBackgroundColor()`to specify the background color. This method is recorded in the history stack when it is executed, and Pen SDK Light may not function properly when an Undo or Redo operation occurs as a result.
3. To avoid this issue, call `SpenPageDoc.clearHistory()` to clear the history stack.
4. Use the `SpenSimpleSurfaceView.setPageDoc()` method to connect the SpenPageDoc instance to your SpenSimpleSurfaceView instance.

```
try {
mSpenNoteDoc =new SpenNoteDoc(mContext, rect.width(), rect.height());
} catch (IOException e) {
     e.printStackTrace();
       finish();
} catch (Exception e) {
       e.printStackTrace();
       finish();
}
// After adding a page to NoteDoc, get an instance
// and set it as a member variable.
mSpenPageDoc = mSpenNoteDoc.appendPage();
mSpenPageDoc.setBackgroundColor(0xFFD6E6F5);
mSpenPageDoc.clearHistory();
// Set PageDoc to View.
mSpenSimpleSurfaceView.setPageDoc(mSpenPageDoc, true);
```

4.1.1.5 Preventing Memory Leaks

To prevent memory leaks:

1. Call `SpenNoteDoc.close()` and `SpenSimpleSurfaceView.close()` to close the SpenNoteDoc and SpenSimpleSurfaceView instances to prevent memory leaks when your application closes.

To discard the cache data for your SpenNoteDoc instance, call `SpenNoteDoc.close()` with the Boolean parameter set to true.

An exception is thrown if you refer to an SpenNoteDoc instance after you have closed it. You can close SpenNoteDoc and SpenSimpleSurfaceView in the `onDestroy()` method.

```
if(mSpenSimpleSurfaceView != null) {
mSpenSimpleSurfaceView.close();
mSpenSimpleSurfaceView = null;
}

if(mSpenNoteDoc != null) {
try {
mSpenNoteDoc.close();
       } catch (Exception e) {
           e.printStackTrace();
       }
mSpenNoteDoc = null;
}
```

Note

Once you create an SpenSimpleSurfaceView instance, Pen SDK Light sets the default action of TOOL_FINGER to ACTION_GESTURE. This activates the zoom in, zoom out, and pan with finger gestures. To disable the zoom and pan features, call SpenSimpleSurfaceView.setToolTypeAction() and set TOOL_FINGER to ACTION_NONE. By default, Pen SDK Light sets TOOL_SPEN to ACTION_STROKE, which allows the S pen to draw strokes on the screen.

```
mSpenSimpleSurfaceView.setToolTypeAction(SpenSimpleSurfaceView.TOOL_FINGER,
SpenSimpleSurfaceView.ACTION_NONE);
```

The following tables contain the available tools and actions for SpenSimpleSurfaceView.

Pen SDK Light supports the following tool types.

Tool type	Value	Description
TOOL_UNKNOWN	0	Unknown tool.
TOOL_FINGER	1	Human fingers.
TOOL_SPEN	2	S pen.
TOOL_MOUSE	3	Mouse.
TOOL_ERASER	4	Eraser tool.
TOOL_MULTI_TOUCH	5	Multi-touch.

Pen SDK Light supports the following action types.

Action type	Value	Description
ACTION_NONE	0	No action.
ACTION_GESTURE	1	Finger gesture.
ACTION_STROKE	2	Pen SDK Light stroke.
ACTION_ERASER	3	Eraser tool.
ACTION_STROKE_REMOVER	4	Erasing pen stroke.
ACTION_COLOR_PICKER	5	Color picker.

Note		
ACTION_SELECTION	6	Selection.

4.1.2. Adding Pen SDK Light Settings

You can add a pen settings button to your application for setting user preferences for the pen size, color and type.

The sample application implements the following features:

1. Pen settings button for setting user preferences.

When the button is clicked, the SpenSettingPenLayout view appears to allow the user to configure the settings for the Pen SDK Light. If the button is clicked again, the window closes.

Listener to launch a color picker. When the listener is called, the sample application applies the selected color to the settings.

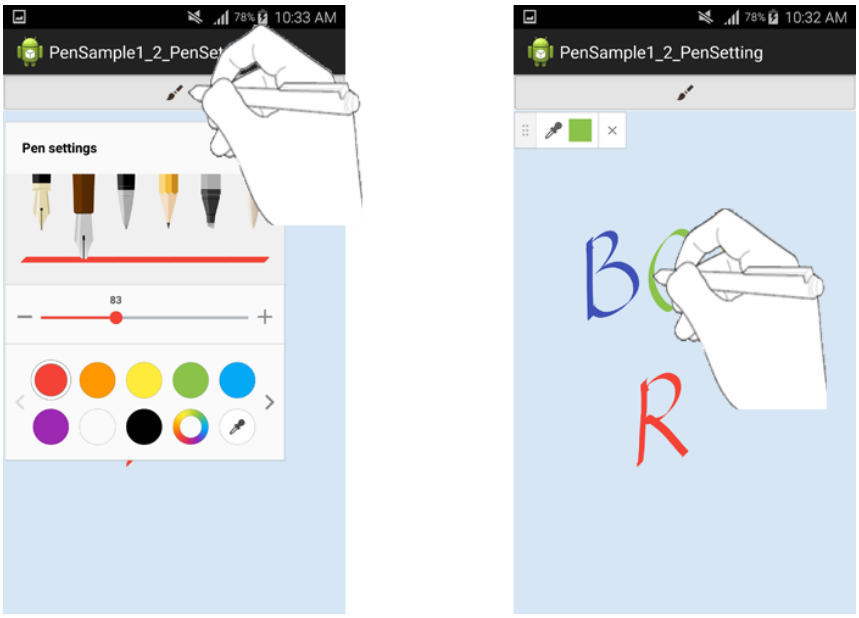


Figure 6: Pen SDK Light settings and color picker

```
public class PenSample1_2_PenSetting extends Activity {

    private Context mContext;
    private SpenNoteDoc mSpenNoteDoc;
    private SpenPageDoc mSpenPageDoc;
    private SpenSimpleSurfaceView mSpenSimpleSurfaceView;
    private SpenSettingPenLayout mPenSettingView;
    private LinearLayout mLayoutPreset;
    private ImageButton mBtnPreset;
```

```
    private Button mBtnEditPreset;
    private ImageButton mBtnAddPreset;
    private ImageView mPenBtn;
    private final float SCREEN_UNIT = 360.0f;
    private static final int PRESET_MODE_VIEW = 1;
    private static final int PRESET_MODE_EDIT = 2;
    private int mPresetViewMode;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pen_setting);
        mContext = getApplicationContext();

        // Initialize Spen
        boolean isSpenFeatureEnabled = false;
        Spen spenPackage = new Spen();
        try {
            spenPackage.initialize(this);
            isSpenFeatureEnabled = spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
        } catch (SSDK UnsupportedException e) {
            if (SDK Utils.processUnsupportedException(this, e) == true) {
                return;
            }
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Spen.",
Toast.LENGTH_SHORT).show();
            e1.printStackTrace();
            finish();
        }

        FrameLayout spenViewContainer = (FrameLayout)
findViewById(R.id.spenViewContainer);
        RelativeLayout spenViewLayout = (RelativeLayout)
findViewById(R.id.spenViewLayout);

        // Create PenSettingView
        mPenSettingView = new SpenSettingPenLayout(mContext, "", spenViewLayout);
        spenViewContainer.addView(mPenSettingView);

        // Create SpenView
        mSpenSimpleSurfaceView = new SpenSimpleSurfaceView(mContext);
        if (mSpenSimpleSurfaceView == null) {
            Toast.makeText(mContext, "Cannot create new SpenView.",
Toast.LENGTH_SHORT).show();
            finish();
        }
        mSpenSimpleSurfaceView.setToolTipEnabled(true);
        spenViewLayout.addView(mSpenSimpleSurfaceView);
        mPenSettingView.setCanvasView(mSpenSimpleSurfaceView);

        // Get the dimension of the device screen
        Display display = getWindowManager().getDefaultDisplay();
        Rect rect = new Rect();
        display.getRectSize(rect);
        try {
            mSpenNoteDoc = new SpenNoteDoc(mContext, rect.width(), rect.height());
        } catch (IOException e) {
            Toast.makeText(mContext, "Cannot create new NoteDoc",
```

```
Toast.LENGTH_SHORT).show();
            e.printStackTrace();
            finish();
        } catch (Exception e) {
            e.printStackTrace();
            finish();
        }
        // Add a Page to NoteDoc, get an instance, and set it to the member variable.
        mSpenPageDoc = mSpenNoteDoc.appendPage();
        mSpenPageDoc.setBackgroundColor(0xFFD6E6F5);
        mSpenPageDoc.clearHistory();
        // Set PageDoc to View
        mSpenSimpleSurfaceView.setPageDoc(mSpenPageDoc, true);

        initPenSettingInfo();
        mSpenSimpleSurfaceView.setColorPickerListener(mColorPickerListener);

        mPenBtn = (ImageView) findViewById(R.id.penBtn);
           mPenBtn.setOnClickListener(mPenBtnClickListener);

   mPenBtn = (ImageView) findViewById(R.id.penBtn);
   mPenBtn.setOnClickListener(mPenBtnClickListener);

        if (isSpenFeatureEnabled == false) {
            mSpenSimpleSurfaceView.setToolTypeAction(SpenSimpleSurfaceView.TOOL_FINGER,
SpenSimpleSurfaceView.ACTION_STROKE);
            Toast.makeText(mContext, "Device does not support Spen. \n You can draw
stroke by finger",
                    Toast.LENGTH_SHORT).show();
        } else {
            mSpenSimpleSurfaceView.setToolTypeAction(SpenSimpleSurfaceView.TOOL_SPEN,
SpenSimpleSurfaceView.ACTION_STROKE);
           }

    private void initPenSettingInfo() {
        // Initialize Pen settings
        SpenSettingPenInfo penInfo = new SpenSettingPenInfo();
        penInfo.color = Color.BLUE;
        penInfo.size = 10;
        mSpenSimpleSurfaceView.setPenSettingInfo(penInfo);
        mPenSettingView.setInfo(penInfo);
       }

   private final OnClickListener mPenBtnClickListener =
   new OnClickListener() {
   @Override
   public void onClick(View v) {
   // If PenSettingView is displayed, close PenSettingView.
   if (mPenSettingView.isShown()) {
   mPenSettingView.setVisibility(View.GONE);
   // If PenSettingView is not displayed, display PenSettingView.
                   } else {
   mPenSettingView
                           .setViewMode(SpenSettingPenLayout.VIEW_MODE_EXTENSION);
   mPenSettingView.setVisibility(View.VISIBLE);
                   }
               }
           };
```

```
    private SpenColorPickerListener mColorPickerListener =
    new SpenColorPickerListener() {
    @Override
    public void onChanged(int color, int x, int y) {
    // Insert the color from the color picker into SettingView.
    if (mPenSettingView != null) {
                        SpenSettingPenInfo penInfo = mPenSettingView.getInfo();
                        penInfo.color = color;
    mPenSettingView.setInfo(penInfo);
                    }
                }
            };

    @Override
    protected void onDestroy() {
    super.onDestroy();

    if (mPenSettingView != null) {
    mPenSettingView.close();
            }

    if(mSpenSimpleSurfaceView != null) {
    mSpenSimpleSurfaceView.close();
    mSpenSimpleSurfaceView = null;
            }

    if(mSpenNoteDoc != null) {
    try {
    mSpenNoteDoc.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
    mSpenNoteDoc = null;
            }
        };
}
```

For more information, see PenSample1_2_PenSetting.java in PenSample1_2_PenSetting.

```
public class SDK Utils {

    public static boolean processUnsupportedException(final Activity activity, SSDK
UnsupportedException e) {
        e.printStackTrace();
        int errType = e.getType();
        // If the device is not a Samsung device or the device does not support Pen.
        if (errType == SSDK UnsupportedException.VENDOR_NOT_SUPPORTED
                || errType == SSDK UnsupportedException.DEVICE_NOT_SUPPORTED) {
            Toast.makeText(activity, "This device does not support Spen.",
Toast.LENGTH_SHORT).show();
            activity.finish();
        } else if (errType == SSDK UnsupportedException.LIBRARY_NOT_INSTALLED) {
            // If SPen SDK Light  APK is not installed.
            showAlertDialog(activity, "You need to install additional Spen software"
                    + " to use this application."
                    + "You will be taken to the installation screen."
                    + "Restart this application after the software has been
installed.", true);
```

```
        } else if (errType == SSDK UnsupportedException.LIBRARY_UPDATE_IS_REQUIRED) {
            // SPen SDK Light  APK must be updated.
            showAlertDialog(activity, "You need to update your Spen software to use
this application."
                    + " You will be taken to the installation screen."
                    + " Restart this application after the software has been updated.",
true);
        } else if (errType == SSDK UnsupportedException.LIBRARY_UPDATE_IS_RECOMMENDED)
{
            // Recommended to update SPen SDK Light  APK to a new version available.
            showAlertDialog(activity, "We recommend that you update your Spen software"
                    + " before using this application." + " You will be taken to the
installation screen."
                    + " Restart this application after the software has been updated.",
false);
            return false; // Procceed to the normal activity process if it is not
updated.
        }
        return true;
    }

    private static void showAlertDialog(final Activity activity, String msg, final
boolean closeActivity) {

        AlertDialog.Builder dlg = new AlertDialog.Builder(activity);

dlg.setIcon(activity.getResources().getDrawable(android.R.drawable.ic_dialog_alert));
        dlg.setTitle("Upgrade Notification").setMessage(msg)
                .setPositiveButton(android.R.string.yes, new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        // Go to the market site and install or update APK.
                        Uri uri = Uri.parse("market://details?id=" +
Spen.getSpenPackageName());
                        Intent intent = new Intent(Intent.ACTION_VIEW, uri);
                        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TASK);
                        activity.startActivity(intent);

                        dialog.dismiss();
                        activity.finish();
                    }
                }).setNegativeButton(android.R.string.no, new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        if (closeActivity == true) {
                            // Terminate the activity if APK is not installed.
                            activity.finish();
                        }
                        dialog.dismiss();
                    }
                }).setOnCancelListener(new DialogInterface.OnCancelListener() {
                    @Override
                    public void onCancel(DialogInterface dialog) {
                        if (closeActivity == true) {
                            // Terminate the activity if APK is not installed.
                            activity.finish();
```

```
                        }
                    }
                }).show();
        dlg = null;
    }
```

For more information, see SDK Utils.java.

The following sections provide more details on the steps involved in adding settings.

4.1.2.1 Creating SpenSimpleSurfaceView and SpenSettingPenLayout

To create the layout for the penand settings view in your application:

1. Create the SpenSimpleSurfaceView and SpenSettingPenLayout instances.
2. To stack the SpenSettingPenLayout view on your SpenSimpleSurfaceView instance in the viewport, call `addView()` and add your SpenSettingPenLayout instance to the SpenSimpleSurfaceView container defined in FrameLayout.

To connect the settings view to your SpenSimpleSurfaceView instance, call `SpenSettingPenLayout.setCanvasView()` and pass your SpenSimpleSurfaceView instance.

```
mPenSettingView =new SpenSettingPenLayout(mContext, new String(),
                 spenViewLayout);
if (mPenSettingView == null) {
       finish();
}
spenViewContainer.addView(mPenSettingView);

mSpenSimpleSurfaceView = new SpenSimpleSurfaceView(mContext);
if (mSpenSimpleSurfaceView == null) {
       finish();
}
spenViewLayout.addView(mSpenSimpleSurfaceView);
mPenSettingView.setCanvasView(mSpenSimpleSurfaceView);
```

4.1.2.2 Initializing Pen Settings

To initialize the pen settings:

1. Create an SpenSettingPenInfo instance with your default settings for the pen tool.
2. Call `SpenSimpleSurfaceView.setPenSettingInfo()` to save the settings modified on your SpenSimpleSurfaceView instance.

Call `setInfo()` to save the settings to your SpenSettingPenLayout instance.

```
    private void initPenSettingInfo() {
        // Initialize Pen settings
        SpenSettingPenInfo penInfo = new SpenSettingPenInfo();
```

```
        penInfo.color = Color.BLUE;
        penInfo.size = 10;
        mSpenSimpleSurfaceView.setPenSettingInfo(penInfo);
        mPenSettingView.setInfo(penInfo);
    }
```

4.1.2.3 Registering a Listener for the Pen Settings Button

To handle pen Settings button events in your application:

1. Create a pen Settings button.

Create an `OnClickListener` listener instance, mPenBtnClickListener in the sample, for the pen Settings button and register it by calling `setOnClickListener()` on the button.

Handle the button click events.
If you are already displaying the SpenSettingPenLayout view, call `setVisibility(View.GONE)` to close the window.
If you are not displaying the view, call `setVisibility(View. VISIBLE)` to display the window.

In the sample code, setViewMode() is called to switch to the view mode with VIEW_MODE_EXTENSION. In this mode, you can:

- configure the type, size, and color of the pen tool
- select a preset to access frequently used pen types
- preview the configurations

```
    private final OnClickListener mPenBtnClickListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            // If PenSettingView is open, close it.
            if (mPenSettingView.isShown()) {
                mPenSettingView.setVisibility(View.GONE);
                // If PenSettingView is not open, open it.
            } else {
                mPenSettingView.setViewMode(SpenSettingPenLayout.VIEW_MODE_NORMAL);
                mPenSettingView.setVisibility(View.VISIBLE);
            }
        }
    };
```

Note
SpenSettingPenLayout provides the following view modes:

View mode	Value	Settings options for pen tool

Note

VIEW_MODE_NORMAL	0	Type, size, and color
VIEW_MODE_MINIMUM	1	Color, size
VIEW_MODE_EXTENSION	2	Color, type, size, preset, and preview
VIEW_MODE_EXTENSION_WITHOUT_PRESET	3	Color, type, size , and preview
VIEW_MODE_TYPE	4	Type of pen tool
VIEW_MODE_SIZE	5	Size
VIEW_MODE_COLOR	6	Color
VIEW_MODE_PRESET	7	Presets

4.1.2.4 Registering a Listener for Color Picking

Your application can use a color picker tool to pick a color from anywhere in the viewport and apply it to the pen color settings. The following figure shows how the color picker tool extracts a green color for insertion in the SpenSettingPenLayout view. Clicking the pen settings button displays the selection.

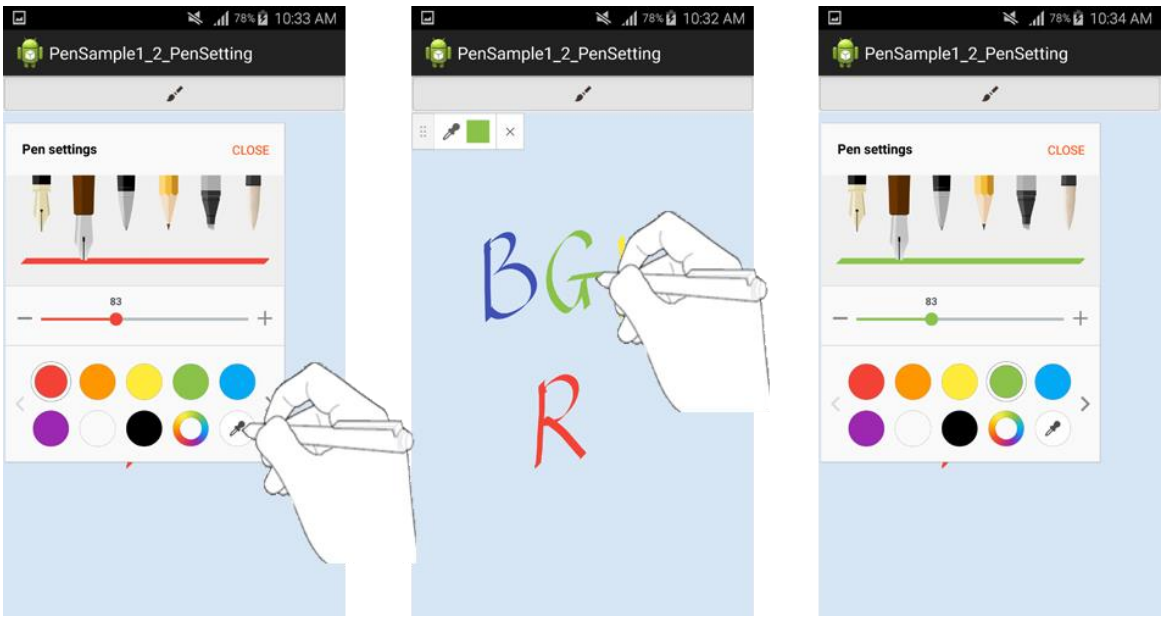


Figure 7: Color picker

To add a color picker to your application:

1. Create an `SpenColorPickerListener` listener instance, mColorPickerListener in the sample, for the color picker in the SpenSettingPenLayout view and handle the event it returns.

```
    private final SpenColorPickerListener mColorPickerListener = new
SpenColorPickerListener() {
        @Override
        public void onChanged(int color, int x, int y) {
            // Set the color from the Color Picker to the setting view.
            if (mPenSettingView != null) {
                SpenSettingPenInfo penInfo = mPenSettingView.getInfo();
                penInfo.color = color;
                mPenSettingView.setInfo(penInfo);
            }
        }
    };
```

Register the listener by calling `SpenSimpleSurfaceView.setColorPickerListener()`.

4.1.2.5 Preventing Memory Leaks

To prevent memory leaks:

1. Call `SpenSettingPenLayout.close()` to close your SpenSettingPenLayout instance to prevent memory leaks when your application closes. You can close SpenNoteDoc in the `onDestroy()` method.

```
if (mPenSettingView != null) {
mPenSettingView.close();
}
```

Note

Instead of using the SpenSettingPenLayout class methods provided in Pen, you can customize the pen settings for your application. After creating an SpenSettingPenInfo instance, you can select the options you need and call setPenSettingInfo() to register them. For example, if you want to add a blue marker for your application, add the following code to the `onClick()`method of the button.

```
    SpenSettingPenInfo penInfo = new SpenSettingPenInfo();
    penInfo.color = Color.BLUE;
    penInfo.alpha = 0x70;
    penInfo.size = 10;
    penInfo.name = "com.samsung.android.SDK .pen.pen.preload.Marker";
    mSpenSimpleSurfaceView.setPenSettingInfo(penInfo);
```

4.1.3. Adding Eraser Settings

You can add an eraser settings button to your application to configure the eraser settings and save the configurations to SpenSimpleSurfaceView with the SpenSettingEraserLayout class.

The sample application implements the following features:

1. Eraser Settings button for setting eraser preferences.

When the button is clicked, the SpenSettingEraserLayout view appears to allow the user to configure the eraser tool settings. The mode switches to eraser mode.

If the Clear All button is clicked, all the objects on the viewport are removed and the eraser settings window is closed.

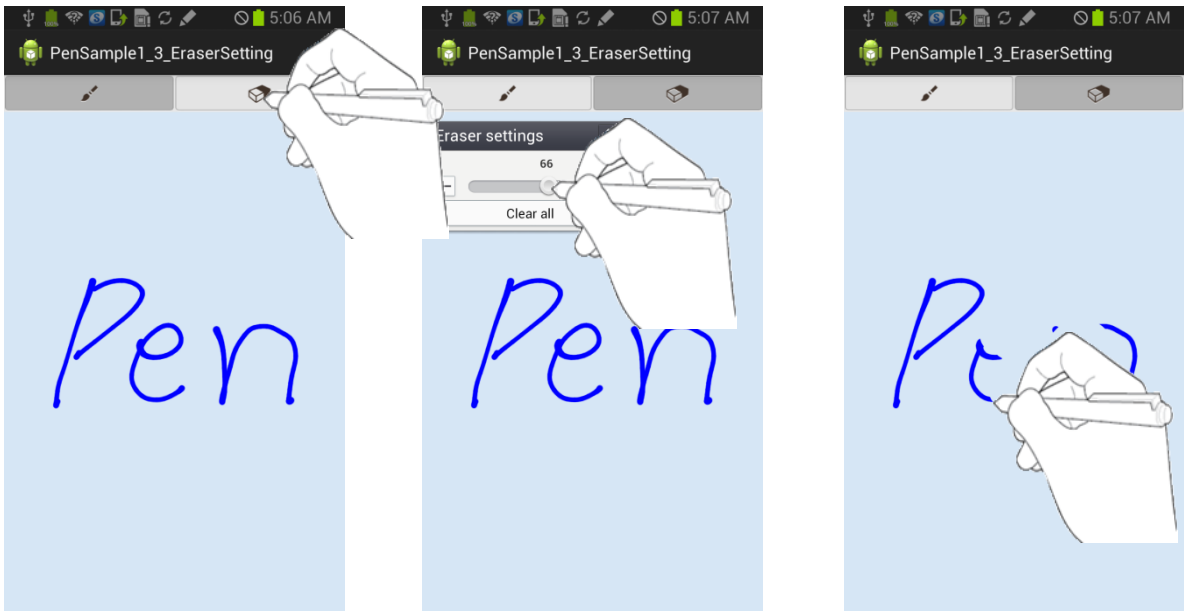


Figure 8: Eraser settings

```
   .........

private int mToolType = SpenSimpleSurfaceView.TOOL_SPEN;

protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_eraser_setting);
mContext = this;

        // Initialize Spen
        boolean isSpenFeatureEnabled = false;
        Spen spenPackage = new Spen();
        try {
            spenPackage.initialize(this);
            isSpenFeatureEnabled = spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
        } catch (SSDK UnsupportedException e) {
            if (SDK Utils.processUnsupportedException(this, e) == true) {
                return;
            }
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Spen.",
Toast.LENGTH_SHORT).show();
            e1.printStackTrace();
            finish();
```

```
        }

        FrameLayout spenViewContainer = (FrameLayout)
findViewById(R.id.spenViewContainer);
        RelativeLayout spenViewLayout = (RelativeLayout)
findViewById(R.id.spenViewLayout);

        // Create PenSettingView
        mPenSettingView = new SpenSettingPenLayout(getApplicationContext(), "",
spenViewLayout);

        // Create EraserSettingView
        mEraserSettingView = new SpenSettingEraserLayout(getApplicationContext(), "",
spenViewLayout);

        spenViewContainer.addView(mPenSettingView);
        spenViewContainer.addView(mEraserSettingView);

        // Create SpenSimpleSurfaceView
        mSpenSimpleSurfaceView = new SpenSimpleSurfaceView(mContext);
        if (mSpenSimpleSurfaceView == null) {
            Toast.makeText(mContext, "Cannot create new SpenSimpleSurfaceView.",
Toast.LENGTH_SHORT).show();
            finish();
        }
        mSpenSimpleSurfaceView.setToolTipEnabled(true);
        spenViewLayout.addView(mSpenSimpleSurfaceView);
        mPenSettingView.setCanvasView(mSpenSimpleSurfaceView);
        mEraserSettingView.setCanvasView(mSpenSimpleSurfaceView);

        // Get the dimension of the device screen.
        Display display = getWindowManager().getDefaultDisplay();
        Rect rect = new Rect();
        display.getRectSize(rect);
        // Create SpenNoteDoc
        try {
            mSpenNoteDoc = new SpenNoteDoc(mContext, rect.width(), rect.height());
        } catch (IOException e) {
            Toast.makeText(mContext, "Cannot create new NoteDoc",
Toast.LENGTH_SHORT).show();
            e.printStackTrace();
            finish();
        } catch (Exception e) {
            e.printStackTrace();
            finish();
        }
        // Add a Page to NoteDoc, get an instance, and set it to the member variable.
        mSpenPageDoc = mSpenNoteDoc.appendPage();
        mSpenPageDoc.setBackgroundColor(0xFFD6E6F5);
        mSpenPageDoc.clearHistory();
        // Set PageDoc to View
        mSpenSimpleSurfaceView.setPageDoc(mSpenPageDoc, true);

        initSettingInfo();
        // Register the listener
        mSpenSimpleSurfaceView.setColorPickerListener(mColorPickerListener);
        mEraserSettingView.setEraserListener(mEraserListener);

        // Set a button
```

```
        mPenBtn = (ImageView) findViewById(R.id.penBtn);
        mPenBtn.setOnClickListener(mPenBtnClickListener);

        mEraserBtn = (ImageView) findViewById(R.id.eraserBtn);
        mEraserBtn.setOnClickListener(mEraserBtnClickListener);

        selectButton(mPenBtn);

       if(isSpenFeatureEnabled == false) {
         mToolType = SpenSimpleSurfaceView.TOOL_FINGER;
mSpenSimpleSurfaceView.setToolTypeAction(mToolType,
          SpenSimpleSurfaceView.ACTION_STROKE);
            Toast.makeText(mContext,
                "Device does not support S pen. \n
                      You can draw strokeswith your finger",
            Toast.LENGTH_SHORT).show();
        }
}

private void initSettingInfo() {

              .........

        // Initialize Eraser settings
        SpenSettingEraserInfo eraserInfo = new SpenSettingEraserInfo();
        eraserInfo.size = 30;
        mSpenSimpleSurfaceView.setEraserSettingInfo(eraserInfo);
        mEraserSettingView.setInfo(eraserInfo);
    }
    private final OnClickListener mPenBtnClickListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            // When Spen is in stroke (pen) mode
            if (mSpenSimpleSurfaceView.getToolTypeAction(mToolType) ==
SpenSimpleSurfaceView.ACTION_STROKE) {
                // If PenSettingView is open, close it.
                if (mPenSettingView.isShown()) {
                    mPenSettingView.setVisibility(View.GONE);
                    // If PenSettingView is not open, open it.
                } else {
                    mPenSettingView.setViewMode(SpenSettingPenLayout.VIEW_MODE_NORMAL);
                    mPenSettingView.setVisibility(View.VISIBLE);
                }
                // If Spen is not in stroke (pen) mode, change it to stroke mode.
            } else {
                int curAction =
mSpenSimpleSurfaceView.getToolTypeAction(SpenSimpleSurfaceView.TOOL_FINGER);
                mSpenSimpleSurfaceView.setToolTypeAction(mToolType,
SpenSimpleSurfaceView.ACTION_STROKE);
                int newAction =
mSpenSimpleSurfaceView.getToolTypeAction(SpenSimpleSurfaceView.TOOL_FINGER);
                if (mToolType == SpenSimpleSurfaceView.TOOL_FINGER) {
                    if (curAction != newAction) {
                        selectButton(mPenBtn);
                    }
                } else {
                    selectButton(mPenBtn);
                }
            }
```

```
        }
    };

    private final OnClickListener mEraserBtnClickListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            // When Spen is in eraser mode
            if (mSpenSimpleSurfaceView.getToolTypeAction(mToolType) ==
SpenSimpleSurfaceView.ACTION_ERASER) {
                // If EraserSettingView is open, close it.
                if (mEraserSettingView.isShown()) {
                    mEraserSettingView.setVisibility(View.GONE);
                    // If EraserSettingView is not open, open it.
                } else {
                    mEraserSettingView.setVisibility(View.VISIBLE);
                }
                // If Spen is not in eraser mode, change it to eraser mode.
            } else {
                selectButton(mEraserBtn);
                mSpenSimpleSurfaceView.setToolTypeAction(mToolType,
SpenSimpleSurfaceView.ACTION_ERASER);
            }
        }
    };
              .........

private EventListener mEraserListener = new EventListener() {
@Override
public void onClearAll() {
//Handlethe Clear All button in EraserSettingView.
mSpenPageDoc.removeAllObject();
mSpenSimpleSurfaceView.update();
        }
};

private void selectButton(View v) {
// Depending on the current mode, enable or disable the button.
mPenBtn.setSelected(false);
mEraserBtn.setSelected(false);
        v.setSelected(true);

        closeSettingView();
}

private void closeSettingView() {
// Close all the setting views.
mEraserSettingView.setVisibility(SpenSimpleSurfaceView.GONE);
mPenSettingView.setVisibility(SpenSimpleSurfaceView.GONE);
}

@Override
protected void onDestroy() {
super.onDestroy();

if (mPenSettingView != null) {
mPenSettingView.close();
        }
if (mEraserSettingView != null) {
mEraserSettingView.close();
```

```
        }

if(mSpenSimpleSurfaceView != null) {
mSpenSimpleSurfaceView.close();
mSpenSimpleSurfaceView = null;
        }

if(mSpenNoteDoc != null) {
try {
mSpenNoteDoc.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
mSpenNoteDoc = null;
        }
}
```

For more information, see PenSample1_3_EraserSetting.java in PenSample1_3_EraserSetting.

The following sections provide more details on the steps involved in adding eraser settings.

4.1.3.1 Creating SpenSimpleSurfaceView and SpenSettingEraserLayout

To create a pen and eraser settings layout for your application:

1. Create an SpenSimpleSurfaceView instance and an SpenSettingEraserLayout instance. For more details, see the previous sections.

To stack the SpenSettingEraserLayout view on your SpenSimpleSurfaceView instance in the viewport, call `addView()` and add your SpenSettingEraserLayout view to the SpenSimpleSurfaceView container defined in FrameLayout.

To connect the eraser settings view to your SpenSimpleSurfaceView instance, call `SpenSettingEraserLayout.setCanvasView()` and pass your SpenSimpleSurfaceView instance.

```
mEraserSettingView =new SpenSettingEraserLayout(mContext, new String(),
                spenViewLayout);
if (mEraserSettingView == null) {
      Toast.makeText(mContext, "Cannot create new EraserSettingView.",
                Toast.LENGTH_SHORT).show();
      finish();
}
spenViewContainer.addView(mEraserSettingView);

.........

mEraserSettingView.setCanvasView(mSpenSimpleSurfaceView);
```

4.1.3.2 Initializing Eraser Settings

To initialize the eraser settings:

1. Create an SpenSettingEraserInfo instance with a default size for the eraser tool.
2. Call `SpenSimpleSurfaceView.setEraserSettingInfo()` to save the settings to your SpenSimpleSurfaceView instance.

Call `EraserSettingView.setInfo()` to save the settings to your SpenSettingEraserLayout instance.

```
SpenSettingEraserInfo eraserInfo = new SpenSettingEraserInfo();
eraserInfo.size = 30;
mSpenSimpleSurfaceView.setEraserSettingInfo(eraserInfo);
mEraserSettingView.setInfo(eraserInfo);
```

4.1.3.3 Registering a Listener for the Eraser Settings Button

To handle Eraser Settings button events:

1. Create an Eraser Settings button.
2. Create an `OnClickListener` listener instance for the Eraser Settings button, mEraserBtnClickListener in the sample, and register it by calling `setOnClickListener()` on the button.

Handle the button click events.
If you are already displaying the SpenSettingEraserLayout window, call `setVisibility(View.GONE)` to close the window.
If you are not displaying the window, call `setVisibility(View.VISIBLE)` to display the window.

```
private final OnClickListener mEraserBtnClickListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        // When Spen is in eraser mode
        if (mSpenSimpleSurfaceView.getToolTypeAction(mToolType) ==
SpenSimpleSurfaceView.ACTION_ERASER) {
            // If EraserSettingView is open, close it.
            if (mEraserSettingView.isShown()) {
                mEraserSettingView.setVisibility(View.GONE);
                // If EraserSettingView is not open, open it.
            } else {
                mEraserSettingView.setVisibility(View.VISIBLE);
            }
            // If Spen is not in eraser mode, change it to eraser mode.
        } else {
            selectButton(mEraserBtn);
            mSpenSimpleSurfaceView.setToolTypeAction(mToolType,
SpenSimpleSurfaceView.ACTION_ERASER);
        }
```

```
    }
};
                .........

private void selectButton(View v) {
    // Enable or disable the button according to the current mode.
    mPenBtn.setSelected(false);
    mEraserBtn.setSelected(false);
    v.setSelected(true);

    closeSettingView();
}
```

You can set the view mode by calling `setViewMode()` and passing VIEW_MODE_NORMAL. This allows you to configure the size of the eraser tool and to click the Clear All button.

Note

SpenSettingPenLayout offers you the following eraser view modes:

View mode	Value	View options
VIEW_MODE_NORMAL	0	Size slider and Clear All button
VIEW_MODE_TYPE	1	Eraser type option: Pen type for erasing objects
VIEW_MODE_SIZE	2	Size slider only

If your mToolType in your application is set to any action other than ACTION_ERASER,call `setToolTypeAction()`to change it to ACTION_ERASER. This changes the pen mode to eraser mode and the Eraser Tool button is displayed as selected. Initialize your mToolType variable to SpenSimpleSurfaceView.TOOL_SPEN on devices that support S pen or to SpenSimpleSurfaceView.TOOL_FINGER on the devices that do not support S pen.

4.1.3.4 Registering a Listener for ClearAll Events

When the Clear All button is clicked in the SpenSettingEraserLayout view, remove all the objects on the SpenSimpleSurfaceView instance and close the SpenSettingEraserLayout view.

To handle a Clear All event:

1. Create an `EventListener` instance, mEraserListener in the sample.
2. Handle the Clear All button event.

```
public void onClearAll() {
// Handle the ClearAll button inEraserSettingView.
mSpenPageDoc.removeAllObject();
mSpenSimpleSurfaceView.update();
}
```

Register the listener by calling `SpenSettingEraserLayout.setEraserListener()`.

4.1.3.5 Preventing Memory Leaks

To prevent memory leaks:

1. Call `SpenSettingEraserLayout.close`to close your SpenSettingEraserLayout instance in the `onDestroy()` method when your application closes.

```
if (mEraserSettingView != null) {
mEraserSettingView.close();
}
```

4.1.4. Adding Undo and Redo Commands

Pen SDK Light generates a history for each user action. History management lets you add undo or redo features to your application.

The sample application implements the following features:

- Undo and Redo buttons to go back or forward in the history stack.
- Listeners for the buttons to check if a state is available to execute the undo or redo commands, which results in the buttons being enabled or disabled.
- If the user clicks an Undo or Redo button, `SpenPageDoc.undo()` or `SpenPageDoc.redo()` retrieve the data and `SpenSimpleSurfaceView.updateUndo()` or `SpenSimpleSurfaceView.updateRedo()` update the SpenSimpleSurfaceViewinstance.
- History listener for receiving history state events.

Figure 9: Undo/Redo function

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_undo_redo);
mContext = this;

              .........

// Register the listeners.
mSpenSimpleSurfaceView.setColorPickerListener(mColorPickerListener);
mSpenPageDoc.setHistoryListener(mHistoryListener);
mEraserSettingView.setEraserListener(mEraserListener);

// Define the buttons.

              .........

mUndoBtn = (ImageView) findViewById(R.id.undoBtn);
mUndoBtn.setOnClickListener(undoNredoBtnClickListener);
mUndoBtn.setEnabled(mSpenPageDoc.isUndoable());

mRedoBtn = (ImageView) findViewById(R.id.redoBtn);
mRedoBtn.setOnClickListener(undoNredoBtnClickListener);
mRedoBtn.setEnabled(mSpenPageDoc.isRedoable());

        selectButton(mPenBtn);
}

              .........

private final OnClickListener undoNredoBtnClickListener =
new OnClickListener() {
@Override
public void onClick(View v) {
if (mSpenPageDoc == null) {
return;
                }
// Undo button is clicked.
if (v.equals(mUndoBtn)) {
if (mSpenPageDoc.isUndoable()) {
                        HistoryUpdateInfo[] userData = mSpenPageDoc.undo();
mSpenSimpleSurfaceView.updateUndo(userData);
                    }
// Redo button is clicked.
                } else if (v.equals(mRedoBtn)) {
if (mSpenPageDoc.isRedoable()) {
                        HistoryUpdateInfo[] userData = mSpenPageDoc.redo();
mSpenSimpleSurfaceView.updateRedo(userData);
                    }
                }
            }
};

            .........
```

```
private HistoryListener mHistoryListener = new HistoryListener() {
@Override
public void onCommit(SpenPageDoc page) {
        }

@Override
public void onUndoable(SpenPageDoc page, boolean undoable) {
// Enable or disable Undo button depending on its availability.
mUndoBtn.setEnabled(undoable);
        }

@Override
public void onRedoable(SpenPageDoc page, boolean redoable) {
// Enable or disable Redo button depending on its availability.
mRedoBtn.setEnabled(redoable);
        }
};

    .........
```

For more information, see PenSample1_4_UndoRedo.java in PenSample1_4_UndoRedo.

The following sections provide more details on the steps involved in adding undo and redo features.

4.1.4.1 Registering Listeners for the Undo and Redo Buttons

To handle Undo and Redo buttons events in your application:

1. Create Undo and Redo buttons.

Create an `OnClickListener` instance for the Undo and Redo buttons, undoNredoBtnClickListener in the sample, and register it by calling `setOnClickListener()`on each button.

In the Undo or Redo button click events, call `SpenPageDoc.isUndoable()` or `SpenPageDoc.isRedoable()` to check if data is available for the command.

Refresh the data of the SpenPageDoc instance by calling its `Undo()` or `Redo()` methods and refresh the viewport by calling `SpenSimpleSurfaceView.updateUndo()` or `SpenSimpleSurfaceView.updateRedo()`.

```
public void onClick(View v) {
if (mSpenPageDoc == null) {
return;
            }
// Undo button is clicked.
if (v.equals(mUndoBtn)) {
if (mSpenPageDoc.isUndoable()) {
                    HistoryUpdateInfo[] userData = mSpenPageDoc.undo();
mSpenSimpleSurfaceView.updateUndo(userData);
                }
// Redo button is clicked.
            } else if (v.equals(mRedoBtn)) {
if (mSpenPageDoc.isRedoable()) {
                    HistoryUpdateInfo[] userData = mSpenPageDoc.redo();
```

```
mSpenSimpleSurfaceView.updateRedo(userData);
            }
}
```

4.1.4.2 Registering a Listener for History

To handle history events in your application:

1. Create a `HistoryListener` instance, mHistoryListener in the sample, and register it by calling `SpenPageDoc.setHistoryListener()`.

2. Enable the Undo button when the `onUndoable()` method is called.

Enable the Redo button when the `onRedoable()` method is called.

```
private HistoryListener mHistoryListener = new HistoryListener() {
@Override
public void onCommit(SpenPageDoc page) {
        }

@Override
public void onUndoable(SpenPageDoc page, boolean undoable) {
// Enable or Disable Undo button depending on its availability.
mUndoBtn.setEnabled(undoable);
        }

@Override
public void onRedoable(SpenPageDoc page, boolean redoable) {
// Enable or Disable Redo button depending on its availability.
mRedoBtn.setEnabled(redoable);
        }
};
```

Note

Pen SDK Light limits the number of undoable states to 50. When the fifty first state is added to the history stack, the oldest state is removed. You can edit the limit by calling `SpenPageDoc.setUndoLimit()`.

Pen provides you the following undo and redo options:

- `undo()` and `redo()`: Undo and redo on a state-by-state basis.
- `undoAll()` and `redoAll()`: Undo and redo all the states in history.
- `undoToTag()` : Undo until the user-selected state.

To use `undoToTag()`, you need to tag the selected state by calling `SpenPageDoc.setHistoryTag()`. When you use `undoToTag()`,Pen SDK Light executes the undo operation up until the tagged state and you cannot redo the states. You can delete the tag using `clearTag()`.

The `undoToTag()` method undoes the data in SpenPageDoc. To refresh the viewport, call `SpenSimpleSurfaceView.updateUndoAll()`.

Note

```
HistoryUpdateInfo[] userDataList = mSpenPageDoc.undoToTag();
mSpenSimpleSurfaceView.updateUndoAll(userDataList);
```

4.1.5. Setting Backgrounds

You can select an image from the gallery and use it as the background for your application by using `SpenPageDoc.setBackgroundImage()`.

The sample application implements the following features:

- Background Setting button to select a background from the gallery.
- Listener for the button and an intent to call `startActivityForResult()` to allow selection of an image from the gallery.
- On image selection, it uses `SpenPageDoc.setBackgroundImage()`in the `onActivityResult()`callback method to set the background.

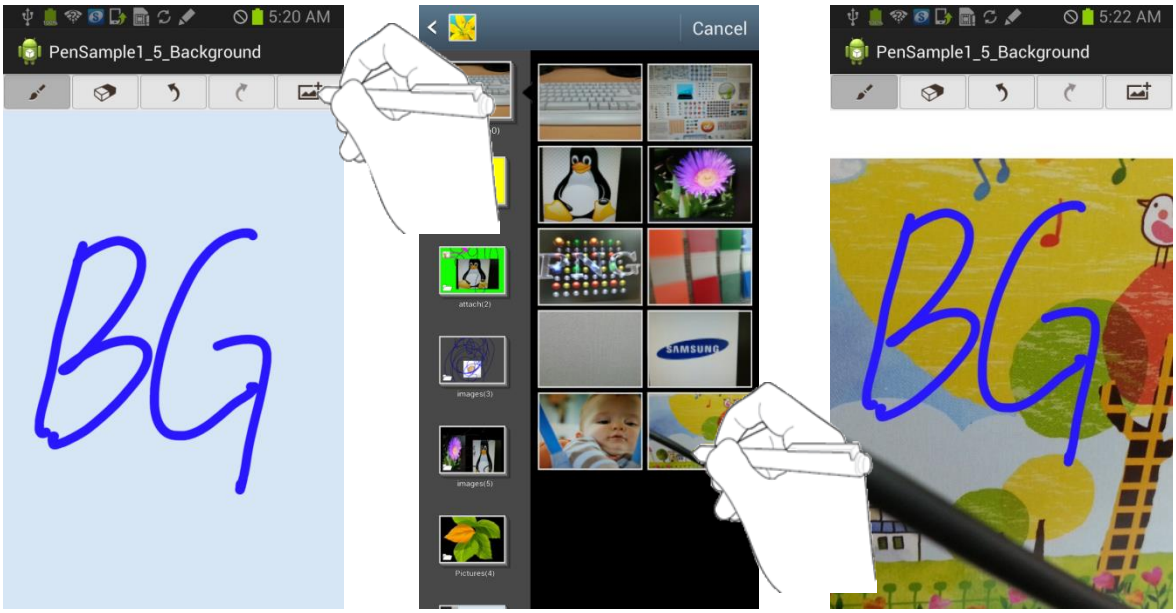


Figure 10: Background settings

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_background);
mContext = this;

              .........

mBgImgBtn = (ImageView) findViewById(R.id.bgImgBtn);
```

```
mBgImgBtn.setOnClickListener(mBgImgBtnClickListener);

        selectButton(mPenBtn);

            .........

private final OnClickListener mBgImgBtnClickListener =
new OnClickListener() {
@Override
public void onClick(View v) {
                closeSettingView();

                callGalleryForInputImage(REQUEST_CODE_SELECT_IMAGE_BACKGROUND);
            }
};

              .........

private void callGalleryForInputImage(int nRequestCode) {
// Get an image from the gallery.
try {
            Intent galleryIntent = new Intent(Intent.ACTION_GET_CONTENT);
            galleryIntent.setType("image/*");
            startActivityForResult(galleryIntent, nRequestCode);
        } catch (ActivityNotFoundException e) {
            Toast.makeText(mContext, "Cannot find gallery.",
                Toast.LENGTH_SHORT).show();
            e.printStackTrace();
        }
}

@Override
protected void Activity(int requestCode, int resultCode,
        Intent data) {
super.onActivityResult(requestCode, resultCode, data);

if (resultCode == RESULT_OK) {
if (data == null) {
                Toast.makeText(mContext, "Cannot find the image",
                    Toast.LENGTH_SHORT).show();
return;
            }

// Process image request for the background.
if (requestCode == REQUEST_CODE_SELECT_IMAGE_BACKGROUND) {
// Get the image's URI and use the location for background image.
                Uri imageFileUri = data.getData();
                Cursor cursor =
                    getContentResolver().query(
                        Uri.parse(imageFileUri.toString()), null, null,
null, null);
                cursor.moveToNext();
                String imagePath =
                    cursor.getString(cursor
                        .getColumnIndex(MediaStore.MediaColumns.DATA));

mSpenPageDoc.setBackgroundImage(imagePath);
mSpenSimpleSurfaceView.update();
            }
```

```
        }
}

              .........
```

For more information, see PenSample1_5_Background.java in PenSample1_5_Background.

The following sections provide more details on the steps involved in setting a background.

4.1.5.1 Registering a Listener for the Background Setting Button

To handle background settings events in your application:

1. Create a Background Setting button.
2. Create an `OnClickListener` instance for the Background Setting button, mBgImgBtnClickListener in the sample, and register it by calling `setOnClickListener()` on the button.

If you are displaying the Background Settings view when the button is clicked, close the window and call the private class that fetches the image.

```
public void onClick(View v) {
           closeSettingView();
           callGalleryForInputImage(REQUEST_CODE_SELECT_IMAGE_BACKGROUND);
}
```

To allow image selection from the gallery, create an intent to call `startActivityForResult()` in your private class.

```
Intent galleryIntent = new Intent(Intent.ACTION_GET_CONTENT);
galleryIntent.setType("image/*");
startActivityForResult(galleryIntent, nRequestCode);
```

4.1.5.2 Registering a Callback Function for Image Selection

To handle the callback function after an image is selected:

1. Use the `onActivityResult()`callback methodfor the image selection for the background.

Get the URI of the image file from the intent after checking if the resultCode is RESULT_OK.

Call `SpenPageDoc.setBackgroundImage()` to set the background image.

To refresh the background on the viewport, call `SpenSimpleSurfaceView.update()`.

```
if (requestCode == REQUEST_CODE_SELECT_IMAGE_BACKGROUND) {
// Get the image's URI and use the location for background image.
            Uri imageFileUri = data.getData();
            Cursor cursor =getContentResolver().query(
                    Uri.parse(imageFileUri.toString()), null, null, null, null);
            cursor.moveToNext();
            String imagePath =cursor.getString(cursor
                        .getColumnIndex(MediaStore.MediaColumns.DATA));

mSpenPageDoc.setBackgroundImage(imagePath);
mSpenSimpleSurfaceView.update();
}
```

Note

Pen SDK Light registers all the background image setting actions in history. You can undo both the background image from the application startup and the one set by the user. To prevent any unintentional background image changes, clear the history states by calling `clearHistory()`.

4.1.6. Capturing Screen Shots

Pen SDK Light allows you to take a screen shot and save it as an image file.

You can implement this function in your application using `SpenSimpleSurfaceView.captureCurrentView()`, which creates a bitmap from SpenSimpleSurfaceView.

The sample application implements the following features:

- Screen Capture button to take a screen shot.
- Listener for the button to allow the application to save the bitmap from `captureCurrentView()` as CaptureImg.png in the SPen/images folder in external memory.

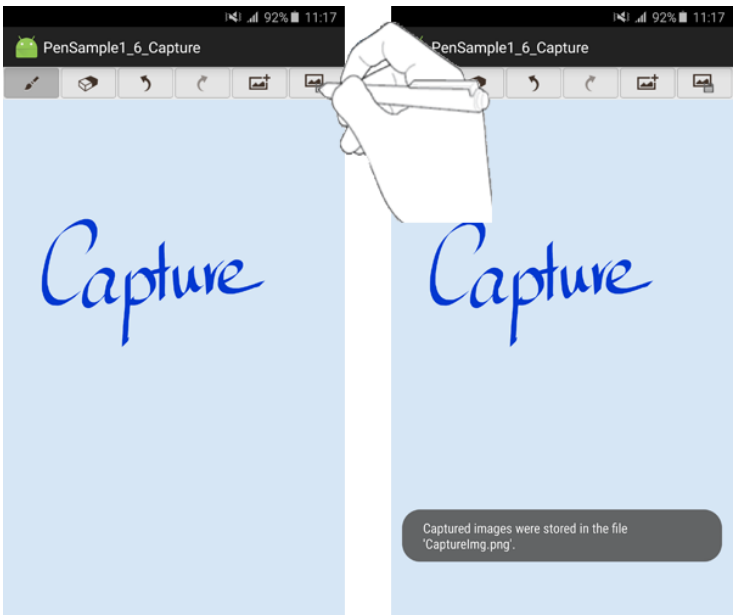


Figure 11: Capture function

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_capture);
mContext = this;

              .........

mCaptureBtn = (ImageView) findViewById(R.id.captureBtn);
mCaptureBtn.setOnClickListener(mCaptureBtnClickListener);

        selectButton(mPenBtn);

mSpenPageDoc.startRecord();


           .........

private final OnClickListener mCaptureBtnClickListener =
new OnClickListener() {
@Override
public void onClick(View v) {
                closeSettingView();
                captureSpenSimpleSurfaceView();
            }
};

              .........

private void captureSpenSimpleSurfaceView() {
// Select the location to save the image.
        String filePath = Environment.getExternalStorageDirectory()
            .getAbsolutePath() + "/SPen/images";
        File fileCacheItem = new File(filePath);
if (!fileCacheItem.exists()) {
if (!fileCacheItem.mkdirs()) {
                Toast.makeText(mContext, "Save Path Creation Error",
                    Toast.LENGTH_SHORT).show();
return;
            }
        }
        filePath = fileCacheItem.getPath() + "/CaptureImg.png";

// Save the screen shot as a Bitmap.
        Bitmap imgBitmap = mSpenSimpleSurfaceView.captureCurrentView(true);

        OutputStream out = null;
try {
// Save the Bitmap in the selected location.
            out = new FileOutputStream(filePath);
            imgBitmap.compress(CompressFormat.PNG, 100, out);
            Toast
                .makeText(
mContext,
"Captured images were stored in the file \'CaptureImg.png\'.",
```

```
                    Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
Toast
                .makeText(mContext, "Capture failed.", Toast.LENGTH_SHORT)
                .show();
            e.printStackTrace();
        } finally {
try {
if(out!= null) {
                    out.close();
                }
                sendBroadcast(new Intent(Intent.ACTION_MEDIA_MOUNTED,
Uri.parse("file://"
                        + Environment.getExternalStorageDirectory())));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        imgBitmap.recycle();
}

.........
```

For more information, see PenSample1_6_Capture.java in PenSample1_6_Capture.

The following sections provide more details on the steps involved in taking a screen shot.

4.1.6.1 Registering a Listener for the Screen Capture Button

To handle Screen Capture button events in your application:

1. Create a Screen Capture button.
2. Create an `OnClickListener` instance for the Screen Capture button, mCaptureBtnClickListener in the sample, and register it by calling `setOnClickListener()` on the button.
3. Specify the file name and path for the screen shot.

Call `SpenSimpleSurfaceView.captureCurrentView()` to take the screens shot.

Save the resulting Bitmap.

To register the saved file with the gallery, call `sendBroadcast()` Intent.ACTION_MEDIA_MOUNTED.

Call `recycle()`to prevent memory leaks.

```
private void captureSpenSimpleSurfaceView() {
// Select the location to save the image.

String filePath = Environment.getExternalStorageDirectory()
            .getAbsolutePath() + "/SPen/images";
        File fileCacheItem = new File(filePath);
if (!fileCacheItem.exists()) {
```

```
if (!fileCacheItem.mkdirs()) {
                Toast.makeText(mContext, "Save Path Creation Error",
                    Toast.LENGTH_SHORT).show();
return;
            }
        }
        filePath = fileCacheItem.getPath() + "/CaptureImg.png";

// Save the screen shot as a Bitmap.
        Bitmap imgBitmap = mSpenSimpleSurfaceView.captureCurrentView(true);

        OutputStream out = null;
try {
// Save the Bitmap in the selected location.
            out = new FileOutputStream(filePath);
            imgBitmap.compress(CompressFormat.PNG, 100, out);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
try {
if(out!= null) {
                    out.close();
}
sendBroadcast(new Intent(Intent.ACTION_MEDIA_MOUNTED,
                    Uri.parse("file://"
                        + Environment.getExternalStorageDirectory())));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        imgBitmap.recycle();
}
```

Note

You can also take screen shots of SpenPageDoc instances that are not connected to a SpenSimpleSurfaceView instance. Use the SpenCapturePage class to capture screens that do not have a View.

Call SpenCapturePage.setPageDoc()to specify which SpenPageDoc to capture and then call compressPage() with the file name. The screen shot of the SpenPageDoc is saved in PNG format. To prevent memory leaks, call close() after completion.

Do not use the SpenPageDoc instance connected to your SpenSimpleSurfaceView instance with the SpenCapturePage class methods.

4.2. Using Pen SDK Light Documents

A document is a Pen SDK Light component that:

- adds and deletes an object
- saves and opens a file
- manages history

Using the document methods, your application can offer the following features:

- add, delete, or save strokes objects as files
- add extra data or a file when saving an object
- open and edit a saved file
- manage history for undo and redo commands

Note that you cannot save history states as a file

4.2.1. Inserting Stroke Objects

Pen SDK Light offers you features to create stroke objects in your application.

The sample application implements the following features:

- Insert Stroke button to add a stroke each time the pen touches the viewport.
- Custom mode to add pen strokes.
- Listener for touch events.
- Stroke insertion and SpenPageDoc and viewport update.

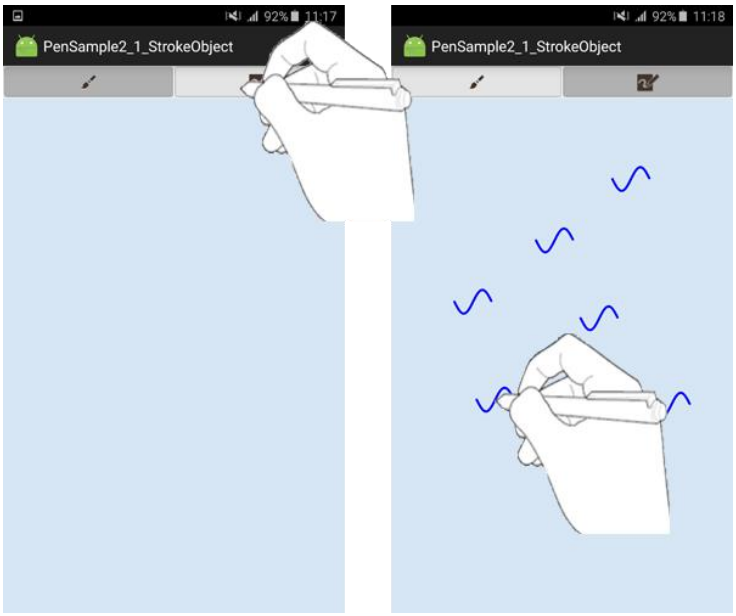


Figure 12: Stroke object

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_stroke_object);
```

```
mContext = this;

                .........

mSpenSimpleSurfaceView.setTouchListener(mPenTouchListener);

                .........

mStrokeObjBtn = (ImageView) findViewById(R.id.strokeObjBtn);
mStrokeObjBtn.setOnClickListener(mStrokeObjBtnClickListener);

        selectButton(mPenBtn);
}

                .........

private final SpenTouchListener mPenTouchListener = new SpenTouchListener() {

    @Override
    public boolean onTouch(View view, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_UP && event.getToolType(0) ==
mToolType) {
            // Check if the control is created.
            SpenControlBase control = mSpenSimpleSurfaceView.getControl();
            if (control == null) {
                if (mMode == MODE_STROKE_OBJ) {
                    // Set the location to insert ObjectStroke and add it to PageDoc.
                    PointF canvasPos = getCanvasPoint(event);
                    float posX = canvasPos.x;
                    int pointSize = 157;

                    PointF[] points = new PointF[pointSize];
                    float[] pressures = new float[pointSize];
                    int[] timestamps = new int[pointSize];

                    for (int i = 0; i < pointSize; i++) {
                        points[i] = new PointF();
                        points[i].x = posX++;
                        points[i].y = (float) (canvasPos.y + Math.sin(.04 * i) * 50);
                        pressures[i] = 1;
                        timestamps[i] = (int) android.os.SystemClock.uptimeMillis();
                    }

                    SpenObjectStroke strokeObj = new
SpenObjectStroke(mPenSettingView.getInfo().name, points,
                            pressures, timestamps);
                    strokeObj.setPenSize(mPenSettingView.getInfo().size);
                    strokeObj.setColor(mPenSettingView.getInfo().color);
                    mSpenPageDoc.appendObject(strokeObj);
                    mSpenSimpleSurfaceView.update();
                }
            }
        }
        return false;
    }
};              .........

private final OnClickListener mStrokeObjBtnClickListener = new OnClickListener() {
    @Override
```

```
    public void onClick(View v) {
        mSpenSimpleSurfaceView.closeControl();

        mMode = MODE_STROKE_OBJ;
        selectButton(mStrokeObjBtn);
        mSpenSimpleSurfaceView.setToolTypeAction(mToolType,
SpenSimpleSurfaceView.ACTION_NONE);
    }
};
              .........

private void selectButton(View v) {
    // Enable or disable the button according to the current mode.
    mPenBtn.setSelected(false);
    mStrokeObjBtn.setSelected(false);

    v.setSelected(true);
    closeSettingView();
}
          .........
```

For more information, see PenSample2_3_StrokeObject.java in PenSample2_3_StrokeObject.

The following sections provide more details on the steps involved in adding strokes in your application.

4.2.1.1 Registering a Listener for the Insert Stroke Button

To handle Insert Stroke button events:

1. Create an Insert Stroke button.
2. Create an `OnClickListener` listener instance for the Insert Stroke button, mStrokeObjBtnClickListener in the sample, and register it by calling `setOnClickListener()` on the button.

In the `onClick()` method, set`mToolType`to ACTION_NONE, use the internal application stroke mode, and indicate that the Insert Stroke button is selected.

```
mMode = MODE_STROKE_OBJ;
selectButton(mStrokeObjBtn);
mSpenSimpleSurfaceView.setToolTypeAction(mToolType,
                    SpenSimpleSurfaceView.ACTION_NONE);
```

4.2.1.2 Creating and Registering a Touch Event Listener

To handle touch events in your application in stroke mode:

1. Create an `SpenTouchListener` listener instance, mPenTouchListener in the sample, and implement the `onTouch()` callback method for pen touch events in the View area.
2. Call `SpenSimpleSurfaceView.setTouchListener()` to register the listener.

In the onTouch () method,if the SpenSimpleSurfaceView tool type is S pen and the application internal mode is Insert Stroke, implement the following:

- Calculate the coordinates of the stroke based on the location where the event took place.
- Set the time stamp with the system clock and set Pressure to 1.
- Get the pen name from the settings information for use as an input variable to call SpenObjectStroke().
- Call SpenObjectStroke() to create a stroke object. In this case, the size of an array of 'points' and 'pressures' must always be same; otherwise, an exception will be thrown.
- From the setting information, get the pen size and color for the new object and call setPenSize() and setColor() to set them.
- Call SpenPageDoc.appendObject() to insert the stroke object.
- Call SpenSimpleSurfaceView.update() to refresh the screen.

```
if (mMode == MODE_STROKE_OBJ) {
                        // Set the location to insert ObjectStroke and add it to
PageDoc.
                        PointF canvasPos = getCanvasPoint(event);
                        float posX = canvasPos.x;
                        int pointSize = 157;

                        PointF[] points = new PointF[pointSize];
                        float[] pressures = new float[pointSize];
                        int[] timestamps = new int[pointSize];

                        for (int i = 0; i < pointSize; i++) {
                            points[i] = new PointF();
                            points[i].x = posX++;
                            points[i].y = (float) (canvasPos.y + Math.sin(.04 * i) *
50);
                            pressures[i] = 1;
                            timestamps[i] = (int)
android.os.SystemClock.uptimeMillis();
                        }

                        SpenObjectStroke strokeObj = new
SpenObjectStroke(mPenSettingView.getInfo().name, points,
                                pressures, timestamps);
                        strokeObj.setPenSize(mPenSettingView.getInfo().size);
                        strokeObj.setColor(mPenSettingView.getInfo().color);
                        mSpenPageDoc.appendObject(strokeObj);
                        mSpenSimpleSurfaceView.update();
                    }
```

Note

If ToolTypeAction is set to ACTION_NONE, ACTION_GESTURE, ACTION_SELECTION, or ACTION_TEXT, a touch-to-zoom animation will be performed by GestureDetector.OnDoubleTapListener.onDoubleTap() method when a double-tap gesture occurs

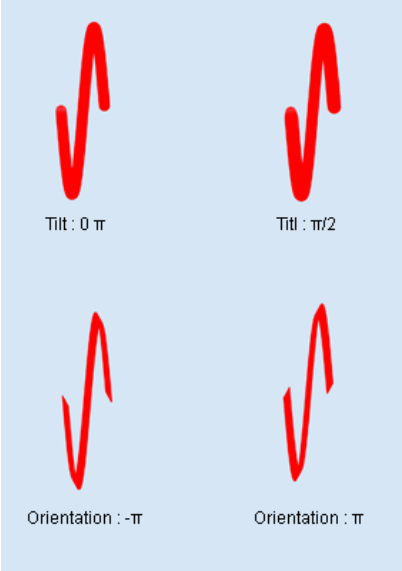
Note

Refer to the tips below if you don't want a touch-to-zoom animation.

- Do not set ToolTypeAction to ACTION_NONE, ACTION_GESTURE, ACTION_SELECTION, or ACTION_TEXT.
- Or, use `SpenSimpleSurfaceView.setPreTouchListener()` method to receive and consume pre-touch events.

If the current used pen type is foutain pen, you can use the following API to add tilt/orientation infomation to the stroke to customize stroke's style:

```
publicSpenObjectStroke(String penName, PointF[] points, float[]
pressures, int[] timestamps, float[] tilts,float[] orientations)
```



4.2.2. Saving Files

The sample application saves the data created with Pen SDK Light in a file. The application supports the SPD format for Pen SDK Light data files and the +SPD data format (image file with added SPD data) for general image files.

Typical drawing applications display files saved in an image format as images but applications using Pen SDK Light can read them in the SPD data format. When the image data is modified with common editing tools, Pen SDK Light applications can no longer open them because these modifications remove the SPD data from the image data.

The sample application implements the following features:

- Save File button for saving files.
- When this button is clicked, a view allows users to name the file and select a format - SPD or PNG.

- When the file is saved in SPD format, it is saved with the provided file name.
- When the file is saved in PN G, a Bitmap is generated first and then it is saved. The SPD data behind the image is then added.

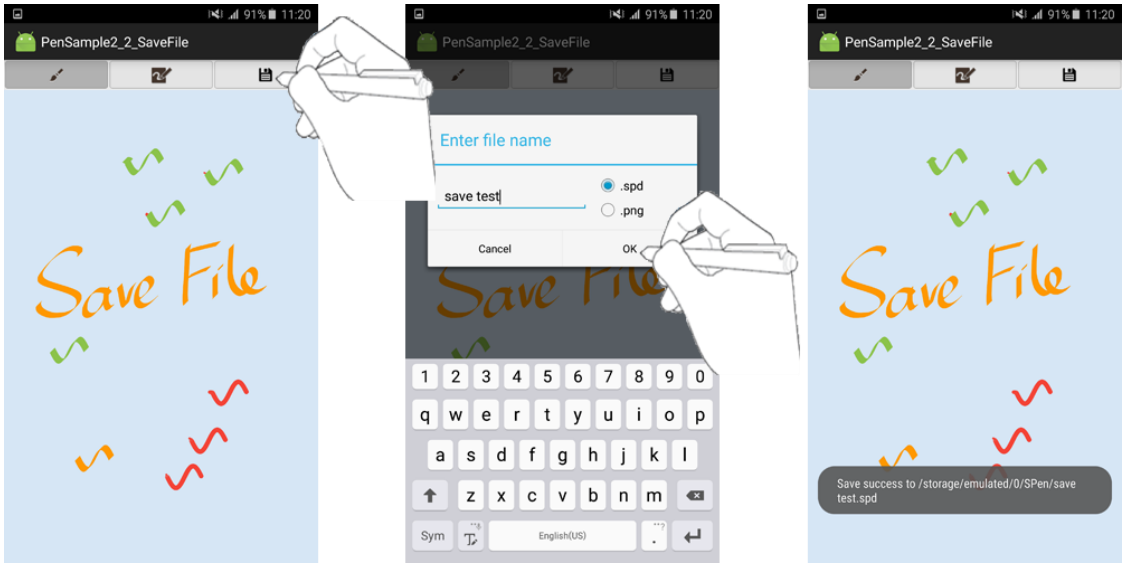


Figure 13: File save function

```
public class PenSample2_2_SaveFile extends Activity {
                .........

private boolean isDiscard = false;

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_save_file);
    mContext = this;
                .........

mSaveFileBtn = (ImageView) findViewById(R.id.saveFileBtn);
mSaveFileBtn.setOnClickListener(mSaveFileBtnClickListener);

        selectButton(mPenBtn);
    }

                .........

private final OnClickListener mSaveFileBtnClickListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        mSpenSimpleSurfaceView.closeControl();
        closeSettingView();
        saveNoteFile(false);
    }
};
private boolean saveNoteFile(final boolean isClose) {
    // Prompt Save File dialog to get the file name
    // and get its save format option (note file or image).
```

```
    LayoutInflater inflater = (LayoutInflater)
mContext.getSystemService(LAYOUT_INFLATER_SERVICE);
    final View layout = inflater.inflate(R.layout.save_file_dialog, (ViewGroup)
findViewById(R.id.layout_root));

    AlertDialog.Builder builderSave = new AlertDialog.Builder(mContext);
    builderSave.setTitle("Enter file name");
    builderSave.setView(layout);

    final EditText inputPath = (EditText) layout.findViewById(R.id.input_path);
    inputPath.setText("Note");

    builderSave.setPositiveButton("OK", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {

            final RadioGroup selectFileExt = (RadioGroup)
layout.findViewById(R.id.radioGroup);

            // Set the save directory for the file.
            File filePath = new
File(Environment.getExternalStorageDirectory().getAbsolutePath() + "/SPen/");
            if (!filePath.exists()) {
                if (!filePath.mkdirs()) {
                    Toast.makeText(mContext, "Save Path Creation Error",
Toast.LENGTH_SHORT).show();
                    return;
                }
            }
            String saveFilePath = filePath.getPath() + '/';
            String fileName = inputPath.getText().toString();
            if (!fileName.equals("")) {
                saveFilePath += fileName;
                int checkedRadioButtonId = selectFileExt.getCheckedRadioButtonId();
                if (checkedRadioButtonId == R.id.radioNote) {
                    saveFilePath += ".spd";
                    saveNoteFile(saveFilePath);
                } else if (checkedRadioButtonId == R.id.radioImage) {
                    saveFilePath += ".png";
                    captureSpenSimpleSurfaceView(saveFilePath);
                }
                if (isClose) {
                    finish();
                }
            } else {
                Toast.makeText(mContext, "Invalid filename !!!",
Toast.LENGTH_LONG).show();
            }
        }
    });
    builderSave.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            if (isClose) {
                finish();
            }
        }
    });
```

```
    AlertDialog dlgSave = builderSave.create();
    dlgSave.show();

    return true;
}
private boolean saveNoteFile(String strFileName) {
    try {
        // Save NoteDoc
        mSpenNoteDoc.save(strFileName, false);
        Toast.makeText(mContext, "Save success to " + strFileName,
Toast.LENGTH_SHORT).show();
    } catch (IOException e) {
        Toast.makeText(mContext, "Cannot save NoteDoc file.",
Toast.LENGTH_SHORT).show();
        e.printStackTrace();
        return false;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
    return true;
}
private void captureSpenSimpleSurfaceView(String strFileName) {

    // Capture the view
    Bitmap imgBitmap = mSpenSimpleSurfaceView.captureCurrentView(true);
    if (imgBitmap == null) {
        Toast.makeText(mContext, "Capture failed." + strFileName,
Toast.LENGTH_SHORT).show();
        return;
    }

    OutputStream out = null;
    try {
        // Create FileOutputStream and save the captured image.
        out = new FileOutputStream(strFileName);
        imgBitmap.compress(CompressFormat.PNG, 100, out);
        // Save the note information.
        mSpenNoteDoc.save(out, false);
        out.close();
        Toast.makeText(mContext, "Captured images were stored in the file" +
strFileName, Toast.LENGTH_SHORT)
                .show();
    } catch (IOException e) {
        File tmpFile = new File(strFileName);
        if (tmpFile.exists()) {
            tmpFile.delete();
        }
        Toast.makeText(mContext, "Failed to save the file.",
Toast.LENGTH_SHORT).show();
        e.printStackTrace();
    } catch (Exception e) {
        File tmpFile = new File(strFileName);
        if (tmpFile.exists()) {
            tmpFile.delete();
        }
        Toast.makeText(mContext, "Failed to save the file.",
Toast.LENGTH_SHORT).show();
        e.printStackTrace();
```

```
    }
    imgBitmap.recycle();
}
            ……

@Override
public void onBackPressed() {
    if (mSpenPageDoc.getObjectCount(true) > 0 && mSpenPageDoc.isChanged()) {
        AlertDialog.Builder dlg = new AlertDialog.Builder(mContext);

dlg.setIcon(mContext.getResources().getDrawable(android.R.drawable.ic_dialog_alert));
        dlg.setTitle(mContext.getResources().getString(R.string.app_name))
                .setMessage("Do you want to exit after save?")
                .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        saveNoteFile(true);
                        dialog.dismiss();
                    }
                }).setNeutralButton("No", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
                        mIsDiscard = true;
                        finish();
                    }
                }).setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
                    }
                }).show();
    } else {
        super.onBackPressed();
    }
}
@Override
protected void onDestroy() {
    super.onDestroy();

    if (mPenSettingView != null) {
        mPenSettingView.close();
    }

    if (mSpenSimpleSurfaceView != null) {
        mSpenSimpleSurfaceView.closeControl();
        mSpenSimpleSurfaceView.close();
        mSpenSimpleSurfaceView = null;
    }

    if (mSpenNoteDoc != null) {
        try {
            if (mIsDiscard) {
                mSpenNoteDoc.discard();
            } else {
                mSpenNoteDoc.close();
            }
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
        mSpenNoteDoc = null;
    }
};
```

For more information, see PenSample2_4_SaveFile.java in PenSample2_4_SaveFile.

The following sections provide more details on the steps involved in saving a file.

Note

From SPen SDK Light 4.0, we support compatibleMode for saving notedoc and saving pagedoc. And the performance will be bad if compatibleMode of saving notedoc and saving pagedoc are different. Because the default compatibleMode for both save notedoc data and save pagedoc data is true so if you want to save the notedoc without compatibleMode, you should use:

```
SpenPageDoc.setDefaultSaveOption(false);
```

4.2.2.1 Registering a Listener for the Save File Button

To handle Save File button events in your application:

1. Create a Save File button.
2. Create an`OnClickListener` listener instance for the Save File button, mSaveFileBtnClickListener in the sample, and register it by calling `setOnClickListener()`onthe button.

In the `onClick`method for the Save File button, close the properties view in the viewport, and call the `saveNoteFile()`method to generate a dialog to allow the user to save the files. Pass the Boolean value false to not close the application after files are saved. In the dialog, the user specifies the name and the extension (SPD or PNG) for the file.

```
closeSettingView();
saveNoteFile(false);
```

To savea file in SPD format, pass the file name to the `SpenNoteDoc.save()` method. Pen SDK Light stores the file in the "SPen/" folder in external storage.

```
private boolean saveNoteFile(String strFileName) {
    try {
        // Save NoteDoc
        mSpenNoteDoc.save(strFileName, false);
        Toast.makeText(mContext, "Save success to " + strFileName,
Toast.LENGTH_SHORT).show();
    } catch (IOException e) {
        Toast.makeText(mContext, "Cannot save NoteDoc file.",
Toast.LENGTH_SHORT).show();
```

```
            e.printStackTrace();
            return false;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
        return true;
    }
```

To saveafile in PNG format:

- Call `SpenSimpleSurfaceView.captureCurrentView()` to get the SpenSimpleSurfaceView bitmap.
- Encode it in an image format.
- Create a FileOuputStream with the file name.
- Save the encoded image to this stream.
- Pass this stream to the `SpenNoteDoc.save()` method to add the SPD data behind the image.

Call `recycle()` to avoid possible memory leaks.

```
private void captureSpenSimpleSurfaceView(String strFileName) {

    // Capture the view
    Bitmap imgBitmap = mSpenSimpleSurfaceView.captureCurrentView(true);
    if (imgBitmap == null) {
        Toast.makeText(mContext, "Capture failed." + strFileName,
Toast.LENGTH_SHORT).show();
        return;
    }

    OutputStream out = null;
    try {
        // Create FileOutputStream and save the captured image.
        out = new FileOutputStream(strFileName);
        imgBitmap.compress(CompressFormat.PNG, 100, out);
        // Save the note information.
        mSpenNoteDoc.save(out, false);
        out.close();
        Toast.makeText(mContext, "Captured images were stored in the file" +
strFileName, Toast.LENGTH_SHORT)
                .show();
    } catch (IOException e) {
        File tmpFile = new File(strFileName);
        if (tmpFile.exists()) {
            tmpFile.delete();
        }
        Toast.makeText(mContext, "Failed to save the file.",
Toast.LENGTH_SHORT).show();
        e.printStackTrace();
    } catch (Exception e) {
        File tmpFile = new File(strFileName);
        if (tmpFile.exists()) {
            tmpFile.delete();
        }
        Toast.makeText(mContext, "Failed to save the file.",
```

```
Toast.LENGTH_SHORT).show();
        e.printStackTrace();
    }
    imgBitmap.recycle();
}
```

Note

If you have a image file in sdcard/gallery, you can easily save notedoc to this file by using the notedoc API

```
public void attachToFile(String filePath)
```

Or:

```
public void attachToFile(String filePath, boolean compatibleMode)
```

4.2.2.2 Handling Back Key Events

To handle Back key events:

1. In the method handling Back key presses, if `SpenPageDoc.getObjectCount()` returns a value greater than 0 and `SpenPageDoc.isChanged()` returns true, create a dialog prompting the user to confirm the saving of the file.

If the user chooses to save the file, save the file and call `saveNoteFile()`with the Boolean value set to true to close the application.

```
@Override
public void onBackPressed() {
    if (mSpenPageDoc.getObjectCount(true) > 0 && mSpenPageDoc.isChanged()) {
        AlertDialog.Builder dlg = new AlertDialog.Builder(mContext);

dlg.setIcon(mContext.getResources().getDrawable(android.R.drawable.ic_dialog_alert));
        dlg.setTitle(mContext.getResources().getString(R.string.app_name))
                .setMessage("Do you want to exit after save?")
                .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        saveNoteFile(true);
                        dialog.dismiss();
                    }
```

If the user chooses not to save the file, call the following methods:

- The `onDestroy()` callback method to cancel the change in the SpenNoteDoc.
- `SpenNoteDoc.discard()` to close the SpenNoteDoc without saving the file.
- `finish()` to close the application.

```
.setNeutralButton("No", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
                        mIsDiscard = true;
                        finish();
                    }
                })
```

If the user selects Cancel in the dialog, close the dialog and return the application to its previous state.

```
.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
                    }
                }).show();
        dlg = null;
```

If the user selects No in the dialog:

- Check if `isDiscard` is set to true.
- If it is set to true, call `SpenNoteDoc.discard()` to cancel the change in the SpenNoteDoc stored in the cache and close the dialog.

```
protected void onDestroy() {
              .........

    if (mSpenNoteDoc != null) {
        try {
            if (mIsDiscard) {
                mSpenNoteDoc.discard();
            } else {
                mSpenNoteDoc.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        mSpenNoteDoc = null;
    }
};
```

4.2.3. Loading SPD and +SPD Files

You can use Pen SDK Light to load files saved in SPD (Pen SDK Light data files) and +SPD formats (image fileswith added SPD data).

The sample application implements the following features:

- Load File button for loading files.
- When this button is clicked, it saves the active note (the one the user is working with) as “tempNote.spd”.
- Displays a view that shows a list of the SPD and PNG files located in the “SPen/” folder in external storage.
- Creates an SpenNoteDoc instance with the selected file to refresh the screen with the loaded SpenPageDoc.

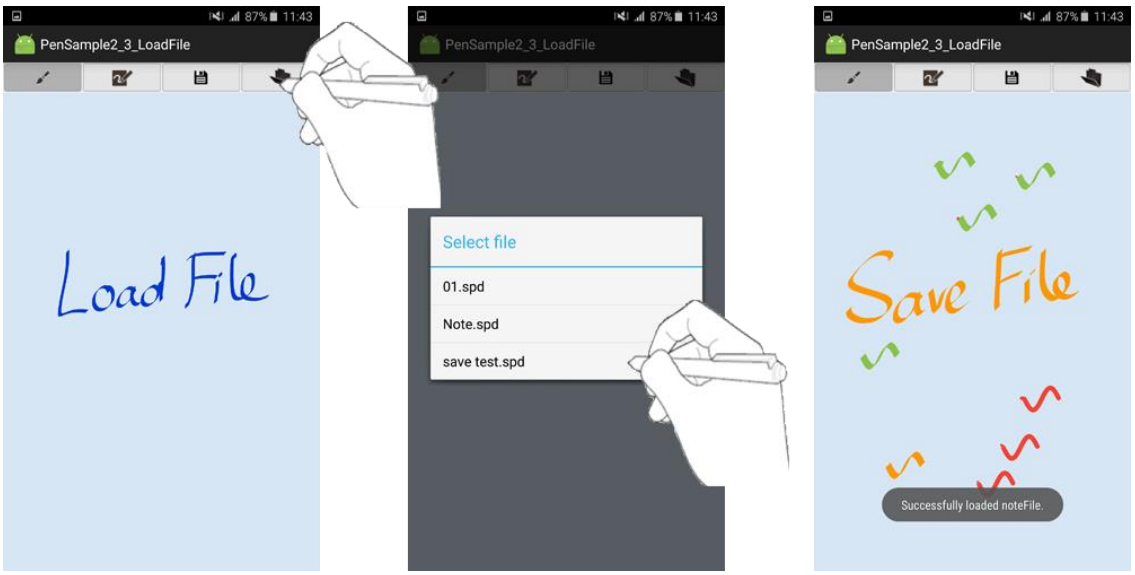


Figure 14: File load function

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_load_file);
    mContext = this;
              .........

    mLoadFileBtn = (ImageView) findViewById(R.id.loadFileBtn);
    mLoadFileBtn.setOnClickListener(mLoadFileBtnClickListener);
    selectButton(mPenBtn);
    initShapeSelectionDialog();

    String filePath = Environment.getExternalStorageDirectory().getAbsolutePath() +
                                                                    "/SPen/";
    mFilePath = new File(filePath);
    if (!mFilePath.exists()) {
        if (!mFilePath.mkdirs()) {
            Toast.makeText(mContext, "Save Path Creation Error",
                                            Toast.LENGTH_SHORT).show();
            return;
        }
    }
              .........
```

```
private final OnClickListener mLoadFileBtnClickListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        mSpenSimpleSurfaceView.closeControl();

        closeSettingView();
        loadNoteFile();
    }
};
              .........

private void loadNoteFile() {
    // Load the file list.
    final String fileList[] = setFileList();
    if (fileList == null) {
        return;
    }

    // Prompt Load File dialog.
    new AlertDialog.Builder(mContext).setTitle("Select file")
            .setItems(fileList, new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    String strFilePath = mFilePath.getPath() + '/' + fileList[which];

                    try
{  SpenObjectTextBox.setInitialCursorPos(SpenObjectTextBox.CURSOR_POS_END);
                        // Create NoteDoc with the selected file.
                        SpenNoteDoc tmpSpenNoteDoc = new SpenNoteDoc(mContext,
strFilePath, mScreenRect.width(),
                                SpenNoteDoc.MODE_WRITABLE, true);
                        mSpenNoteDoc.close();
                        mSpenNoteDoc = tmpSpenNoteDoc;
                        if (mSpenNoteDoc.getPageCount() == 0) {
                            mSpenPageDoc = mSpenNoteDoc.appendPage();
                        } else {
                            mSpenPageDoc =
mSpenNoteDoc.getPage(mSpenNoteDoc.getLastEditedPageIndex());
                        }
                        mSpenSimpleSurfaceView.setPageDoc(mSpenPageDoc, true);
                        mSpenSimpleSurfaceView.update();
                        Toast.makeText(mContext, "Successfully loaded noteFile.",
Toast.LENGTH_SHORT).show();
                    } catch (IOException e) {
                        Toast.makeText(mContext, "Cannot open this file.",
Toast.LENGTH_LONG).show();
                    } catch (SpenUnsupportedTypeException e) {
                        Toast.makeText(mContext, "This file is not supported.",
Toast.LENGTH_LONG).show();
                    } catch (SpenInvalidPasswordException e) {
                        Toast.makeText(mContext, "This file is locked by a password.",
Toast.LENGTH_LONG).show();
                    } catch (SpenUnsupportedVersionException e) {
                        Toast.makeText(mContext, "This file is the version that does
not support.",
                                Toast.LENGTH_LONG).show();
                    } catch (Exception e) {
                        Toast.makeText(mContext, "Failed to load noteDoc.",
```

```
Toast.LENGTH_LONG).show();
                    }
                }
            }).show();
}
private String[] setFileList() {
    // Call the file list under the directory in mFilePath.
    if (!mFilePath.exists()) {
        if (!mFilePath.mkdirs()) {
            Toast.makeText(mContext, "Save Path Creation Error",
Toast.LENGTH_SHORT).show();
            return null;
        }
    }
    // Filter in spd and png files.
    File[] fileList = mFilePath.listFiles(new txtFileFilter());
    if (fileList == null) {
        Toast.makeText(mContext, "File does not exist.", Toast.LENGTH_SHORT).show();
        return null;
    }

    int i = 0;
    String[] strFileList = new String[fileList.length];
    for (File file : fileList) {
        strFileList[i++] = file.getName();
    }

    return strFileList;
}

class txtFileFilter implements FilenameFilter {
@Override
public boolean accept(File dir, String name) {
return (name.endsWith(".spd") || name.endsWith(".png"));
        }
}

              .........
```

For more information, see PenSample2_5_LoadFile in PenSample2_5_LoadFile.java

The following sections provide more details on the steps involved in loading SPD and +SPD (image file with added SPD data) files.

4.2.3.1 Adding a Listener for the Load File Button

To handle Load File button events:

1. Create a Load File button.
2. Create an`OnClickListener`listener instance for the Load File button, mLoadFileBtnClickListener in the sample, and register it by calling `setOnClickListener()` on the button.

In the `onClick()` method, close all the open settingsview and call the file selection view.

```
closeSettingView();
loadNoteFile();
```

In the file selection view, create a window to display a list of theSPD and PNG files in the “SPen/” folder in external storage to allow users to select a file.

```
try {
// Create NoteDoc with the selected file.
          SpenNoteDoc tmpSpenNoteDoc = new SpenNoteDoc(mContext,
                    strFilePath, mScreenRect.width(), SpenNoteDoc.MODE_WRITABLE, true);
mSpenNoteDoc.close();
mSpenNoteDoc = tmpSpenNoteDoc;
if (mSpenNoteDoc.getPageCount() == 0) {
mSpenPageDoc = mSpenNoteDoc.appendPage();
          } else {
mSpenPageDoc = mSpenNoteDoc.getPage(
mSpenNoteDoc.getLastEditedPageIndex());
          }
mSpenSimpleSurfaceView.setPageDoc(mSpenPageDoc, true);
mSpenSimpleSurfaceView.update();
} catch (IOException e) {
          Toast.makeText(mContext,"Cannot open this file.",
              Toast.LENGTH_LONG).show();
} catch (SpenUnsupportedTypeException e) {
          Toast.makeText(mContext,"This file is not supported.",
              Toast.LENGTH_LONG).show();
} catch (SpenInvalidPasswordException e) {
          Toast.makeText(mContext,"This file is locked by a password.",
              Toast.LENGTH_LONG).show();
} catch (SpenUnsupportedVersionException e) {
          Toast.makeText(mContext, "This file is a version that is not supported.",
              Toast.LENGTH_LONG).show();
} catch (Exception e) {
          Toast.makeText(mContext, "Failed to load noteDoc.",
              Toast.LENGTH_LONG).show();
}
```

When a new SpenNoteDoc instance is successfully created with the selected file, call `close()`to close the old SpenNoteDoc. Specify the new SpenNoteDoc instance as a member variable of mSpenNoteDoc.

If the new SpenNoteDoc instance does not have a page, call `SpenNoteDoc.appendPage()` to create a new SpenPageDoc instance; otherwise, use the value returned by `getLastEditedPageIndex()` to call `SpenNoteDoc.getPage()`for getting the last edited page.

Call `SpenSimpleSurfaceView.setPageDoc()` to link the page information and your SpenSimpleSurfaceView instance.

Call `SpenSimpleSurfaceView.update()`to refresh the screen with the loaded file data.

- SpenUnsupportedTypeException is thrown if Pen SDK Light cannot read the format of the selected file.

- SpenInvalidPasswordException is thrown if an invalid password is entered for an encrypted file.
- SpenUnsupportedVersionException is thrown if Pen SDK Light does not support the SPD file format version.

4.2.4. Adding Pages

You can use Pen SDK Light to create an application that can add multiple pages to a note.

You can use `SpenNoteDoc.insertPage()` to insert new pages at a specified indexand`SpenNoteDoc.appendPage()` to append a new page as the last page of the note.

The sample application implements the following features:

- Add Page button for adding pages.
- Listener for the Add Page button.
- When the Add Page button is clicked,the`onClick()` callback method for the Add Page buttoncalls`SpenNoteDoc.appendPage()` to add a page.
- Listener for flick events in the View area.
- When a Flick event occurs, the sample application calls`SpenNoteDoc.getPage()` to get either the previous or the next page, depending on the flick direction.

Figure 15: Page Add function

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_attach_file);
    mContext = this;
              .........

    mSpenSimpleSurfaceView.setFlickListener(mFlickListener);
```

```
            .........

    mAddPageBtn = (ImageView) findViewById(R.id.addPageBtn);
    mAddPageBtn.setOnClickListener(mAddPageBtnClickListener);



            .........

}

            .........

private final OnClickListener mAddPageBtnClickListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        mSpenSimpleSurfaceView.closeControl();

        closeSettingView();
        // Create a page next to the current page.
        int pageIndex = mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId()) + 1;
        mSpenPageDoc = mSpenNoteDoc.insertPage(pageIndex);
        mSpenPageDoc.setBackgroundColor(0xFFD6E6F5);
        mSpenPageDoc.clearHistory();

        v.setClickable(false);
        mSpenSimpleSurfaceView.setPageDoc(mSpenPageDoc, true);
        v.setClickable(true);

        updatePageButton(pageIndex);
    }
};
            .........

private final OnClickListener mNextPageBtnClickListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        int index = mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId()) + 1;
        mSpenPageDoc = mSpenNoteDoc.getPage(index);
        mSpenSimpleSurfaceView.setPageDoc(mSpenPageDoc, true);
        updatePageButton(index);
    }
};

private final OnClickListener mPrevPageBtnClickListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        int index = mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId()) - 1;
        mSpenPageDoc = mSpenNoteDoc.getPage(index);
        mSpenSimpleSurfaceView.setPageDoc(mSpenPageDoc, true);
        updatePageButton(index);
    }
};

    private void updatePageButton(int pageIndex){
        mPrevPageBtn.setEnabled(true);
        mNextPageBtn.setEnabled(true);
```

```
        if(pageIndex == 0)
        {
            mPrevPageBtn.setEnabled(false);
        }

        if(pageIndex == mSpenNoteDoc.getPageCount() - 1)
        {
            mNextPageBtn.setEnabled(false);
        }

        mPageIndexText.setText(pageIndex + 1 + "/" + mSpenNoteDoc.getPageCount());
    }               .........
```

For more information, see PenSample2_7_AddPage.java in PenSample2_7_AddPage.

The following sections provide more details on the steps involved in adding a page.

4.2.4.1 Registering a Listener for the Add Page Button

To handle Add Page button events in your application:

1. Create an Add Page button.
2. Create an `OnClickListener` listener instance for the Add Page button, mAddPageBtnClickListener in the sample,and register it by calling `setOnClickListener()` on the button.

In the `onClick()` method:

- Close any open settingsviews.
- Call `SpenNoteDoc.insertPage()`to add a new page after the current page and get the instance returned for the new page.
- Use `SpenPageDoc.getId()` and `SpenPageDoc.getPageIndexById()`to get the index of the current page.
- Pass this to `SpenSimpleSurfaceView.setPageDoc()` to set the new page in your SpenSimpleSurfaceView instance, and print a text that shows the index of the current page in the View area.
- If the user taps or clicks the Add New Page button multiple times quickly, a new page might be added before the page effect of the previous `SpenSimpleSurfaceView.setPageDoc()` is completed. This can cause problems in your application. To prevent this:
    - Disable the Add New Page button before calling `SpenSimpleSurfaceView.setPageDoc()`.
    - Register an SpenPageEffectListener instance.
    - In the `onFinish()` callback method, which is called on completion of a page effect, enable the button.
-

```
private final OnClickListener mAddPageBtnClickListener = new OnClickListener() {
    @Override
```

```
    public void onClick(View v) {
        mSpenSimpleSurfaceView.closeControl();

        closeSettingView();
        // Create a page next to the current page.
        int pageIndex = mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId()) + 1;
        mSpenPageDoc = mSpenNoteDoc.insertPage(pageIndex);
        mSpenPageDoc.setBackgroundColor(0xFFD6E6F5);
        mSpenPageDoc.clearHistory();

        v.setClickable(false);
        mSpenSimpleSurfaceView.setPageDoc(mSpenPageDoc, true);
        v.setClickable(true);

        updatePageButton(pageIndex);
    }
};
```

4.2.4.2 Registering a Listener for Next and Previous Button

To handle Next and Previous events in the View area in your application:

- Add Next and Previous button for next and previous pages.
- Listener for the these buttons.
- When the Next and Previous button is clicked,the `onClick()` callback method for the Add Page button calls `mSpenSimpleSurfaceView.setPageDoc()` to next and previous.

When the number of pages in the note is greater than 1 and a next or previous event occurs, call `SpenNoteDoc.getPage()` to get either the previous or next page depending on next or previous button.

Call `SpenSimpleSurfaceView.setPageDoc()` to set the page as the current page of the SpenSimpleSurfaceView instance.

Display a text that shows the index of the current page in the View area.

```
private void updatePageButton(int pageIndex){
    mPrevPageBtn.setEnabled(true);
    mNextPageBtn.setEnabled(true);

    if(pageIndex == 0)
    {
        mPrevPageBtn.setEnabled(false);
    }

    if(pageIndex == mSpenNoteDoc.getPageCount() - 1)
    {
        mNextPageBtn.setEnabled(false);
    }

    mPageIndexText.setText(pageIndex + 1 + "/" + mSpenNoteDoc.getPageCount());
}
private final OnClickListener mNextPageBtnClickListener = new OnClickListener() {
    @Override
```

```
    public void onClick(View v) {
        int index = mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId()) + 1;
        mSpenPageDoc = mSpenNoteDoc.getPage(index);
        mSpenSimpleSurfaceView.setPageDoc(mSpenPageDoc, true);
        updatePageButton(index);
    }
};

private final OnClickListener mPrevPageBtnClickListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        int index = mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId()) - 1;
        mSpenPageDoc = mSpenNoteDoc.getPage(index);
        mSpenSimpleSurfaceView.setPageDoc(mSpenPageDoc, true);
        updatePageButton(index);
    }
};
```

4.2.5. Using Extra Data

Pen SDK Light provides methods to save any additional data required by your applications. As shown in the following sample code, Pen SDK Light links a user-defined key value to the data, which enables Pen SDK to load the data corresponding to the key.

```
note.setExtraDataString("STRING_KEY", "String Data");

              .........

if (note.hasExtraDataString("STRING_KEY")) {
        String str = note.getExtraDataString("STRING_KEY");
}
```

You can use the methods listed in the following table in the SpenNoteDoc, SpenPageDoc, and SpenObjectBase classes to save extra data in a note, page, or object. Pen SDK Light does not record changes made to set up extra data in the historystack. You cannot restore extra data with the Undo command.

Method	Description
setExtraDataString setExtraDataInt setExtraDataStringArray setExtraDataByteArray	Sets extra data for the specified key.
getExtraDataString getExtraDataInt	Returns extra data that corresponds to the specified key.

Method	Description
`getExtraDataStringArray` `getExtraDataByteArray`	

`hasExtraDataString`
`hasExtraDataInt`
`hasExtraDataStringArray`
`hasExtraDataByteArray`
	Checks whether there is extra data that corresponds to the specified key.

`removeExtraDataString`
`removeExtraDataInt`
`removeExtraDataStringArray`
`removeExtraDataByteArray`
	Removes extra data thatcorresponds to the specified key.

4.3. Selecting Objects

Pen SDK Light allows you to resize, relocate, or rotate objects added to the SpenSimpleSurfaceView instance. You can use `SpenPageDoc.moveObjectIndex()` to edit the order of objects in the SpenPageDoc instance.

4.3.1. Using the Rectangle and Lasso Selection Tool

You can use Pen SDK Light to create a tool for selecting an object on the SpenSimpleSurfaceView instance.

Pen SDK Light offers SpenSettingSelectionLayout, which enables you to set up the following two types of object selections:

- Lasso selection, which allows you to draw a selection border to select the object enclosed in the shape you draw.
- Rectangle selection,which allows you to draw a rectangle to select the object enclosed by the rectangle.

The sample application implements the following features:

- Adds SpenSettingSelectionLayout to the sample application created in the Selecting Top Objects section.
- When the Selection Tool button is clicked, the `onClick()` callback method displays the SpenSettingSelectionLayout viewtolet users specify the selection type: Lasso or Rectangle.
- One or multiple object selection.

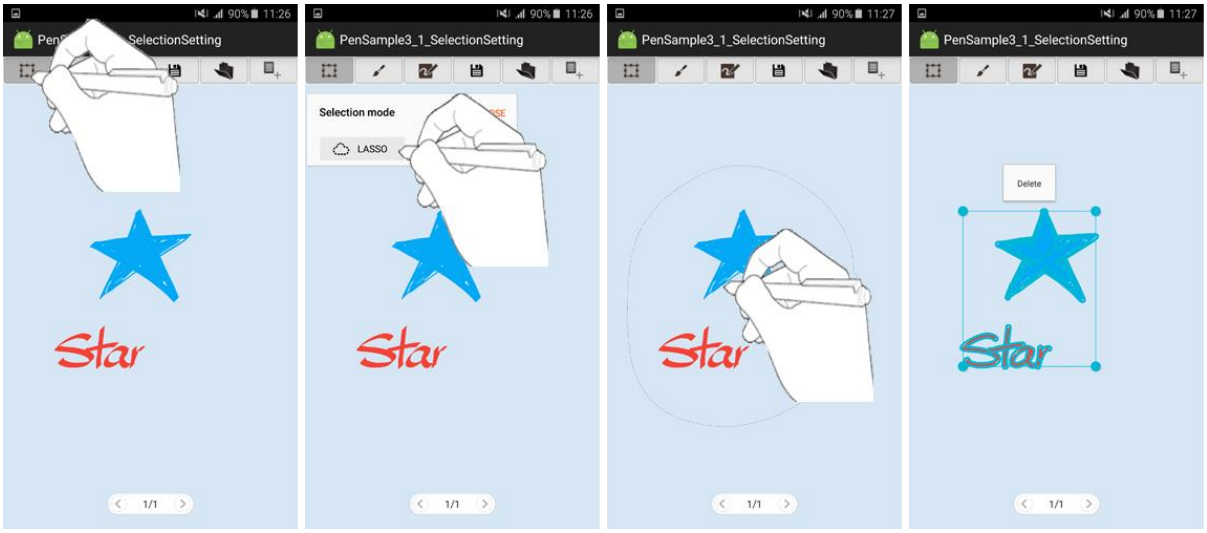


Figure 16: Selection settings

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_selection_setting);
    mContext = this;
              .........

// Create SelectionSettingView
mSelectionSettingView = new SpenSettingSelectionLayout(mContext, "", spenViewLayout);
if (mSelectionSettingView == null) {
    Toast.makeText(mContext, "Cannot create new SelectionSettingView.",
Toast.LENGTH_SHORT).show();
    finish();
}
mSettingView = (FrameLayout) findViewById(R.id.settingView);
mSettingView.addView(mPenSettingView);
mSettingView.addView(mSelectionSettingView);

// Create SpenSimpleSurfaceView
mSpenSimpleSurfaceView = new SpenSimpleSurfaceView(mContext);
if (mSpenSimpleSurfaceView == null) {
    Toast.makeText(mContext, "Cannot create new SpenSimpleSurfaceView.",
Toast.LENGTH_SHORT).show();
    finish();
}
mSpenSimpleSurfaceView.setToolTipEnabled(true);
spenViewLayout.addView(mSpenSimpleSurfaceView);
mPenSettingView.setCanvasView(mSpenSimpleSurfaceView);
mTextSettingView.setCanvasView(mSpenSimpleSurfaceView);
mSelectionSettingView.setCanvasView(mSpenSimpleSurfaceView);
              .........

mSpenSimpleSurfaceView.setSelectionChangeListener(mSelectionListener);

// Set the button.
mSelectionBtn = (ImageView) findViewById(R.id.selectionBtn);
mSelectionBtn.setOnClickListener(mSelectionBtnClickListener);
```

```
            .........

}

            .........

private final OnClickListener mSelectionBtnClickListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        mSpenSimpleSurfaceView.closeControl();

        // When Spen is in selection mode
        if (mSpenSimpleSurfaceView.getToolTypeAction(mToolType) ==
SpenSimpleSurfaceView.ACTION_SELECTION) {
            // If SelectionSettingView is open, close it.
            if (mSelectionSettingView.isShown()) {
                mSelectionSettingView.setVisibility(View.GONE);
                // If SelectionSettingView is not open, open it.
            } else {
                mSelectionSettingView.setVisibility(View.VISIBLE);
            }
            // If Spen is not in selection mode, change it to selection mode.
        } else {
            mMode = MODE_SELECTION;
            selectButton(mSelectionBtn);
            mSpenSimpleSurfaceView.setToolTypeAction(mToolType,
SpenSimpleSurfaceView.ACTION_SELECTION);
        }
    }
};
            .........

private final SpenSelectionChangeListener mSelectionListener = new
SpenSelectionChangeListener() {

    @Override
    public void onChanged(SpenSettingSelectionInfo info) {
        // Close Setting view if selection type is changed.
        mSelectionSettingView.setVisibility(SpenSimpleSurfaceView.GONE);
    }
};

private void selectButton(View v) {
    // Enable or disable the button according to the current mode.
    mSelectionBtn.setSelected(false);
    mPenBtn.setSelected(false);
    mStrokeObjBtn.setSelected(false);

    v.setSelected(true);
    closeSettingView();
}
private void closeSettingView() {
    // Close all the setting views.
    mPenSettingView.setVisibility(SpenSimpleSurfaceView.GONE);
    mSelectionSettingView.setVisibility(SpenSimpleSurfaceView.GONE);
}
protected void onDestroy() {
    super.onDestroy();
```

```
    if (mPenSettingView != null) {
        mPenSettingView.close();
    }

    if (mSelectionSettingView != null) {
        mSelectionSettingView.close();
    }

    if (mSpenSimpleSurfaceView != null) {
        mSpenSimpleSurfaceView.closeControl();
        mSpenSimpleSurfaceView.close();
        mSpenSimpleSurfaceView = null;
    }

    if (mSpenNoteDoc != null) {
        try {
            if (mIsDiscard) {
                mSpenNoteDoc.discard();
            } else {
                mSpenNoteDoc.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        mSpenNoteDoc = null;
    }
}
```

For more information, see PenSample3_1_SelectionSetting.java in PenSample3_1_SelectionSetting.

The following sections provide more details on the steps involved in using the Rectangle and Lasso selection tools.

4.3.1.1 Creating SpenSettingSelectionLayout

To add SpenSettingSelectionLayout to your application:

1. Create an instance of `SpenSettingSlectionLayout`, mSelectionSettingView in the sample.

In the `onClick()` method,handle the selection of the Selection Tool button:

- To stack the SpenSettingSelectionLayout view on your SpenSimpleSurfaceView instance in the viewport, call `addView()` and add your SpenSettingSelectionLayout instance to the SpenSimpleSurfaceView container defined in FrameLayout.
- Pass the SpenSimpleSurfaceView instance when calling`SpenSettingSelectionLayout.setCanvasView()` to link the selection tool functionality to SpenSimpleSurfaceView.

```
mSelectionSettingView =
new SpenSettingSelectionLayout(mContext, new String(),
                spenViewLayout);
if (mSelectionSettingView == null) {
       finish();
```

```
}
spenViewContainer.addView(mSelectionSettingView);

                .........

mSelectionSettingView.setCanvasView(mSpenSimpleSurfaceView);
```

4.3.1.2 Registering a Listener for the Selection Tool Button

To handle Selection Tool button events:

1. Create a Selection Tool button.
2. Create an `OnClickListener` listener instance for the Selection Tool button and register it by calling `setOnClickListener()`onthe button.

In the `onClick()` method, if `mToolType` is set to ACTION_SELECTION and the Selection Tool button is clicked again., add the following:

- Close the SpenSettingSelectionLayout view if it is open.
- If the SpenSettingSelectionLayout view is not open, display it
- In the view, let the user select a selection tool: Lasso or Rectangle.

```
if (mSpenSimpleSurfaceView.getToolTypeAction(mToolType) ==
SpenSimpleSurfaceView.ACTION_SELECTION) {
    // If SelectionSettingView is open, close it.
    if (mSelectionSettingView.isShown()) {
        mSelectionSettingView.setVisibility(View.GONE);
        // If SelectionSettingView is not open, open it.
    } else {
        mSelectionSettingView.setVisibility(View.VISIBLE);
    }
```

4.3.1.3 Creating and Registering a Selection Change Event Listener

To handle selection change events:

1. Create an `SpenSelectionChangeListener` listener instance to handle selection change events.
2. Add the `onChanged()` callback method, which is called when selection settings change
3. Call `SpenSimpleSurfaceView.setSelectionChangeListener()` to register the listener.

In the `onChanged()` method, close the SpenSettingSelectionLayout window.

```
public void onChanged(SpenSettingSelectionInfo info) {
    // Close Setting view if selection type is changed.
        mSelectionSettingView.setVisibility(SpenSimpleSurfaceView.GONE);
```

```
    }
};
```

4.3.1.4 Preventing Memory Leaks

To prevent memory leaks:

1. Call `onDestroy()` to close the SpenSettingSelectionLayout instance.

```
if (mSelectionSettingView != null) {
mSelectionSettingView.close();
}
```

4.3.2. Bringing Objects Forward and Backward

You can use Pen SDK Light to change the placement of objects in your application by bringing them forward and backward.

The sample application implements the following features:

- Adds the following menu items to the control selected objects:
    - “Move to bottom” to send the selected object to the bottom of a group of stacked objects.
    - "Move backward" to send the selected object back one level.
    - "Move forward" to bring the selected object forward one level.
    - "Move to top" to bring the selected object to the top of a group of stacked objects.

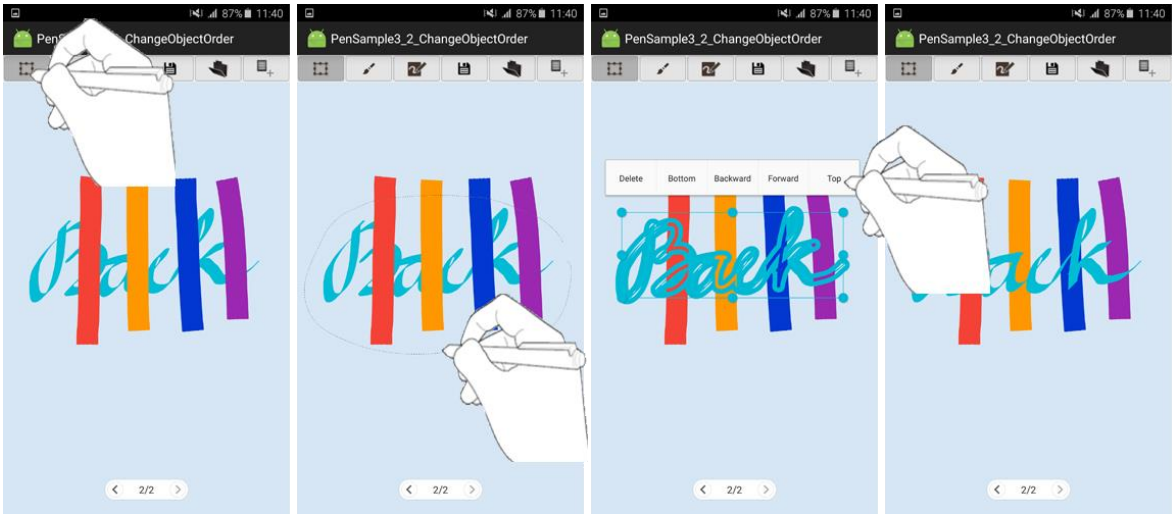


Figure 17: Moving an object

```
public class PenSample3_2_ChangeObjectOrder extends Activity {
    private final int CONTEXT_MENU_PROPERTIES_ID = 0;
```

```
    private final int CONTEXT_MENU_DELETE_ID = 10;
    private final int CONTEXT_MENU_GROUP_ID = 20;
    private final int CONTEXT_MENU_UNGROUP_ID = 21;
    private final int CONTEXT_MENU_MOVE_TO_BOTTOM_ID = 30;
    private final int CONTEXT_MENU_MOVE_TO_BACKWARD_ID = 31;
    private final int CONTEXT_MENU_MOVE_TO_FORWARD_ID = 32;
    private final int CONTEXT_MENU_MOVE_TO_TOP_ID = 33;
                .........

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_change_object_order);
    mContext = this;
                .........

mSpenSimpleSurfaceView.setControlListener(mControlListener);
mSpenSimpleSurfaceView.setSelectionChangeListener(mSelectionListener);
// Set the button.
mSelectionBtn = (ImageView) findViewById(R.id.selectionBtn);
mSelectionBtn.setOnClickListener(mSelectionBtnClickListener);
                .........

addImgObject(mScreenRect.width() / 2, mScreenRect.height() / 2, 3);
addTextObject(mScreenRect.width() / 2, mScreenRect.height() / 2,
"test").setFontSize(100);
addStrokeObject(mScreenRect.width() / 2, mScreenRect.height() / 2);
addShapeObject(mScreenRect.width() / 2, mScreenRect.width() / 2,
SpenObjectShape.TYPE_HEART, "LIKE");
addLineObject(mScreenRect.width()      /      2,      mScreenRect.width()      /      2,
SpenObjectLine.TYPE_STRAIGHT);
                .........

    }

                .........

private final SpenControlListener mControlListener = new SpenControlListener() {

    @Override
    public void onRotationChanged(float arg0, SpenObjectBase arg1) {
    }

    @Override
    public void onRectChanged(RectF arg0, SpenObjectBase arg1) {
    }

    @Override
    public void onObjectChanged(ArrayList<SpenObjectBase> arg0) {
    }

    @Override
    public boolean onMenuSelected(ArrayList<SpenObjectBase> objectList, int itemId) {
        SpenObjectContainer objContainer;
        SpenObjectBase object = objectList.get(0);
        switch (itemId) {
        // Properties of object shape/line
        case CONTEXT_MENU_PROPERTIES_ID:
            shapeProperties();
            mSpenSimpleSurfaceView.closeControl();
```

```
            break;

        // Remove the selected object.
        case CONTEXT_MENU_DELETE_ID:
            // mSpenPageDoc.removeSelectedObject();
            for (SpenObjectBase obj : objectList) {
                mSpenPageDoc.removeObject(obj);
            }
            mSpenSimpleSurfaceView.closeControl();
            mSpenSimpleSurfaceView.update();
            break;

            // Send the selected object to the back.
        case CONTEXT_MENU_MOVE_TO_BOTTOM_ID:
            mSpenPageDoc.moveObjectIndex(object, -mSpenPageDoc.getObjectIndex(object),
true);
            mSpenSimpleSurfaceView.update();
            break;

        // Send the selected object backward by an index.
        case CONTEXT_MENU_MOVE_TO_BACKWARD_ID:
            if (mSpenPageDoc.getObjectIndex(object) > 0) {
                mSpenPageDoc.moveObjectIndex(object, -1, true);
                mSpenSimpleSurfaceView.update();
            }
            break;

        // Bring the selected object forward by an index.
        case CONTEXT_MENU_MOVE_TO_FORWARD_ID:
            if (mSpenPageDoc.getObjectIndex(object) < mSpenPageDoc.getObjectCount(true)
- 1) {
                mSpenPageDoc.moveObjectIndex(object, 1, true);
                mSpenSimpleSurfaceView.update();
            }
            break;

        // Bring the selected object to the front.
        case CONTEXT_MENU_MOVE_TO_TOP_ID:
            mSpenPageDoc.moveObjectIndex(object,
                    mSpenPageDoc.getObjectCount(true) - 1 -
mSpenPageDoc.getObjectIndex(object), true);
            mSpenSimpleSurfaceView.update();
            break;
        default:
            break;
        }

        return true;
    }

    @Override
    public boolean onCreated(ArrayList<SpenObjectBase> objectList, ArrayList<Rect>
relativeRectList,
            ArrayList<SpenContextMenuItemInfo> menu, ArrayList<Integer> styleList, int
pressType, PointF point) {

        // Set the Context menu
        if (objectList.size() == 1
                && (objectList.get(0).getType() == SpenObjectBase.TYPE_SHAPE ||
```

```
objectList.get(0).getType() == SpenObjectBase.TYPE_LINE)) {
            menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_PROPERTIES_ID,
"Properties", true));
        }
        menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_DELETE_ID, "Delete", true));

        // Display Arrange menu if only one object is selected.
        if (objectList.size() == 1) {
            menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_MOVE_TO_BOTTOM_ID,
"Bottom", true));
            menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_MOVE_TO_BACKWARD_ID,
"Backward", true));
            menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_MOVE_TO_FORWARD_ID,
"Forward", true));
            menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_MOVE_TO_TOP_ID, "Top",
true));
            return true;
        }
        // Attach an individual control for each object.
        SpenControlList controlList = new SpenControlList(mContext, mSpenPageDoc);
        controlList.setObject(objectList);
        controlList.setGroup(false);
        mSpenSimpleSurfaceView.setControl(controlList);
        controlList.setContextMenu(menu);

        return false;
    }

    @Override
    public boolean onClosed(ArrayList<SpenObjectBase> arg0) {
        return false;
    }
   };
          .....
```

For more information, see PenSample3_2_MoveObject.java in PenSample3_2_MoveObject.

The following sections provide more details on the steps involved in moving objects forward and backward.

4.3.2.1 Creating a Context Menu in a Control

To create a context menu for control eventsin your application:

1. Create an`SpenControlListener` listener instance.
2. Call `SpenSimpleSurfaceView.setControlListener()`to register the listener.

In the `onCreated()` callback method,which is called when there is a control in the View area, create a context menu:

- Create an SpenContextMenuItemInfo instance to add the menu items that appear in the context menu when users select a single object:
    - “Bottom” to move the selected object to the bottom

- "Backward" to move the object one level back
- "Forward" to move the object one level forward
- "Top" to move the object to the top.

- Create SpenContextMenuItemInfo to register these commands in the context menu.

```
public boolean onCreated(ArrayList<SpenObjectBase> objectList, ArrayList<Rect>
relativeRectList, ArrayList<SpenContextMenuItemInfo> menu, ArrayList<Integer>
styleList, int pressType, PointF point) {
            .........

// Display Arrange menu if only one object is selected.
if (objectList.size() == 1) {
    menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_MOVE_TO_BOTTOM_ID, "Bottom",
true));
    menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_MOVE_TO_BACKWARD_ID, "Backward",
true));
    menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_MOVE_TO_FORWARD_ID, "Forward",
true));
    menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_MOVE_TO_TOP_ID, "Top", true));
    return true;
}
            .........
```

4.3.2.2 Handling Context Menu Events in a Control

To handle context menu events in your application:

1. In the `onMenuSelected()` callback method, which is called when a menu item is selected from the context menu on a control, execute the menu items using their menu IDs.

When the “Bottom” menu item is selected from the context menu, do the following:

- Calculate the step that makes the index of the selected object zero. This is the return value of `SpenPageDoc.getObjectIndex(object)` with a leading minus sign.
- Call `SpenPageDoc.moveObjectIndex()`and pass the step that was calculated as the second parameter. If the step is a negative integer, change the index to the start of the object list. If the step is a positive integer, change the index to the end of the object list.
    - Call `SpenPageDoc.getObjectIndex()`to get the current index of the selected object.
    - Call `SpenPageDoc.getObjectCount()`to get the number of objects in SpenPageDoc.
- Call `SpenPageDoc.moveObjectIndex()`using the calculated step to move the object to the bottom in SpenPageDoc.
- Call `SpenSimpleSurfaceView.update()` to refresh the screen.

```
public boolean onMenuSelected(ArrayList<SpenObjectBase> objectList, int itemId) {
    SpenObjectContainer objContainer;
    SpenObjectBase object = objectList.get(0);
    switch (itemId) {
```

```
            .........

// Send the selected object to the back.
case CONTEXT_MENU_MOVE_TO_BOTTOM_ID:
    mSpenPageDoc.moveObjectIndex(object, -mSpenPageDoc.getObjectIndex(object), true);
    mSpenSimpleSurfaceView.update();
        break;
```

When the "Backward" menu item is selected, do the following::

- Set the step to-1 if the object is not located at the bottom of the stack.Call `SpenPageDoc.moveObjectIndex()` to send the object back one level in SpenPageDoc.
- Call `SpenSimpleSurfaceView.update()` to refresh the screen.

```
case CONTEXT_MENU_MOVE_TO_BACKWARD_ID:
    if (mSpenPageDoc.getObjectIndex(object) > 0) {
        mSpenPageDoc.moveObjectIndex(object, -1, true);
        mSpenSimpleSurfaceView.update();
    }
    break;
```

When the "Forward" menu item is selected, do the following::

- Set the step to 1 if the object is not located at the top of the stack. Call `SpenPageDoc.moveObjectIndex()` to bring the object forward one level in SpenPageDoc.
- Call `SpenSimpleSurfaceView.update()` to refresh the screen.

```
// Bring the selected object forward one index.
caseCONTEXT_MENU_MOVE_TO_FORWARD_ID:
if (mSpenPageDoc.getObjectIndex(object) <
mSpenPageDoc.getObjectCount(true) - 1) {
mSpenPageDoc.moveObjectIndex(object, 1, true);
mSpenSimpleSurfaceView.update();
      }
    break;
```

When the “Top” menu item is selected from the context menu, do the following:

- Calculate the step that makes the index of the selected object -1 subtracted from the count of all objects.Call `SpenPageDoc.moveObjectIndex()`to bring the object to the top in SpenPageDoc.
- Call `SpenSimpleSurfaceView.update()` to refresh the screen.

```
// Bring the selected object to the front.
case CONTEXT_MENU_MOVE_TO_TOP_ID:
    mSpenPageDoc.moveObjectIndex(object,
            mSpenPageDoc.getObjectCount(true) - 1 -
mSpenPageDoc.getObjectIndex(object), true);
    mSpenSimpleSurfaceView.update();
        break;
```

4.4. Using Advanced Pen SDK Light Features

Pen SDK Light also offers you the following advanced features for your applications:

- Smart Scroll
- Smart Zoom
- Translucent View

4.4.1. Displaying Translucent Pen SDK Light Views

You can use Pen SDK Light to provide a Simple View in your application. Simple View creates memos with SpenObjectStroke by inserting an SpenSimpleView instance over the main SpenSimpleSurfaceView instance.

The sample application implements the following features:

- Simple View button for creating anSpenSimpleView instance with a transparent background.
- Stroke creation andsavingin an image file with a specified name.

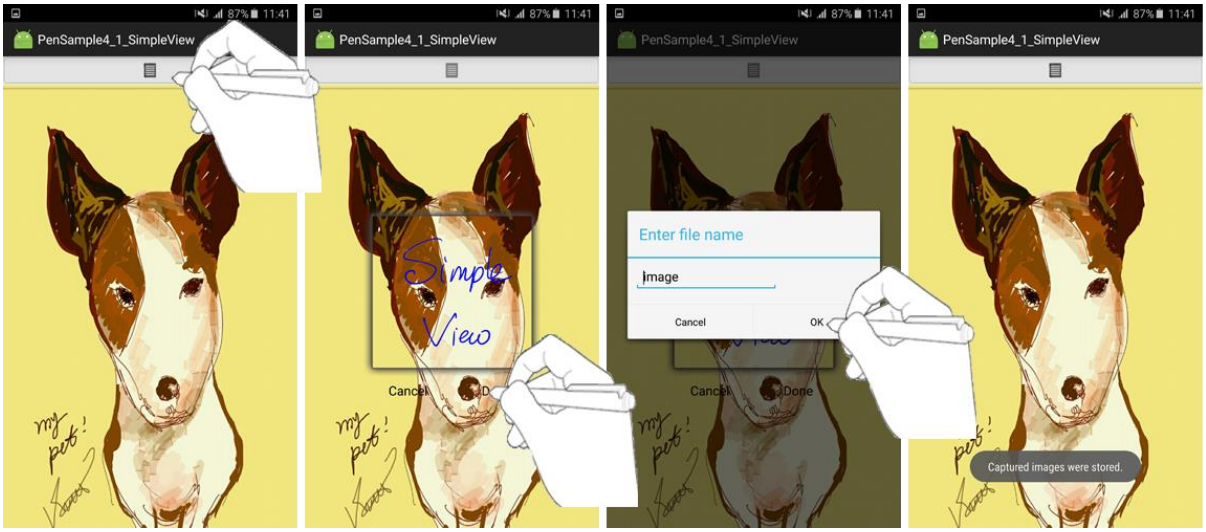


Figure 18: Simple View

```
public class PenSample4_1_SimpleView extends Activity {

   private Context mContext;
   private SpenNoteDoc mSpenNoteDoc;
   private SpenPageDoc mSpenPageDoc;
   private SpenSimpleSurfaceView mSpenSimpleSurfaceView;
   private SpenSimpleView mSpenSimpleView;
   private RelativeLayout mSpenSimpleViewContainer;

   private ImageView mSmartScrollBtn;
   private ImageView mSmartZoomBtn;
   private ImageView mSimpleViewBtn;

   private AlertDialog dlgSave;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_simple_view);
mContext = this;

              .........

// Set the background.
        String path = mContext.getFilesDir().getPath();
        Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
                R.drawable.smemo_bg);
saveBitmapToFileCache(bitmap, path + "/smemo_bg.jpg");
mSpenPageDoc.setBackgroundImageMode(
SpenPageDoc.BACKGROUND_IMAGE_MODE_STRETCH);
mSpenPageDoc.setBackgroundImage(path + "/smemo_bg.jpg");
mSpenPageDoc.clearHistory();

              .........

mSimpleViewBtn = (ImageView) findViewById(R.id.simpleViewBtn);
mSimpleViewBtn.setOnClickListener(mSimpleViewBtnClickListener);
    }

static public void saveBitmapToFileCache(Bitmap bitmap, String strFilePath) {
// Save the resource in a file to set this as a background image.
        File file = new File(strFilePath);
        OutputStream out = null;

if (file.exists() == true) {
return;
        }
try {
            file.createNewFile();
            out = new FileOutputStream(file);

if (strFilePath.endsWith(".jpg")) {
                bitmap.compress(CompressFormat.JPEG, 100, out);
            } else {
                bitmap.compress(CompressFormat.PNG, 100, out);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
try {
if(out!= null) {
                    out.close();
}
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

              .........

private final OnClickListener mSimpleViewBtnClickListener =
new OnClickListener() {
```

```
@Override
public void onClick(View v) {
// Disable Simple View button to avoid any repeated action.
mSimpleViewBtn.setEnabled(false);

mSpenSimpleViewContainer =
                    (RelativeLayout) findViewById(R.id.spenSimpleViewContainer);

mSpenSimpleViewContainer.setVisibility(View.VISIBLE);

                RelativeLayout spenSimpleViewLayout =
                    (RelativeLayout) findViewById(R.id.spenSimpleViewLayout);

                FrameLayout.LayoutParams simpleViewContainerParams =
                    (FrameLayout.LayoutParams)
mSpenSimpleViewContainer.getLayoutParams();
                FrameLayout.LayoutParams simpleViewLayoutParams =
                    (FrameLayout.LayoutParams)
spenSimpleViewLayout.getLayoutParams();

// Get the dimensions of the screen of the device.
                Display display = getWindowManager().getDefaultDisplay();
                Rect rect = new Rect();
                display.getRectSize(rect);
int btnHeight = 100;
// Resize SimpleView to the width of the screen at a random ratio.
if (rect.width() > rect.height()) {
                    simpleViewContainerParams.width = (int) (rect.height() * .6);
                    simpleViewContainerParams.height =
                      (int) (rect.height() * .6) + btnHeight;
                } else {
                    simpleViewContainerParams.width = (int) (rect.width() * .6);
                    simpleViewContainerParams.height =
                      (int) (rect.width() * .6) + btnHeight;
                }
                simpleViewLayoutParams.width =
                      (int) (simpleViewContainerParams.width * .9);
                simpleViewLayoutParams.height =
                      (int) ((simpleViewContainerParams.height)
                        - (simpleViewContainerParams.width * .1) - btnHeight);
mSpenSimpleViewContainer.setLayoutParams(simpleViewContainerParams);
                spenSimpleViewLayout.setLayoutParams(simpleViewLayoutParams);

int screenWidth = simpleViewLayoutParams.width;
int screenHeight = simpleViewLayoutParams.height;
// Create SimpleView.
mSpenSimpleView = new SpenSimpleView(mContext, screenWidth,
screenHeight);
                spenSimpleViewLayout.addView(mSpenSimpleView);

                initSimpleViewPenSettingInfo();

// Define the button.
                Button doneBtn = (Button) findViewById(R.id.done_btn);
                doneBtn.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
                     if(mSpenSimpleView != null) {
                     inputFileName();
```

```
                    }
                }
            });

            Button cancelBtn = (Button) findViewById(R.id.cancel_btn);
            cancelBtn.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
                    if(dlgSave != null && dlgSave.isShowing()) {
                        return;
                    }
                    closeSimpleView();
return;
                }
            });
        }
    };

private void initSimpleViewPenSettingInfo() {
// Initialize settings for the pen for use in Simple View.
        SpenSettingPenInfo penInfo = new SpenSettingPenInfo();
        penInfo.color = Color.BLUE;
        penInfo.size = 10;
mSpenSimpleView.setPenSettingInfo(penInfo);
    }

private void inputFileName() {
// Display the File Save dialog to prompt users to enter file names.
        LayoutInflater inflater =
            (LayoutInflater) mContext
                .getSystemService(LAYOUT_INFLATER_SERVICE);
final View layout =
            inflater.inflate(R.layout.save_image_dialog,
                (ViewGroup) findViewById(R.id.layout_root));

        AlertDialog.Builder builderSave =
new AlertDialog.Builder(mContext);
        builderSave.setTitle("Enter file name");
        builderSave.setView(layout);

final EditText inputPath =
            (EditText) layout.findViewById(R.id.input_path);
        inputPath.setText("image");

        builderSave.setPositiveButton("OK",
new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {

// Specify the path to the location where files are saved.
                    File filePath =
new File(Environment.getExternalStorageDirectory()
                            .getAbsolutePath() + "/SPen/images");
if (!filePath.exists()) {
if (!filePath.mkdirs()) {
                            Toast.makeText(mContext, "Save Path Creation Error",
                                Toast.LENGTH_SHORT).show();
return;
                        }
```

```
                    }
                    String saveFilePath = filePath.getPath() + '/';
                    String fileName = inputPath.getText().toString();
if (!fileName.equals("")) {
                        saveFilePath += fileName + ".png";
                        saveImageFile(saveFilePath);

                        closeSimpleView();
                    }
                }
            });
        builderSave.setNegativeButton("Cancel",
new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
                }
            });
        dlgSave = builderSave.create();
        dlgSave.show();
    }

private void saveImageFile(String strFileName) {
// Specify the name of the file to be captured.
        File fileCacheItem = new File(strFileName);
// Capture and save Bitmap.
        Bitmap imgBitmap = mSpenSimpleView.captureCurrentView();

        OutputStream out = null;
try {
// Save the captured Bitmap in the specified location.
            fileCacheItem.createNewFile();
            out = new FileOutputStream(fileCacheItem);
            imgBitmap.compress(CompressFormat.PNG, 100, out);
            Toast.makeText(mContext, "Captured images were stored.",
                Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
            Toast
                .makeText(mContext, "Capture failed.", Toast.LENGTH_SHORT)
                .show();
            e.printStackTrace();
        } finally {
try {
if(out!= null) {
                    out.close();
                }
                sendBroadcast(new Intent(Intent.ACTION_MEDIA_MOUNTED,
                    Uri.parse("file://"
                        + Environment.getExternalStorageDirectory())));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        imgBitmap.recycle();
    }

private void closeSimpleView() {
// Close SimpleView.
mSimpleViewBtn.setEnabled(true);
mSpenSimpleViewContainer.setVisibility(View.GONE);
```

```
    mSpenSimpleView.setVisibility(View.GONE);
    mSpenSimpleView.close();
    mSpenSimpleView = null;
        }

    @Override
    protected void onDestroy() {
    super.onDestroy();

    if(mSpenSimpleView != null) {
    mSpenSimpleView.close();
    mSpenSimpleView = null;
            }

    if(mSpenSimpleSurfaceView != null) {
    mSpenSimpleSurfaceView.close();
    mSpenSimpleSurfaceView = null;
            }

    if(mSpenNoteDoc != null) {
    try {
    mSpenNoteDoc.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
    mSpenNoteDoc = null;
            }
        }
}
```

For more information, see PenSample4_1_SimpleView.java in PenSample4_1_SimpleView.

The following sections provide more details on the steps involved in adding Simple View to your application.

4.4.1.1 Setting Background Images

To set a background image in your application:

1. Specify the background image of the main SpenSimpleSurfaceView instance to make it easier to understand the SimpleView functionality.
2. Decode the resource file 'smemo_bg.jpg' and save it into the file folder of the application.
3. Call `SpenPageDoc.setBackgroundImage()`.
4. Use the BACKGROUND_IMAGE_MODE_STRETCH option to stretch the background image to the full size of the screen when calling `setBackgroundImageMode()`.

```
String path = mContext.getFilesDir().getPath();
Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
                R.drawable.smemo_bg);
saveBitmapToFileCache(bitmap, path + "/smemo_bg.jpg");
mSpenPageDoc.setBackgroundImageMode(SpenPageDoc.BACKGROUND_IMAGE_MODE_STRETCH);
mSpenPageDoc.setBackgroundImage(path + "/smemo_bg.jpg");
```

Note

Pen SDK Light supports the following background image modes:

Image mode	Value	Description
BACKGROUND_IMAGE_MODE_CENTER	0	Draws the original image in the center.
BACKGROUND_IMAGE_MODE_STRETCH	1	Stretches the image to fit the screen.
BACKGROUND_IMAGE_MODE_FIT	2	Stretches the image to fit the screen and keeps the aspect ratio.
BACKGROUND_IMAGE_MODE_TILE	3	Repeats the original image to make it a tiling image.

4.4.1.2 Setting Up Simple View

To handle Simple View button events in your application:

1. Create a Simple View button.
2. Create an `OnClickListener` listener instance for the Simple View button, mSimpleViewBtnClickListener in the sample, and register it by calling `setOnClickListener()` onthe button
3. In the `onClick ()` method, disable the Simple View button to avoid repeated actions.
4. Display SimpleViewLayout, which is created with the active_simple_view.xml resource. The spenSimpleViewLayout view is a SimpleView area where users can enter stroke data, while mSpenSimpleViewContainer is an area that contains spenSimpleViewLayout and the button.

```
mSimpleViewBtn.setEnabled(false);
mSpenSimpleViewContainer = (RelativeLayout) findViewById(R.id.spenSimpleViewContainer);
mSpenSimpleViewContainer.setVisibility(View.VISIBLE);
RelativeLayout spenSimpleViewLayout = (RelativeLayout)
                                            findViewById(R.id.spenSimpleViewLayout);
```

Use `getLayoutParams()` to get information on the Simple View Layout.

Calculate the size of Simple View Layout at a specific ratio.

Call `setLayoutParams()` to set the size of SimpleViewLayout.

```
FrameLayout.LayoutParams simpleViewContainerParams =
(FrameLayout.LayoutParams) mSpenSimpleViewContainer.getLayoutParams();
FrameLayout.LayoutParams simpleViewLayoutParams =
                (FrameLayout.LayoutParams) spenSimpleViewLayout.getLayoutParams();
```

```
// Get the dimension of the screen of the device.
Display display = getWindowManager().getDefaultDisplay();
Rect rect = new Rect();
display.getRectSize(rect);
int btnHeight = 100;
// Resize SimpleView to the width of the screen at a random ratio.
if (rect.width() > rect.height()) {
          simpleViewContainerParams.width = (int) (rect.height() * .6);
          simpleViewContainerParams.height = (int) (rect.height() * .6) + btnHeight;
} else {
          simpleViewContainerParams.width = (int) (rect.width() * .6);
          simpleViewContainerParams.height = (int) (rect.width() * .6) + btnHeight;
}
simpleViewLayoutParams.width = (int) (simpleViewContainerParams.width * .9);
simpleViewLayoutParams.height = (int) ((simpleViewContainerParams.height)
- (simpleViewContainerParams.width * .1) - btnHeight);
mSpenSimpleViewContainer.setLayoutParams(simpleViewContainerParams);
spenSimpleViewLayout.setLayoutParams(simpleViewLayoutParams);
```

Create an SpenSimpleView instance with these dimensions and pass it when calling`addView()` to connect the instance with the SimpleViewLayout view and set up the pen for use in SpenSimpleView.

```
int screenWidth = simpleViewLayoutParams.width;
int screenHeight = simpleViewLayoutParams.height;
// Create SimpleView.
mSpenSimpleView = new SpenSimpleView(mContext, screenWidth, screenHeight);
spenSimpleViewLayout.addView(mSpenSimpleView);

initSimpleViewPenSettingInfo();
```

4.4.1.3 Registering a Listener for the Done Button in SimpleViewLayout

To handle Done button events in your application:

1. Create a Done button.

Create an`OnClickListener`listener for the Done button in the SpenSimpleView instance.

In the `onClick` method, save the image to the “SPen/images” folder in external storage under the user-defined name.

Call `SpenSimpleView.captureCurrentView()`to get the image of the current SpenSimpleView in Bitmap format.

Save the bitmap in PNG format and call `sendBroadcast()` with Intent.ACTION_MEDIA_MOUNTED to register the new image file in the gallery application.

Call `recycle()` to clear instances of SpenSimpleView to avoid memory leaks.

```
Button doneBtn = (Button) findViewById(R.id.done_btn);
doneBtn.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
                inputFileName();
        }
});

              .........

private void inputFileName() {

              .........

        builderSave.setPositiveButton("OK",
new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {

              .........

if (!fileName.equals("")) {
                        saveFilePath += fileName + ".png";
                        saveImageFile(saveFilePath);

                        closeSimpleView();
                    }

              .........

}

private void saveImageFile(String strFileName) {
// Specify the path to the file to be captured.
        File fileCacheItem = new File(strFileName);
// Capture and save the image in Bitmap format.
        Bitmap imgBitmap = mSpenSimpleView.captureCurrentView();

        OutputStream out = null;
try {
// Specify the path to the location of the captured Bitmap.
            fileCacheItem.createNewFile();
            out = new FileOutputStream(fileCacheItem);
            imgBitmap.compress(CompressFormat.PNG, 100, out);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
try {
if(out!= null) {
                    out.close();
                }

                sendBroadcast(new Intent(Intent.ACTION_MEDIA_MOUNTED,
                    Uri.parse("file://"
                        + Environment.getExternalStorageDirectory())));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
```

```
        imgBitmap.recycle();
}
```

4.4.1.4 Registering a Listener for the Cancel Button in SimpleViewLayout

To handle Cancel button events in your application:

1. Create a Cancel button
2. Create an `OnClickListener`instance for the Cancel button in SpenSimpleView.

In the`onClick() method`, enable the Simple View button, hide SimpleViewLayout, and clear the instances of SpenSimpleView to avoid memory leaks.

```
Button cancelBtn = (Button) findViewById(R.id.cancel_btn);
cancelBtn.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
               closeSimpleView();
return;
        }
});

              .........

private void closeSimpleView() {
// Close SimpleView.
mSimpleViewBtn.setEnabled(true);
mSpenSimpleViewContainer.setVisibility(View.GONE);
mSpenSimpleView.setVisibility(View.GONE);
mSpenSimpleView.close();
mSpenSimpleView = null;
}
```

4.4.2. Working Only with Pen SDK Light

If you are running your application using Pen SDK Light on the common Android View instead of Pen SDK Light -provided SpenView or SpenSimpleSurfaceView, you can only use SpenPen.

The sample application implements the following features:

- It creates a Pen SDK Light instance and a Bitmap sized for the viewport.
- It links the Pen SDK Light and the view.

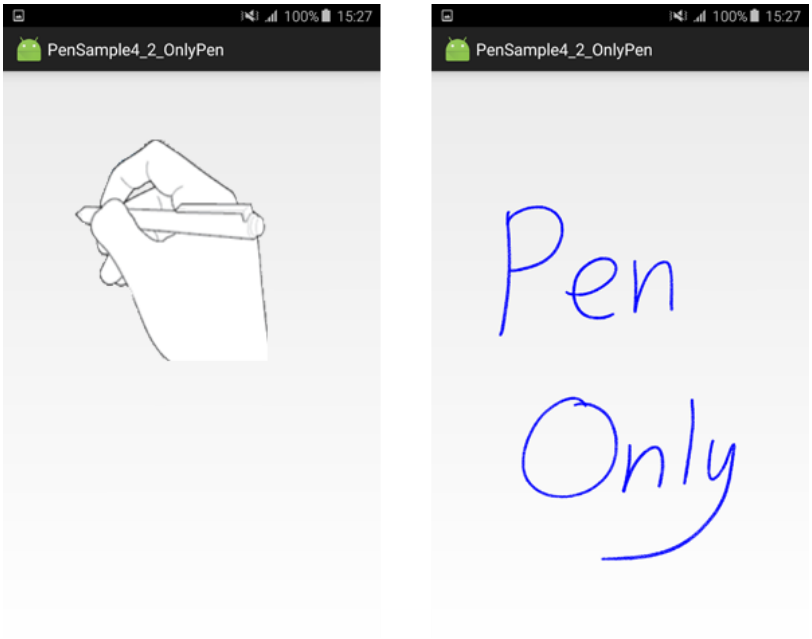


Figure 19: Pen SDK Light drawing on Android View

```
public class PenSample4_2_OnlyPen extends Activity {

private SpenPen mPen;
private SpenPenManager mPenManager;
private Bitmap mBitmap;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        Context context = this;

// Initialize Pen.
        Spen spenPackage = new Spen();
try {
spenPackage.initialize(this);
        } catch (SSDK UnsupportedException e) {
if( SDK Utils.processUnsupportedException(this, e) == true) {
return;
            }
        } catch (Exception e1) {
            Toast.makeText(context, "Cannot initialize Pen.",
Toast.LENGTH_SHORT).show();
            e1.printStackTrace();
            finish();
        }

        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);

// The pen manager gets the configurations for the pen to set up the pen.
```

```
mPenManager = new SpenPenManager(context);
        SpenPenInfo penInfo = new SpenPenInfo();
        List<SpenPenInfo> penInfoList = mPenManager.getPenInfoList();
for (SpenPenInfo info : penInfoList) {
if(info.name.equalsIgnoreCase("Brush")) {
                penInfo = info;
break;
            }
        }
try {
mPen = mPenManager.createPen(penInfo);
        } catch (ClassNotFoundException e) {
            Toast.makeText(context, "SpenPenManager class not found.",
                Toast.LENGTH_SHORT).show();
            e.printStackTrace();
        } catch (InstantiationException e) {
            Toast.makeText(context,
               "Failed to access the SpenPenManager constructor.",
                Toast.LENGTH_SHORT).show();
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            Toast.makeText(context,
               "Failed to access the SpenPenManager field or method.",
                Toast.LENGTH_SHORT).show();
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(context, "SpenPenManager is not loaded.",
                Toast.LENGTH_SHORT).show();
        }
mPen.setSize(10);
mPen.setColor(Color.BLUE);

// Get the dimensions of the screen and set the View.
        Display display = getWindowManager().getDefaultDisplay();
        Rect mScreenSize = new Rect();
        display.getRectSize(mScreenSize);

        View view = new MyView(context, mScreenSize.width(), mScreenSize.height());
        setContentView(view);
    }

protectedclass MyView extends View {

private RectF bitmapRect = new RectF();

public MyView(Context context, int w, int h) {
super(context);
            createBitmap(w, h);
        }

@Override
public booleanonTouchEvent(MotionEvent event) {
            RectF tempRect = new RectF();
// Get the touch event to draw as the pen draws
if (mBitmap != null) {
mBitmap.setPixel(0, 0, 0);
            }
mPen.draw(event, tempRect);
```

```
            invalidate(convertRect(tempRect));

return true;
        }

@Override
protected void onDraw(Canvas canvas) {
// Display the bitmap that the pen draws on the canvas.
            canvas.drawBitmap(mBitmap, null, bitmapRect, null);
super.onDraw(canvas);
        }

private void createBitmap(int w, int h) {
// Create a new bitmap and set it to the pen to enablepen drawing.
mBitmap = Bitmap.createBitmap(w, h, Config.ARGB_8888);
bitmapRect.set(0, 0, mBitmap.getWidth(), mBitmap.getHeight());
mPen.setBitmap(mBitmap);
        }

private Rect convertRect(RectF src) {
// Convert the RectF of the bitmap to be updated into Rect.
            Rect dst = new Rect();
            dst.left = (int) src.left;
            dst.right = (int) src.right;
            dst.top = (int) src.top;
            dst.bottom = (int) src.bottom;

return dst;
        }
    }

@Override
protected void onDestroy() {
super.onDestroy();

mPenManager.destroyPen(mPen);
mBitmap.recycle();
    }
}
```

For more information, seePenSample4_2_OnlyPen.java in PenSample4_2_OnlyPen.

4.4.2.1 Loading Pen SDK Light Plug-ins

To load a Pen SDK Light plug-in:

1. Create an SpenPenManager instance.

Call `SpenPenManager.getPenInfoList()` to get the list of pen information objects on the available Pen SDK Light plug-ins.

Select an appropriate Pen SDK Light plug-in from the list and call `SpenPenManager.createPen()`with the associated pen information object. The sample uses the name “Brush” to create a “Brush” SpenPen instance.

```
mPenManager = new SpenPenManager(context);
SpenPenInfo penInfo = new SpenPenInfo();
List<SpenPenInfo> penInfoList = mPenManager.getPenInfoList();
for (SpenPenInfo info : penInfoList) {
if(info.name.equalsIgnoreCase("Brush")) {
           penInfo = info;
break;
       }
}
try {
mPen = mPenManager.createPen(penInfo);
} catch (ClassNotFoundException e) {
} catch (InstantiationException e) {
} catch (IllegalAccessException e) {
} catch (Exception e) {
}
mPen.setSize(10);
mPen.setColor(Color.BLUE);
```

If you know the class name, you can use the full class name to create a pen instance without getting the pen information. The preloaded class names that you can use are listed below:

Note

Pen SDK Light supports the following pre-loadedpen plug-ins:

Name	Value	Class Name
InkPen	SPEN_INK_PEN	com.samsung.android.SDK .pen.pen.preload.InkPen
Pencil	SPEN_PENCIL	com.samsung.android.SDK .pen.pen.preload.Pencil
Marker	SPEN_MARKER	com.samsung.android.SDK .pen.pen.preload.Marker
Brush	SPEN_BRUSH	com.samsung.android.SDK .pen.pen.preload.Brush
ChineseBrush	SPEN_CHINESE_BRUSH	com.samsung.android.SDK .pen.pen.preload.ChineseBrush

The following sample code shows how to create a pen instance with a class name defined as a static variable:

```
SpenPenManager mPenManager = new SpenPenManager(context);
SpenPen mPen = mPenManager.createPen(SpenPenManager.SPEN_BRUSH);
```

4.4.2.2 Linking the Pen SDK Light Plug-in and Viewport

To link the pen plug-in and the viewport:

1. Inherit the Android View class to create a view that displays the object data drawn with finger or with S pen input.

Create a bitmap of the viewport.

Call `SpenPen.setBitmap()` to link the pen plug-in and viewport to enable users to draw objects.

```
protectedclass MyView extends View {
private RectF bitmapRect = new RectF();

public MyView(Context context, int w, int h) {
super(context);
            createBitmap(w, h);
        }

private void createBitmap(int w, int h) {
// Create a new bitmap and set it to the pen instance to enable pen drawing
mBitmap = Bitmap.createBitmap(w, h, Config.ARGB_8888);
bitmapRect.set(0, 0, mBitmap.getWidth(), mBitmap.getHeight());
mPen.setBitmap(mBitmap);
        }
```

4.4.2.3 Handling Drawing

To handle touch events:

1. In the onTouchEvent()method, call SpenPen.draw() and pass the event.The SpenPen instance drawsthe objects and getsthe RectF values representing the area where the objects are drawn.

To convert the RectF values to Rect, call invalidate().

In theonDraw()method,callcanvas.drawBitmap()to display the bitmap drawn by the pen on the canvas.

```
public booleanonTouchEvent(MotionEvent event) {
            RectF tempRect = new RectF();
// Get the touch event to draw as the pen draws
if (mBitmap != null) {
mBitmap.setPixel(0, 0, 0);
            }
mPen.draw(event, tempRect);
            invalidate(convertRect(tempRect));

return true;
}

@Override
protected void onDraw(Canvas canvas) {
// Display the bitmap that the pen draws on the canvas.
            canvas.drawBitmap(mBitmap, null, bitmapRect, null);
super.onDraw(canvas);
}

private Rect convertRect(RectF src) {
// RectF of the Bitmap to be updated is converted into Rect.
            Rect dst = new Rect();
            dst.left = (int) src.left;
            dst.right = (int) src.right;
            dst.top = (int) src.top;
            dst.bottom = (int) src.bottom;

return dst;
}
```

4.4.2.4 Preventing Memory Leaks

To prevent memory leaks:

1. Call `SpenPenManager.destroyPen()` to unload the SpenPen plug-in.

Call `Bitmap.recycle()` to release the resources.

Copyright

Though every care has been taken to ensure the accuracy of this document, Samsung Electronics Co., Ltd.cannot accept responsibility for any errors or omissions or for any loss occurred to any person,whether legal or natural, from acting, or refraining from action, as a result of the information contained herein. Information in this document is subject to change at any time without obligation to notify any person of such changes.

Samsung Electronics Co. Ltd.may have patents or patent pending applications, trademarks copyrights or other intellectual property rights covering subject matter in this document.The furnishing of this document does not give the recipient or reader any license to these patents, trademarks copyrights or other intellectual property rights.

No part of this document may be communicated, distributed, reproduced or transmitted in any form or by any means, electronic or mechanical or otherwise, for any purpose, without the prior written permission of Samsung Electronics Co. Ltd.

The document is subject to revision without further notice.

All brand names and product names mentioned in this document are trademarks or registered trademarks of their respective owners.

For more information, please visit http://developer.samsung.com/