Pen

Programming Guide

Version 3.1.4

Table of Contents

1. Overview

Pen 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 offers a richer set of features than existing input tools. Because it senses the pressure underneath its tip, Pen makes it feel more like you are actually writing or drawing on the device.

Pen provides functions for verifying if the S pen 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 to:

- draw using a finger and/or S pen
- set user preferences for pens, erasers, and text
- edit and save input objects (text, strokes, and images) as a file
- manage history for undo and redo commands

1.1. Basic Knowledge

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

The following figure shows the relationship between SpenPageDoc and SpenSurfaceView.

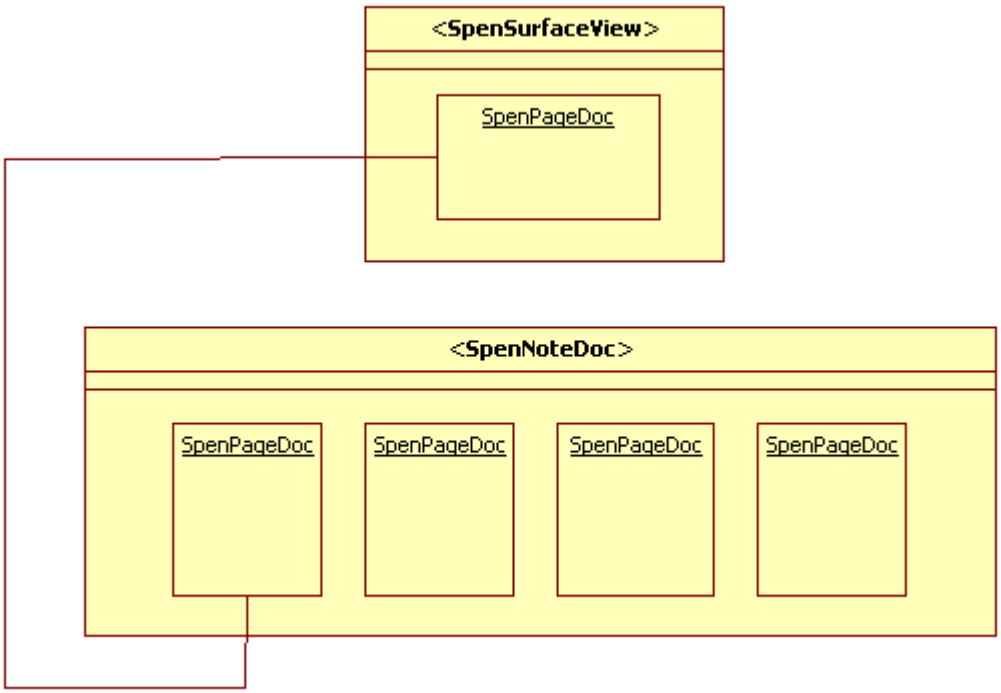


Figure 1: Relationship between SpenPageDoc and SpenSurfaceView

1.2. Architecture

The following figure shows the Pen architecture.

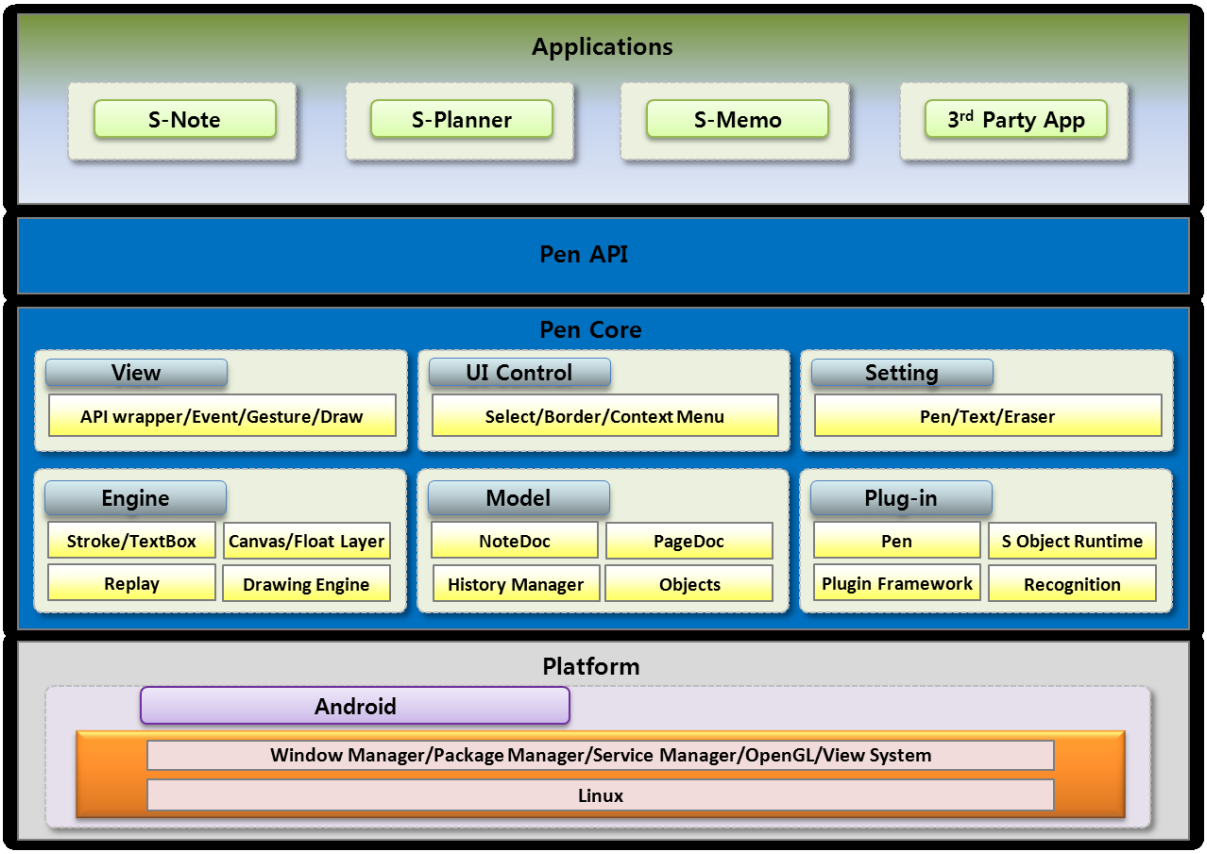


Figure 2: Pen architecture

The architecture consists of:

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

1.3. Package Diagram

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

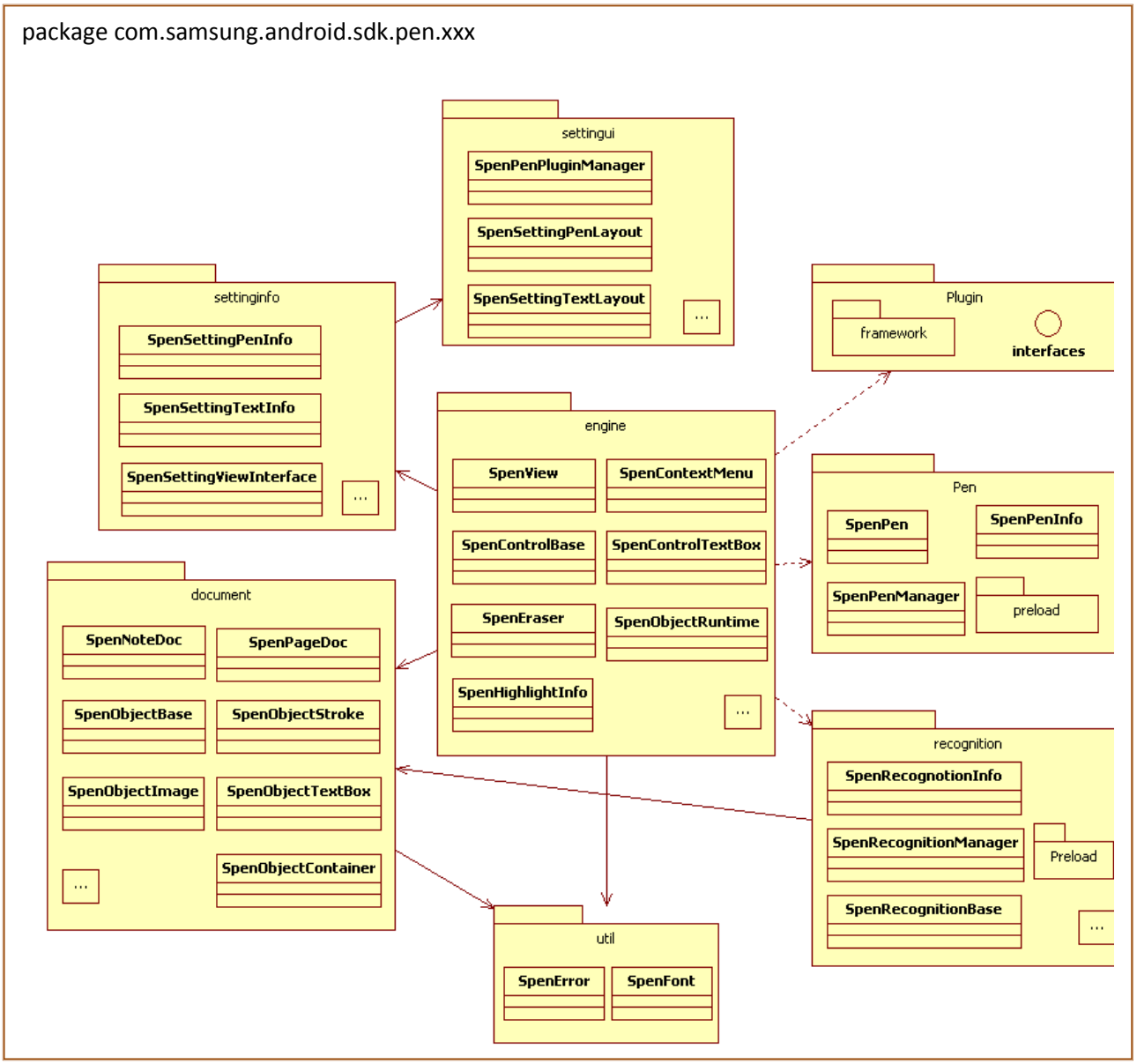


Figure 3: Pen packages and classes

The Pen 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.
- **SpenObjectTextBox:** Manages text input. Each instance corresponds to a text box.
- **SpenObjectImage:** Manages images. Each instance corresponds to an image.
- **SpenObjectContainer:** Manages groups of user strokes, text boxes and images.
- **SpenSurfaceView:** 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.

1.5. Supported Features

Pen supports the following features:

- Processing S pen touch events and hover events, sensing S pen pressure and checking if the side button is pressed.
- Processing handwritten input with a finger or a S pen and converting it to text or a vector image.
- Zoom in and out and pan on the viewport.
- Managing user preferences for pens, erasers, and text.
- Managing input objects and maintaining their states.
- Selecting, scaling, moving, rotating, grouping, and ungrouping objects.
- Managing the history of an input object.
- Adding third party templates and runtime objects.

1.6. Components

- Components
    - pen-v3.1.4.jar
    - sdk-v1.0.0.jar
- Imported Pen:
    - com.samsung.android.sdk.pen

1.7. Importing Libraries

To import Pen libraries to the application project:

Add the pen-v3.1.4.jar and sdk-v1.0.0.jar files to the libs folder.

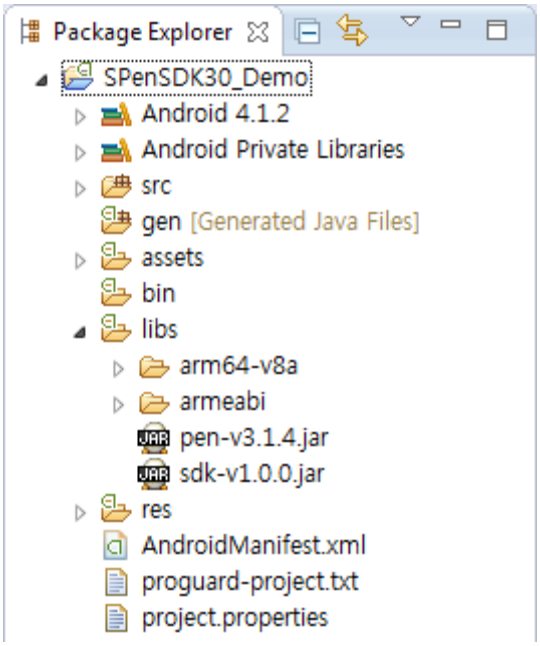


Figure 4: libs folder in Eclipse

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

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

Add the following permission to your Android manifest file to access the camera from methods such as `SpenSurfaceView.takeStrokeFrame()` and `SpenSurfaceView.retakeStrokeFrame()`.

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

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 file to initialize Pen.

```
<uses-permission android: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. Hello Pen

Hello Pen 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 SpenSurfaceView mSpenSurfaceView;

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

        // Initialize Pen.
        boolean isSpenFeatureEnabled = false;
        Spen spenPackage = new Spen();
        try {
            spenPackage.initialize(this);
            isSpenFeatureEnabled =
                  spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
        } catch (SsdkUnsupportedException e) {
            Toast.makeText(mContext, "This device does not support S pen.",
                Toast.LENGTH_SHORT).show();
            e.printStackTrace();
            finish();
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Pen.",
                Toast.LENGTH_SHORT).show();
            e1.printStackTrace();
            finish();
        }

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

        // Get the dimensions of the 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();
        }
        // 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.
        mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);

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

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

        if(mSpenSurfaceView != null) {
            mSpenSurfaceView.close();
            mSpenSurfaceView = 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. You need to initialize Pen before you can use it. If the device does not support S Pen, SsdkUnsupportedException is thrown.
- `getVersionCode()` returns the Pen SDK version number as an integer.
- `getVersionName()` returns the Pen SDK version name as a string.
- `isFeatureEnabled()` checks if a Pen feature is available on the device.

```
boolean isSpenFeatureEnabled = false;
Spen spenPackage = new Spen();
try {
       spenPackage.initialize(this);
       isSpenFeatureEnabled = spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
} catch (SsdkUnsupportedException e) {
       int eType = e.getType();
       if (eType == SsdkUnsupportedException.VENDOR_NOT_SUPPORTED) {
           // The device is not a Samsung device.
       } else if (eType == SsdkUnsupportedException.DEVICE_NOT_SUPPORTED) {
           // The device does not support Pen.
       } else if (eType == SsdkUnsupportedException.LIBRARY_NOT_INSTALLED) {
           // SpenSdk3.apk is not installed on the device.
       } else if (eType
                   == SsdkUnsupportedException.LIBRARY_UPDATE_IS_REQUIRED) {
           // The Pen library or SpenSdk3.apk requires to be updated.
       } else if (eType
                   == SsdkUnsupportedException.LIBRARY_UPDATE_IS_RECOMMENDED) {
           // It is recommended that the Pen library or SpenSdk3.apk is updated 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
- checks if the device is a Samsung device
- checks if the Samsung device supports Pen

- checks if SpenSdk3.apk are installed on the device

```
void initialize(Context context) throws SsdkUnsupportedException
```

If initializing Pen is failed, the `initialize()` method throws an SsdkUnsupportedException exception. To find out the reason for the exception, check the exception message.

3.2. Handling SsdkUnsupportedException

If an SsdkUnsupportedException exception is thrown, check the exception message type using `SsdkUnsupportedException.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.
- LIBRARY_NOT_INSTALLED: SpenSdk3.apk is not installed on the device.
- LIBRARY_UPDATE_IS_REQUIRED: A necessary update for the Pen library or SpenSdk3.apk. If the library or apk is not updated, the user cannot use the application.
- LIBRARY_UPDATE_IS_RECOMMENDED: A recommendation to update the Pen library or SpenSdk3.apk, but it is not mandatory. The user can use the application without updating them.

3.3. Checking the Availability of Pen Features

You can check if a Pen feature is supported on the device with the `isFeatureEnabled()` method. The feature types are 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 `SpenSurfaceView.setToolTypeAction()` and set TOOL_FINGER to ACTION_STROKE to enable input with user fingers.

```
if(spenPackage.isFeatureEnabled(Spen.DEVICE_PEN) == false) {
    mSpenSurfaceView.setToolTypeAction(SpenSurfaceView.TOOL_FINGER,
                SpenSurfaceView.ACTION_STROKE);
    Toast.makeText(mContext,
                "Device does not support S pen. \n You can draw strokes with your
```

```
finger",
    Toast.LENGTH_SHORT).show();
}
```

4. Using Pen

4.1. Using Pen Views

SpenSurfaceView, which inherits from Android SurfaceView, processes finger gestures or S pen input to express drawings on the viewport. It also converts handwritten input to text or replays user input.

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

SpenSurfaceView 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 SpenSurfaceView 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 SpenSurfaceView mSpenSurfaceView;
```

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

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

    // Create a Pen View.
    RelativeLayout spenViewLayout =
        (RelativeLayout) findViewById(R.id.spenViewLayout);
    mSpenSurfaceView = new SpenSurfaceView(mContext);
    if (mSpenSurfaceView == null) {
        Toast.makeText(mContext, "Cannot create new SpenSurfaceView.",
            Toast.LENGTH_SHORT).show();
        finish();
    }
    spenViewLayout.addView(mSpenSurfaceView);

    // Get the dimensions of the screen of the device.
    Display display = getWindowManager().getDefaultDisplay();
    Rect rect = new Rect();
    display.getRectSize(rect);
    // Create an 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();
    }
    // After adding a page to NoteDoc get the instance and
    // set it as a member variable.
    mSpenPageDoc = mSpenNoteDoc.appendPage();
    mSpenPageDoc.setBackgroundColor(0xFFD6E6F5);
    mSpenPageDoc.clearHistory();
    // Set PageDoc to View.
```

```
        mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);

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

    private void processUnsupportedException(SsdkUnsupportedException e) {

        e.printStackTrace();
        int errorType = e.getType();
        // The device is not a Samsung device or it is a Samsung device that does not
support S pen.
        if(errorType == SsdkUnsupportedException.VENDOR_NOT_SUPPORTED ||
                errorType == SsdkUnsupportedException.DEVICE_NOT_SUPPORTED ) {
            Toast.makeText(mContext, "This device does not support S pen.",
                    Toast.LENGTH_SHORT).show();
            finish();
        } else if(errorType == SsdkUnsupportedException.LIBRARY_NOT_INSTALLED) {
            // SpenSdk3.apk is not installed on the device.
            showAlertDialog( "You need to install an additional package"
                + " to use this application."
                + "You will be taken to the installation screen."
                + "Restart this application after the software has been installed."
                , true);
        } else if(errorType ==
                    SsdkUnsupportedException.LIBRARY_UPDATE_IS_REQUIRED) {
            // The Pen library or SpenSdk3.apk requires to be updated.
            showAlertDialog("You need to update the installed Pen library or package"
                + "to use this application."
                + " You will be taken to the installation screen."
                + " Restart this application after the software has been updated."
                , true);
        } else if(errorType ==
                    SsdkUnsupportedException.LIBRARY_UPDATE_IS_RECOMMENDED) {
            // It is recommended that the Pen library or SpenSdk3.apk is updated to
            // the latest version as possible.
            showAlertDialog("We recommend that you update the installed Pen library or
package"
                + " 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 SpenSdk3.apk.
                            Uri uri = Uri.parse("market://details?id="
                                    + Spen.SPEN_NATIVE_PACKAGE_NAME);
                            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 the user does not wish to
   install and closes the dialog.
                                finish();
                            }
                            dialog.dismiss();
                        }
                 }).show();
            dlg = null;
        }

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

            if(mSpenSurfaceView != null) {
                mSpenSurfaceView.close();
                mSpenSurfaceView = 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

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

```
Spen spenPackage = new Spen();
try {
       spenPackage.initialize(this);
} catch (SsdkUnsupportedException e) {
       if( processUnsupportedException(e) == true) {
           return;
       }
} catch (Exception e1) {
       e1.printStackTrace();
       finish();
}
```

Pen runs only on Samsung devices. The `Spen.initialize()` method throws an SsdkUnsupportedException exception on other devices. Handle the SsdkUnsupportedException 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.
- Call `finish()` to close the application.

If `SpenSdk3.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 `SpenSdk3.apk` and open the website to download the package.

```
private void processUnsupportedException(SsdkUnsupportedException e) {

        e.printStackTrace();
        int errorType = e.getType();
        // The device is not a Samsung device or it is a Samsung device that does not
support S pen.
        if(errorType == SsdkUnsupportedException.VENDOR_NOT_SUPPORTED ||
                errorType == SsdkUnsupportedException.DEVICE_NOT_SUPPORTED ) {
            Toast.makeText(mContext, "This device does not support S pen.",
                    Toast.LENGTH_SHORT).show();
            finish();
        } else if(errorType == SsdkUnsupportedException.LIBRARY_NOT_INSTALLED) {
            // SpenSdk3.apk is not installed on the device.
            showAlertDialog( "You need to install an additional package"
                + " to use this application."
                + "You will be taken to the installation screen."
                + "Restart this application after the software has been installed."
                , true);
        } else if(errorType ==
                  SsdkUnsupportedException.LIBRARY_UPDATE_IS_REQUIRED) {
            // The Pen library or SpenSdk3.apk requires to be updated.
            showAlertDialog( "You need to update the installed Pen library or package"
                + "to use this application."
                + " You will be taken to the installation screen."
```

```
                + " Restart this application after the software has been updated."
                , true);
        } else if(errorType ==
                  SsdkUnsupportedException.LIBRARY_UPDATE_IS_RECOMMENDED) {
            // It is recommended that the Pen library or SpenSdk3.apk is updated
            // the latest version as possible.
            showAlertDialog( "We recommend that you update the installed Pen library or
pakcage"
                + " 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 the Pen library.
                        // or package
                        Uri uri = Uri.parse("market://details?id="
                                + Spen.SPEN_NATIVE_PACKAGE_NAME);
                        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 the user does not install it.
                            finish();
                        }
                        dialog.dismiss();
                    }
            }).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 `SpenSurfaceView.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) {
        mSpenSurfaceView.setToolTypeAction(SpenSurfaceView.TOOL_FINGER,
                SpenSurfaceView.ACTION_STROKE);
        Toast.makeText(mContext,
                "Device does not support S pen. \n You can draw strokes with your
finger",
                Toast.LENGTH_SHORT).show();
}
```

4.1.1.3 Creating SpenSurfaceView

To create a drawing container in your application:

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

2. Add the SpenSurfaceView instance to the main layout.

```
RelativeLayout spenViewLayout = (RelativeLayout) findViewById(R.id.spenViewLayout);
mSpenSurfaceView = new SpenSurfaceView(mContext);
if (mSpenSurfaceView == null) {
       finish();
}
spenViewLayout.addView(mSpenSurfaceView);
```

4.1.1.4 Connecting SpenPageDoc to SpenSurfaceView

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

1. Create an SpenNoteDoc instance by passing your application Context and the width and height of the SpenSurfaceView instance to the SpenNoteDoc constructor. If Pen 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 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 `SpenSurfaceView.setPageDoc()` method to connect the SpenPageDoc instance to your SpenSurfaceView 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.
mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);
```

4.1.1.5 Preventing Memory Leaks

To prevent memory leaks:

1. Call `SpenNoteDoc.close()` and `SpenSurfaceView.close()` to close the SpenNoteDoc and SpenSurfaceView 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 SpenSurfaceView in the `onDestroy()` method.

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

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

Note

Once you create an SpenSurfaceView instance, Pen 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 SpenSurfaceView.setToolTypeAction() and set TOOL_FINGER to ACTION_NONE. By default, Pen sets TOOL_SPEN to ACTION_STROKE, which allows the S pen to draw strokes on the screen.

```
mSpenSurfaceView.setToolTypeAction(SpenSurfaceView.TOOL_FINGER,
SpenSurfaceView.ACTION_NONE);
```

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

Pen 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 supports the following action types.

Action type	Value	Description
ACTION_NONE	0	No action.
ACTION_GESTURE	1	Finger gesture.
ACTION_STROKE	2	Pen stroke.
ACTION_ERASER	3	Eraser tool.
ACTION_STROKE_REMOVER	4	Erasing pen stroke.
ACTION_COLOR_PICKER	5	Color picker.
ACTION_SELECTION	6	Selection.
ACTION_TEXT	7	Text.

4.1.2. Adding Pen 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. 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.

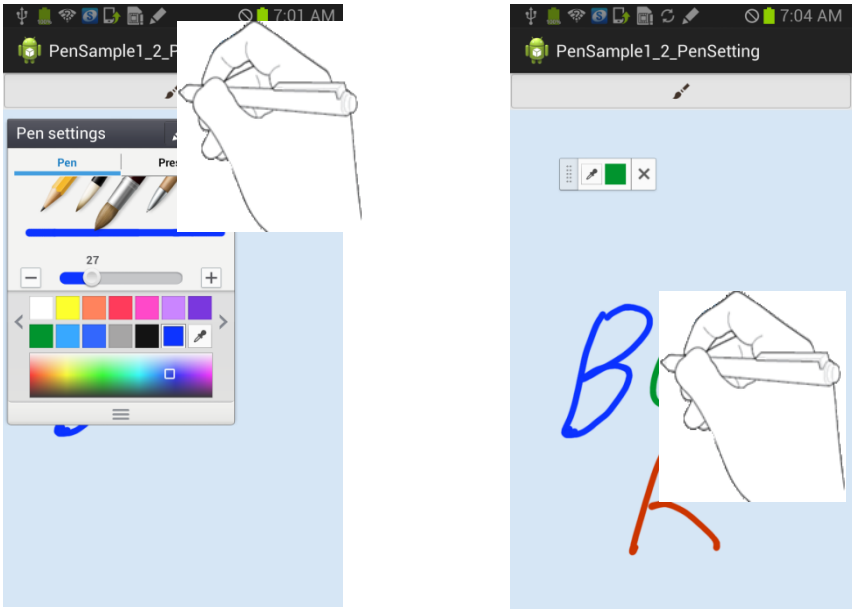


Figure 6: Pen settings and color picker

```
public class PenSample1_2_PenSetting extends Activity {

	private Context mContext;
	private SpenNoteDoc mSpenNoteDoc;
	private SpenPageDoc mSpenPageDoc;
	private SpenSurfaceView mSpenSurfaceView;
	private SpenSettingPenLayout mPenSettingView;

	private ImageView mPenBtn;

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

	    // Initialize Pen.
	    boolean isSpenFeatureEnabled = false;
	    Spen spenPackage = new Spen();
	    try {
	        spenPackage.initialize(this);
	        isSpenFeatureEnabled = spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
	    } catch (SsdkUnsupportedException e) {
	        if( SDKUtils.processUnsupportedException(this, e) == true) {
	            return;
```

```
            }
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Pen.",
                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, new String(),
                spenViewLayout);
        if (mPenSettingView == null) {
            Toast.makeText(mContext, "Cannot create new PenSettingView.",
                Toast.LENGTH_SHORT).show();
            finish();
        }
        spenViewContainer.addView(mPenSettingView);

        // Create Pen View.
        mSpenSurfaceView = new SpenSurfaceView(mContext);
        if (mSpenSurfaceView == null) {
            Toast.makeText(mContext, "Cannot create new SpenSurfaceView.",
                Toast.LENGTH_SHORT).show();
            finish();
        }
        spenViewLayout.addView(mSpenSurfaceView);
        mPenSettingView.setCanvasView(mSpenSurfaceView);

        // Get the dimensions of the screen of the device.
        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();
        }
        // 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.
        mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);
```

```
        initPenSettingInfo();
        mSpenSurfaceView.setColorPickerListener(mColorPickerListener);

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

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

    private void initPenSettingInfo() {
        // Initialize pen settings.
        SpenSettingPenInfo penInfo = new SpenSettingPenInfo();
        penInfo.color = Color.BLUE;
        penInfo.size = 10;
        mSpenSurfaceView.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(mSpenSurfaceView != null) {
                mSpenSurfaceView.close();
                mSpenSurfaceView = 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 SDKUtils {

    public static boolean processUnsupportedException(final Activity activity,
            SsdkUnsupportedException e) {
        e.printStackTrace();
        int errType = e.getType();
        // The device is not a Samsung device or it is a Samsung device that does not
   support S pen.

        if(errType == SsdkUnsupportedException.VENDOR_NOT_SUPPORTED ||
                errType == SsdkUnsupportedException.DEVICE_NOT_SUPPORTED ) {
            Toast.makeText(activity, "This device does not support S pen.",
                    Toast.LENGTH_SHORT).show();
            activity.finish();
        } else if( errType == SsdkUnsupportedException.LIBRARY_NOT_INSTALLED ) {
            // SpenSdk3.apk is not installed on the device.
            showAlertDialog( activity,
                "You need to install an additional package"
                +" 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 == SsdkUnsupportedException.LIBRARY_UPDATE_IS_REQUIRED ) {
            // The Pen library or SpenSdk3.apk requires to be updated.
            showAlertDialog( activity,
                "You need to update the installed Pen library or package"
                + "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 == SsdkUnsupportedException.LIBRARY_UPDATE_IS_RECOMMENDED )
{
            // It is recommended that the Pen library or SpenSdk3.apk is updated
            // the latest version as possible.
            showAlertDialog( activity,
                "We recommend that you update the installed Pen library or pakcage"
```

```
                + " before using this application."
                + " You will be taken to the installation screen."
                + " Restart this application after the software has been updated."
                , false);
            return false; // Run the application normally if the user does not install
it.
        }
        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 website and install/update the Pen library
or package.
                        Uri uri = Uri.parse("market://details?id="
                                + Spen.SPEN_NATIVE_PACKAGE_NAME);
                        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 the user does not install it.
                            activity.finish();
                        }
                        dialog.dismiss();
                    }
            })
            .setOnCancelListener(new DialogInterface.OnCancelListener() {
                @Override
                public void onCancel(DialogInterface dialog) {
                    if(closeActivity == true) {
                        // Terminate the activity if the user does not install it.
                        activity.finish();
                    }
                }
            })
            .show();
        dlg = null;
```

```
    }
}
```

For more information, see SDKUtils.java.

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

4.1.2.1 Creating SpenSurfaceView and SpenSettingPenLayout

To create the layout for the pen and settings view in your application:

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

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

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

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

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 `SpenSurfaceView.setPenSettingInfo()` to save the settings modified on your SpenSurfaceView instance.

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

```
private void initPenSettingInfo() {
        SpenSettingPenInfo penInfo = new SpenSettingPenInfo();
        penInfo.color = Color.BLUE;
        penInfo.size = 10;
        mSpenSurfaceView.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 displayed, close PenSettingView.
                if (mPenSettingView.isShown()) {
                    mPenSettingView.setVisibility(View.GONE);
                //  If PenSettingView is not displayed, display it.
                } else {
                    mPenSettingView
                        .setViewMode(SpenSettingPenLayout.VIEW_MODE_EXTENSION);
                    mPenSettingView.setVisibility(View.VISIBLE);
                }
         }
};
```

Note

SpenSettingPenLayout provides the following view modes:

View mode	Value	Settings options for pen tool
VIEW_MODE_NORMAL	0	Type, size, and color

Note

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.

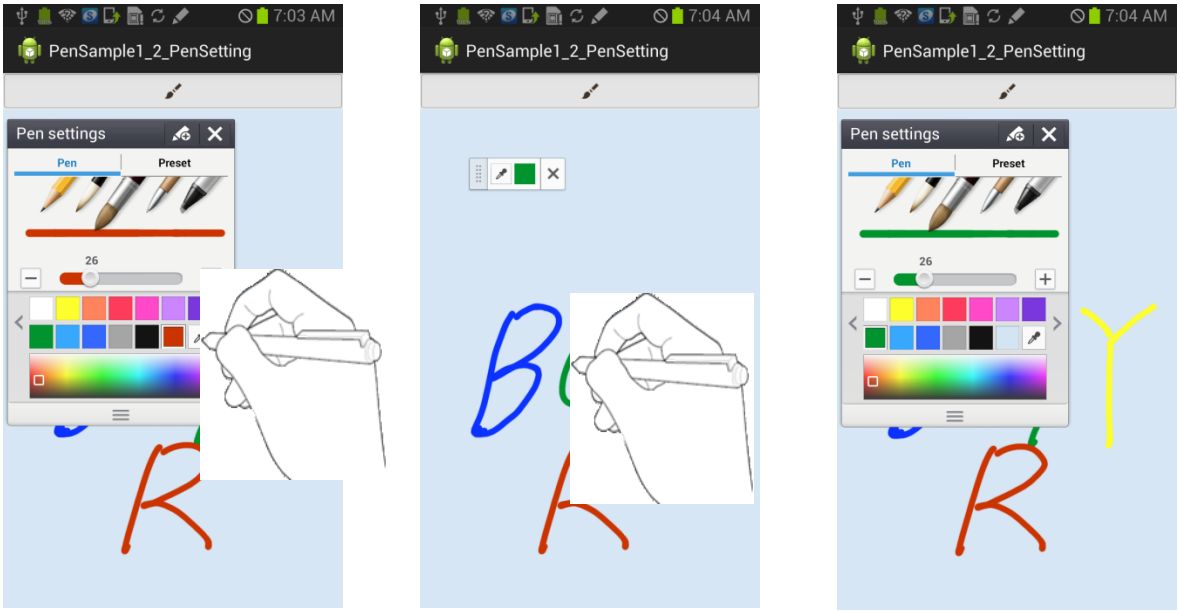


Figure 7: Color picker

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 SpenColorPickerListener mColorPickerListener =
        new SpenColorPickerListener() {
```

```
            @Override
            public void onChanged(int color, int x, int y) {
                // Set the color selected by the color picker for the setting view.
                if (mPenSettingView != null) {
                    SpenSettingPenInfo penInfo = mPenSettingView.getInfo();
                    penInfo.color = color;
                    mPenSettingView.setInfo(penInfo);
                }
            }
};
```

Register the listener by calling `SpenSurfaceView.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";`
`mSpenSurfaceView.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 SpenSurfaceView 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.

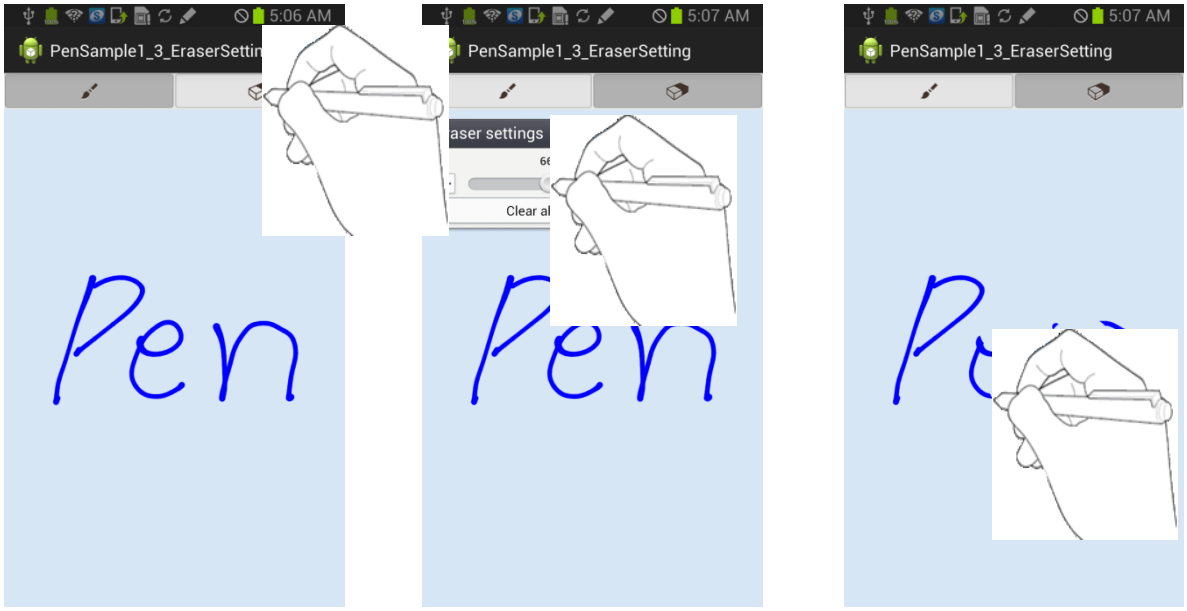


Figure 8: Eraser settings

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

private int mToolType = SpenSurfaceView.TOOL_SPEN;

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

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

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

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

            .........

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

        // Create Pen View.
        mSpenSurfaceView = new SpenSurfaceView(mContext);
        if (mSpenSurfaceView == null) {
            Toast.makeText(mContext, "Cannot create new SpenSurfaceView.",
                Toast.LENGTH_SHORT).show();
            finish();
        }
        spenViewLayout.addView(mSpenSurfaceView);
        mPenSettingView.setCanvasView(mSpenSurfaceView);
        mEraserSettingView.setCanvasView(mSpenSurfaceView);

            .........

        initSettingInfo();
        // Register the listeners.
        mSpenSurfaceView.setColorPickerListener(mColorPickerListener);
        mEraserSettingView.setEraserListener(mEraserListener);

        // Define the buttons.
        mPenBtn = (ImageView) findViewById(R.id.penBtn);
        mPenBtn.setOnClickListener(mPenBtnClickListener);

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

        selectButton(mPenBtn);

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

```
private void initSettingInfo() {

              .........

        // Initialize eraser settings.
        SpenSettingEraserInfo eraserInfo = new SpenSettingEraserInfo();
        eraserInfo.size = 30;
        mSpenSurfaceView.setEraserSettingInfo(eraserInfo);
        mEraserSettingView.setInfo(eraserInfo);
}

private final OnClickListener mPenBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                // If it is in pen tool mode.
                if (mSpenSurfaceView.getToolTypeAction(mToolType) ==
                        SpenSurfaceView.ACTION_STROKE) {
                    // If PenSettingView is displayed, close it.
                    if (mPenSettingView.isShown()) {
                        mPenSettingView.setVisibility(View.GONE);
                    // If PenSettingView is not displayed, display it.
                    } else {
                        mPenSettingView
                            .setViewMode(SpenSettingPenLayout.VIEW_MODE_EXTENSION);
                        mPenSettingView.setVisibility(View.VISIBLE);
                    }
                // If it is not in pen tool mode, change it to pen tool mode.
                } else {
                    selectButton(mPenBtn);
                    mSpenSurfaceView.setToolTypeAction(
                        mToolType,
                        SpenSurfaceView.ACTION_STROKE);
                }
            }
};

private final OnClickListener mEraserBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                // If it is in eraser tool mode.
                if (mSpenSurfaceView.getToolTypeAction(mToolType) ==
                        SpenSurfaceView.ACTION_ERASER) {
                    // If EraserSettingView is displayed, close it.
                    if (mEraserSettingView.isShown()) {
                        mEraserSettingView.setVisibility(View.GONE);
                    // If EraserSettingView is not displayed, display it.
                    } else {
                        mEraserSettingView
                            .setViewMode(SpenSettingEraserLayout.VIEW_MODE_NORMAL);
                        mEraserSettingView.setVisibility(View.VISIBLE);
                    }
                // If it is not in eraser tool mode, change it to eraser tool mode.
                } else {
                    selectButton(mEraserBtn);
                    mSpenSurfaceView.setToolTypeAction(
                        mToolType,
```

```
                        SpenSurfaceView.ACTION_ERASER);
                }
            }
};

              .........

private EventListener mEraserListener = new EventListener() {
        @Override
        public void onClearAll() {
            //Handle the Clear All button in EraserSettingView.
            mSpenPageDoc.removeAllObject();
            mSpenSurfaceView.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(SpenSurfaceView.GONE);
        mPenSettingView.setVisibility(SpenSurfaceView.GONE);
}

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

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

        if(mSpenSurfaceView != null) {
            mSpenSurfaceView.close();
            mSpenSurfaceView = 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 SpenSurfaceView and SpenSettingEraserLayout

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

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

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

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

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 `SpenSurfaceView.setEraserSettingInfo()` to save the settings to your SpenSurfaceView instance.

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

```
SpenSettingEraserInfo eraserInfo = new SpenSettingEraserInfo();
eraserInfo.size = 30;
mSpenSurfaceView.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) {
                // If it is in eraser tool mode.
                if (mSpenSurfaceView.getToolTypeAction(mToolType) ==
                        SpenSurfaceView.ACTION_ERASER) {
                    // If EraserSettingView is displayed, close it.
                    if (mEraserSettingView.isShown()) {
                        mEraserSettingView.setVisibility(View.GONE);
                    // If EraserSettingView is not displayed, display it.
                    } else {
                        mEraserSettingView
                            .setViewMode(SpenSettingEraserLayout.VIEW_MODE_NORMAL);
                        mEraserSettingView.setVisibility(View.VISIBLE);
                    }
                // If it is not in eraser tool mode, change it to eraser tool mode.
                } else {
                    selectButton(mEraserBtn);
                    mSpenSurfaceView.setToolTypeAction(
                        mToolType,
                        SpenSurfaceView.ACTION_ERASER);
                }
            }
};

              .........

private void selectButton(View v) {
        // Depending on the current mode, enable/disable the button.
        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:

Note

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 and text type for deleting text
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 SpenSurfaceView.TOOL_SPEN on devices that support S pen or to SpenSurfaceView.TOOL_FINGER on the devices that do not support S pen.

4.1.3.4 Registering a Listener for Clear All Events

When the Clear All button is clicked in the SpenSettingEraserLayout view, remove all the objects on the SpenSurfaceView 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 Clear All button in EraserSettingView.
            mSpenPageDoc.removeAllObject();
            mSpenSurfaceView.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 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 `SpenSurfaceView.updateUndo()` or `SpenSurfaceView.updateRedo()` update the SpenSurfaceView instance.
- History listener for receiving history state events.

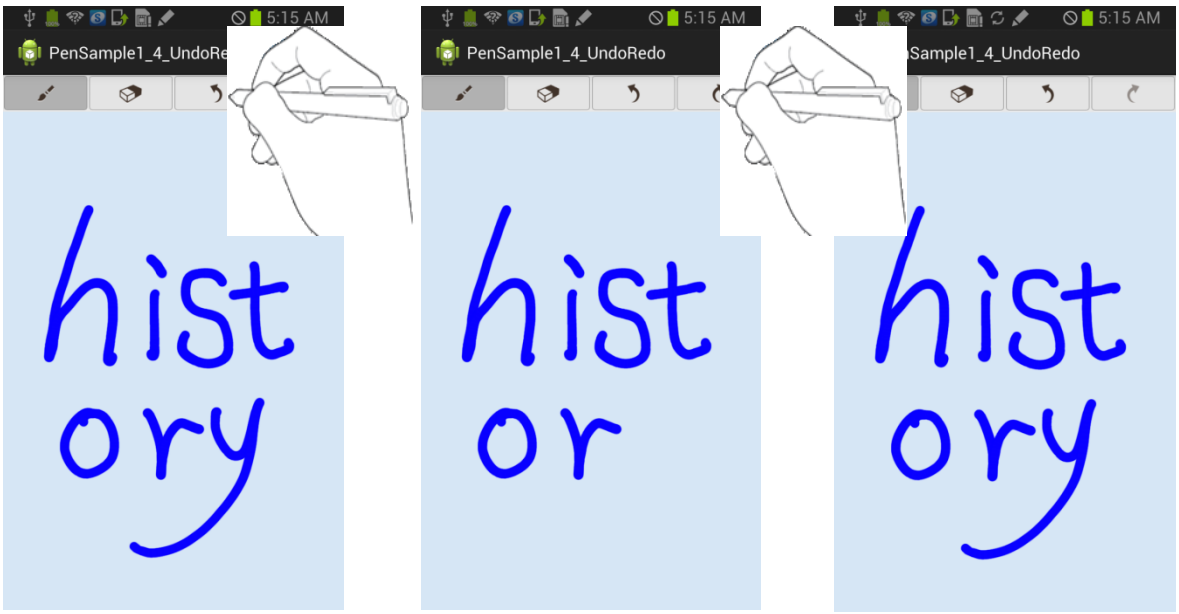


Figure 9: Undo/Redo function

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

              .........

        // Register the listeners.
        mSpenSurfaceView.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();
                        mSpenSurfaceView.updateUndo(userData);
                    }
                // Redo button is clicked.
                } else if (v.equals(mRedoBtn)) {
                    if (mSpenPageDoc.isRedoable()) {
                        HistoryUpdateInfo[] userData = mSpenPageDoc.redo();
                        mSpenSurfaceView.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 `SpenSurfaceView.updateUndo()` or `SpenSurfaceView.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();
                    mSpenSurfaceView.updateUndo(userData);
                }
            // Redo button is clicked.
            } else if (v.equals(mRedoBtn)) {
                if (mSpenPageDoc.isRedoable()) {
                    HistoryUpdateInfo[] userData = mSpenPageDoc.redo();
                    mSpenSurfaceView.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 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 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 `SpenSurfaceView.updateUndoAll()`.

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

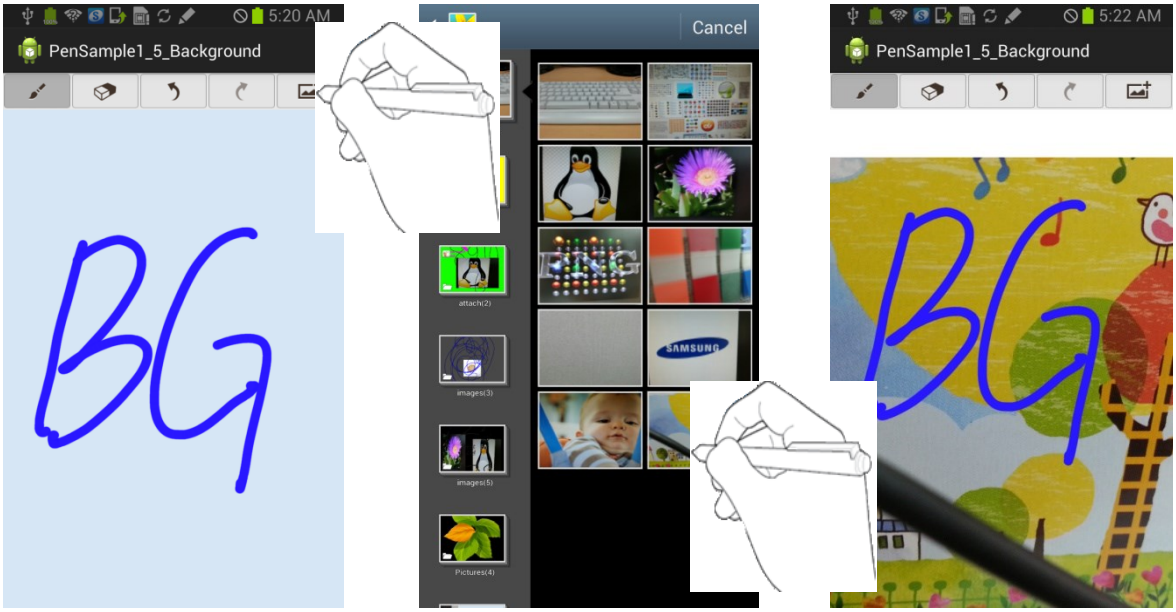


Figure 10: Background settings

```
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);
                mSpenSurfaceView.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 method for 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 `SpenSurfaceView.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);
            mSpenSurfaceView.update();
}
```

Note

Note
Pen 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. Using Replay Animation

Pen allows you to replay drawings or text editing in your application.

You can add replay animation features to your application by using `SpenPageDoc.startRecord()`, `SpenPageDoc.stopRecord()`, `SpenSurfaceView.startReplay()`, `SpenSurfaceView.stopReplay()`, and other replay-related methods.

The sample application implements the following features:

1. Replay Button to replay the drawings or text editing.

Recording of drawing from application start with `SpenPageDoc.startRecord()`.

If the Replay button is clicked, `SpenSurfaceView.startReplay()` replays the drawing from the first object. When the replay animation is played, the buttons are disabled.

When the animation is complete, the buttons are enabled again.

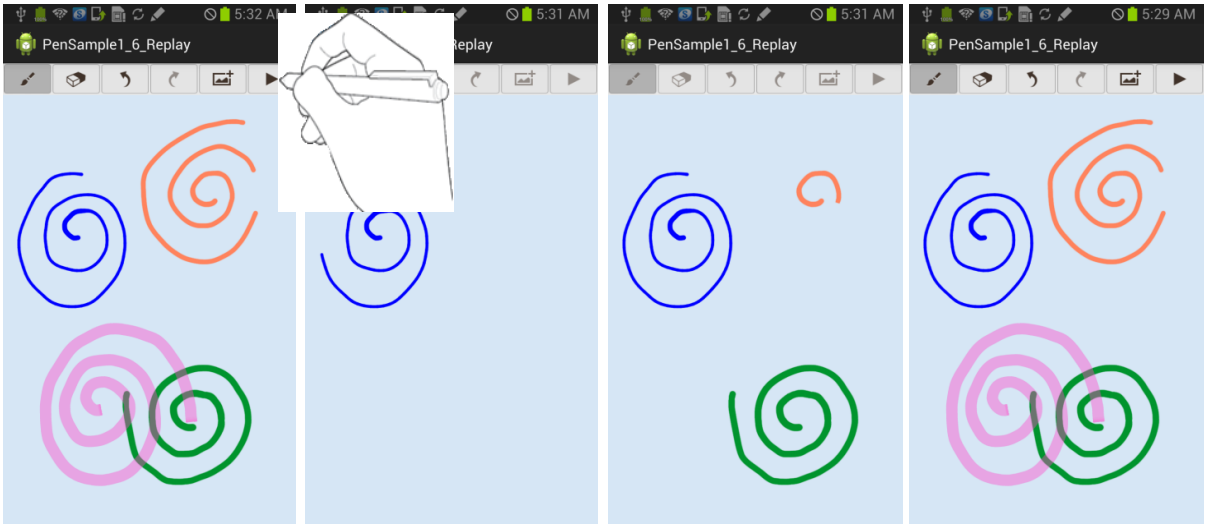


Figure 11: Replay function

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

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

        mSpenSurfaceView.setReplayListener(mReplayListener);

          .........

        mPlayBtn = (ImageView) findViewById(R.id.playBtn);
        mPlayBtn.setOnClickListener(mPlayBtnClickListener);

        selectButton(mPenBtn);

        mSpenPageDoc.startRecord();


          .........

private final OnClickListener mPlayBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                closeSettingView();
                setBtnEnabled(false);
                mSpenSurfaceView.startReplay();
            }
};

          .........

private SpenReplayListener mReplayListener = new SpenReplayListener() {

        @Override
        public void onProgressChanged(int progress, int id) {
        }

        @Override
        public void onCompleted() {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    // Enable the buttons when replay animation is complete.
                    setBtnEnabled(true);
                    mUndoBtn.setEnabled(mSpenPageDoc.isUndoable());
                    mRedoBtn.setEnabled(mSpenPageDoc.isRedoable());
                }
            });
        }
};

          .........

private void setBtnEnabled(boolean clickable) {
        // Enable or Disable all the buttons.
        mPenBtn.setEnabled(clickable);
        mEraserBtn.setEnabled(clickable);
        mUndoBtn.setEnabled(clickable);
        mRedoBtn.setEnabled(clickable);
        mBgImgBtn.setEnabled(clickable);
        mPlayBtn.setEnabled(clickable);
}
```

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

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

        if (mSpenPageDoc.isRecording()) {
            mSpenPageDoc.stopRecord();
        }
        if (mSpenSurfaceView.getReplayState() ==
            SpenSurfaceView.REPLAY_STATE_PLAYING) {
            mSpenSurfaceView.stopReplay();
        }

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

For more information, see PenSample1_6_Replay.java in PenSample1_6_Replay.

The following sections provide more details on the steps involved in replaying drawings.

4.1.6.1 Registering a Listener for the Replay Button

To handle replay button events in your application:

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

In the `onClick()` method, close the settings window (if any is displayed) and disable all the buttons on the viewport.

Replay the animation by calling `SpenSurfaceView.startReplay()`.

```
closeSettingView();
setBtnEnabled(false);
mSpenSurfaceView.startReplay();
```

4.1.6.2 Creating and Registering a Listener for Replay

To handle replay events in your application:

1. Create an `SpenReplayListener` instance and register it by calling `SpenSurfaceViewc.setReplayListener()`.

In the `onCompleted()` method, enable all the buttons. Enable the Undo and Redo buttons depending on their availability.

If necessary, override the object ID and the callback function for the process status, `onProgressChanged()`.

```
public void onCompleted() {
	runOnUiThread(new Runnable() {
		@Override
		public void run() {
			// Enable the buttons on completion of replay animation.
			setBtnEnabled(true);
			mUndoBtn.setEnabled(mSpenPageDoc.isUndoable());
			mRedoBtn.setEnabled(mSpenPageDoc.isRedoable());
		}
	});
}
```

4.1.6.3 Recording Drawing

To record drawing in your application when it starts:

1. Call `SpenPageDoc.startRecord()` in `onCreate()` to start recording drawing on the viewport when your application starts.

If the user clicks the Replay button, all the drawn objects from the first object are replayed.

4.1.6.4 Stopping Recording or Replay

To stop recording when your application closes:

1. If recording is in progress, call `SpenPageDoc.stopRecord()`.
2. If a replay is running, call `SpenSurfaceView.stopReplay()`.

```
if (mSpenPageDoc.isRecording()) {
	mSpenPageDoc.stopRecord();
}
if (mSpenSurfaceView.getReplayState() == SpenSurfaceView.REPLAY_STATE_PLAYING) {
	mSpenSurfaceView.stopReplay();
}
```

Note
You cannot replay the editing of objects with replay animations. Only the final state of each object is redrawn because a replay animation follows the order of the objects listed in SpenPageDoc.

4.1.7. Capturing Screen Shots

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

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

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.

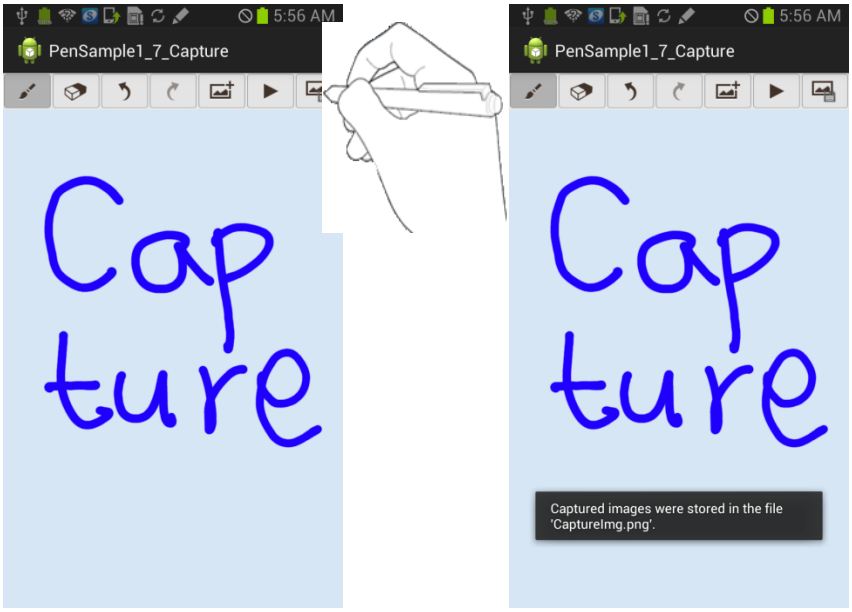


Figure 12: Capture function

```
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();
                captureSpenSurfaceView();
            }
```

```
};

              .........

private void captureSpenSurfaceView() {
// 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 = mSpenSurfaceView.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_7_Capture.java in PenSample1_7_Capture.

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

4.1.7.1 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 `SpenSurfaceView.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 captureSpenSurfaceView() {
        // 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 = mSpenSurfaceView.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 SpenSurfaceView 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 SpenSurfaceView instance with the SpenCapturePage class methods.

4.1.8. Using Custom Drawing

Pen offers you Custom Drawing features for your application.

Use `SpenSurfaceView.setPreDrawListener()` and `SpenView.setPostDrawListener()` to add multiple custom drawings on an SpenSurfaceView instance. This emulates a layered canvas.

The sample application implements the following features:

- Translucent background for its SpenPageDoc instance.
- PreDraw listener to add the checkered pattern.
- User drawing on the SpenSurfaceView instance.
- PostDraw listener to add the translucent text image, “Samsung Mobile SDK S-Pen Package.”

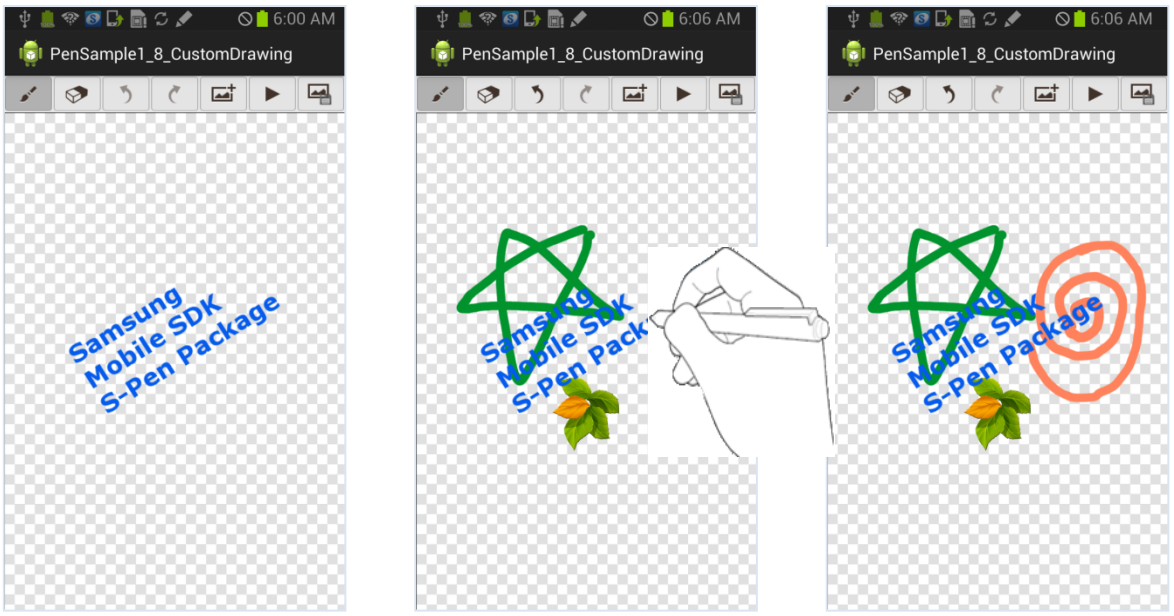


Figure 13: Custom drawing

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

              .........

        mSpenPageDoc.setBackgroundColor(0);
        mSpenPageDoc.clearHistory();

              .........

        mSpenSurfaceView.setPreDrawListener(mPreDrawListener);
        mSpenSurfaceView.setPostDrawListener(mPosteDrawListener);

              .........

}


              .........

private SpenDrawListener mPreDrawListener = new SpenDrawListener() {

        @Override
        public void onDraw(Canvas canvas, float x, float y, float ratio,
            float frameStartX, float frameStartY, RectF updateRect) {
            Bitmap bm =
                ((BitmapDrawable) getResources().getDrawable(
                    R.drawable.canvas_bg)).getBitmap();
            canvas.drawColor(Color.TRANSPARENT, Mode.CLEAR);
            canvas.drawBitmap(bm, 0, 0, null);
        }
};
```

```
private SpenDrawListener mPosteDrawListener = new SpenDrawListener() {

        @Override
        public void onDraw(Canvas canvas, float x, float y, float ratio,
            float frameStartX, float frameStartY, RectF updateRect) {
            Bitmap bm =
                ((BitmapDrawable) getResources().getDrawable(
                    R.drawable.watermark)).getBitmap();
            float pointX = (mScreenRect.width() - bm.getWidth()) / 2;
            float pointY = mScreenRect.height() / 2 - bm.getHeight();
            canvas.drawBitmap(bm, pointX, pointY, null);
        }
};

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

For more information, see PenSample1_8_CustomDrawing.java in PenSample1_8_CustomDrawing.

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

4.1.8.1 Registering a Listener for PreDraw

To handle PreDraw listener events in your application:

1. Create a `SpenDrawListener` instance that is called before Pen displays drawing data on the SpenSurfaceView instance.

To register the listener, call `SpenSurfaceView.setPreDrawListener()`.

In the `onDraw` method, add your code for handling the event. The sample code adds the checkered image by drawing a resource file to the screen.

4.1.8.2 Registering a Listener for PostDraw

To handle PostDraw listener events in your application:

1. Create a `SpenDrawListener` instance that is called after Pen displays drawing data on the SpenSurfaceView instance.

To register the listener, call `SpenSurfaceView.setPostDrawListener()`.

In the `onDraw` method, add your code for handling the event. The sample code adds the translucent text image, “Samsung Mobile SDK S-Pen Package” by drawing a resource file to the screen.

4.2. Using Pen Documents

A document is a Pen 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, images, and text 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. Adding Image Objects

The sample application implements the following features:

- Add Image button to add an image each time the S pen touches the viewport.
- Custom mode to generate images on the viewport.
- Listener for touch events.
- Image object placement and SpenPageDoc instance and viewport update.

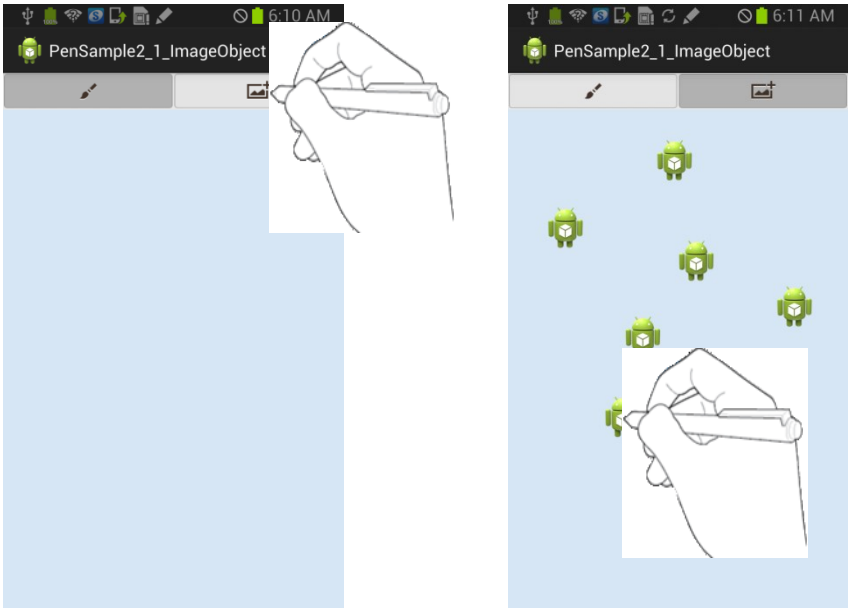


Figure 14: Image object

```
public class PenSample2_1_ImageObject extends Activity {

	private final int MODE_PEN = 0;
	private final int MODE_IMG_OBJ = 1;

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

```
    private SpenSurfaceView mSpenSurfaceView;
    private SpenSettingPenLayout mPenSettingView;

    private ImageView mPenBtn;
    private ImageView mImgObjBtn;

    private int mMode = MODE_PEN;

    private int mToolType = SpenSurfaceView.TOOL_SPEN;

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

        // Initialize Pen.
        boolean isSpenFeatureEnabled = false;
        Spen spenPackage = new Spen();
        try {
            spenPackage.initialize(this);
            isSpenFeatureEnabled =
                spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
        } catch (SsdkUnsupportedException e) {
            if( SDKUtils.processUnsupportedException(this, e) == true) {
                return;
            }
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Pen.",
                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, new String(),
                spenViewLayout);
        if (mPenSettingView == null) {
            Toast.makeText(mContext, "Cannot create new PenSettingView.",
                Toast.LENGTH_SHORT).show();
            finish();
        }
        spenViewContainer.addView(mPenSettingView);

        // Create Pen View.
        mSpenSurfaceView = new SpenSurfaceView(mContext);
        if (mSpenSurfaceView == null) {
            Toast.makeText(mContext, "Cannot create new SpenSurfaceView.",
                Toast.LENGTH_SHORT).show();
            finish();
        }
        spenViewLayout.addView(mSpenSurfaceView);
        mPenSettingView.setCanvasView(mSpenSurfaceView);
```

```
        // Get the dimensions of the screen of the device.
        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();
        }
        // 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.
        mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);

        initPenSettingInfo();
        // Register the listener.
        mSpenSurfaceView.setTouchListener(mPenTouchListener);
        mSpenSurfaceView.setColorPickerListener(mColorPickerListener);

        // Define the button.
        mPenBtn = (ImageView) findViewById(R.id.penBtn);
        mPenBtn.setOnClickListener(mPenBtnClickListener);

        mImgObjBtn = (ImageView) findViewById(R.id.imgObjBtn);
        mImgObjBtn.setOnClickListener(mImgObjBtnClickListener);

        selectButton(mPenBtn);

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

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

```
    private 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 = mSpenSurfaceView.getControl();
                if (control == null) {
                    // A touch event occurs in ObjectImage adding mode.
                    if (mMode == MODE_IMG_OBJ) {
                        // Set Bitmap file for ObjectImage.
                        SpenObjectImage imgObj = new SpenObjectImage();
                        Bitmap imageBitmap =
                            BitmapFactory.decodeResource(
                                mContext.getResources(),
                                R.drawable.ic_launcher);
                        imgObj.setImage(imageBitmap);

                        // Set the position of ObjectImage and add it to PageDoc.
                        float x = event.getX();
                        float y = event.getY();
                        float panX = mSpenSurfaceView.getPan().x;
                        float panY = mSpenSurfaceView.getPan().y;
                        float zoom = mSpenSurfaceView.getZoomRatio();
                        float imgWidth = imageBitmap.getWidth() * zoom;
                        float imgHeight = imageBitmap.getHeight() * zoom;
                        RectF imageRect = new RectF();
                        imageRect.set((x - imgWidth / 2) / zoom + panX,
                            (y - imgHeight / 2) / zoom + panY,
                            (x + imgWidth / 2) / zoom + panX,
                            (y + imgHeight / 2) / zoom + panY);
                        imgObj.setRect(imageRect, true);
                        mSpenPageDoc.appendObject(imgObj);
                        mSpenSurfaceView.update();

                        imageBitmap.recycle();
                        return true;
                    }
                }
            }
            return false;
        }
    };

    private final OnClickListener mPenBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                // If it is in pen tool mode.
                if (mSpenSurfaceView.getToolTypeAction(mToolType) ==
                        SpenSurfaceView.ACTION_STROKE) {
                    // If PenSettingView is displayed, close it.
                    if (mPenSettingView.isShown()) {
                        mPenSettingView.setVisibility(View.GONE);
                    // If PenSettingView is not displayed, display it.
                    } else {
                        mPenSettingView
```

```
                            .setViewMode(SpenSettingPenLayout.VIEW_MODE_EXTENSION);
                        mPenSettingView.setVisibility(View.VISIBLE);
                    }
                // If it is not in pen tool mode, change it to pen tool mode.
                } else {
                    mMode = MODE_PEN;
                    selectButton(mPenBtn);
                    mSpenSurfaceView.setToolTypeAction(
                        mToolType,
                        SpenSurfaceView.ACTION_STROKE);
                }
            }
        };

    private final OnClickListener mImgObjBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                mMode = MODE_IMG_OBJ;
                selectButton(mImgObjBtn);
                mSpenSurfaceView.setToolTypeAction(
                    mToolType,
                    SpenSurfaceView.ACTION_NONE);
            }
        };

    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);
                }
            }
        };

    private void selectButton(View v) {
        // Close PenSettingView.
        mPenSettingView.setVisibility(SpenSurfaceView.GONE);

        // Depending on the current mode, enable or disable the button.
        mPenBtn.setSelected(false);
        mImgObjBtn.setSelected(false);

        v.setSelected(true);
    }

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

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

        if(mSpenSurfaceView != null) {
```

```
                mSpenSurfaceView.close();
                mSpenSurfaceView = null;
            }

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

For more information, see PenSample2_1_ImageObject.java in PenSample2_1_ImageObject.

The following sections provide more details on the steps involved in adding an image object to the screen.

4.2.1.1 Registering a Listener for the Add Image Button

To handle Add Image button events in your application:

1. Create an Add Image button.

Create an `OnClickListener` instance for the Add Image button, mImgObjBtnClickListener in the sample, and register it by calling `setOnClickListener()` on the button.

In the `onClick()` method for the Add Image button listener, set `mToolType` to ACTION_NONE, switch to an internal mode for adding objects, and indicate that the button is selected.

```
mMode = MODE_IMG_OBJ;
selectButton(mImgObjBtn);
mSpenSurfaceView.setToolTypeAction(SpenSurfaceView.mToolType,
                    SpenSurfaceView.ACTION_NONE);
```

4.2.1.2 Creating and Registering a Touch Event Listener

To handle touch events in your application:

1. Create an `SpenTouchListener` instance to listen for touch events, mPenTouchListener in the sample.

Add the `onTouch()` callback method to handle touch events in the viewport.

Register the listener by calling `SpenSurfaceView.setTouchListener()`.

In the `onTouch()` method, if the SpenSurfaceView tool type is S pen and the application internal action mode is Add Image, create an image object.

Call `SpenObjectImage.setImage()` to assign your Bitmap object. The sample application users an Android icon.

Check if the viewport can be zoomed in, zoomed out, or panned, and set the position of the image accordingly by calling `SpenObjectImage.setRect()`.

Call `SpenPageDoc.appendObject()` to add the image object to the viewport.

To refresh the viewport, call `SpenSurfaceView.update()`.

Call `recycle()`to prevent memory leaks.

```
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 = mSpenSurfaceView.getControl();
		if (control == null) {
			// A S pen touches the screen
			// when it is in Add ObjectImage mode.
			if (mMode == MODE_IMG_OBJ) {
				// Set a Bitmap file for ObjectImage.
				SpenObjectImage imgObj = new SpenObjectImage();
				Bitmap imageBitmap =
					BitmapFactory.decodeResource(
						mContext.getResources(),
						R.drawable.ic_launcher);
				imgObj.setImage(imageBitmap);

				// Set the position and size for an ObjectImage
				// and add it to PageDoc.
				float x = event.getX();
				float y = event.getY();
				float panX = mSpenSurfaceView.getPan().x;
				float panY = mSpenSurfaceView.getPan().y;
				float zoom = mSpenSurfaceView.getZoomRatio();
				float imgWidth = imageBitmap.getWidth() * zoom;
				float imgHeight = imageBitmap.getHeight() * zoom;
				RectF imageRect = new RectF();
				imageRect.set((x - imgWidth / 2) / zoom + panX,
						(y - imgHeight / 2) / zoom + panY,
						(x + imgWidth / 2) / zoom + panX,
						(y + imgHeight / 2) / zoom + panY);
				imgObj.setRect(imageRect, true);
				mSpenPageDoc.appendObject(imgObj);
				mSpenSurfaceView.update();

				imageBitmap.recycle();
				return true;
			}
		}
	}
	return false;
}
```

Note

Before you add an object to an SpenPageDoc instance, set the second input variable to true to select the area for the object. For example, `SpenObjectBase.setRect(rect, true)`.

In Pen, you can also use setRect() for moving and scaling objects. In that case, set the second input variable to false. To move an object by (dx, dy), see the following sample code.

```
RectF rect = object.getRect();
rect.left += dx;
rect.top += dy;
rect.right += dx;
rect.bottom += dy;
object.setRect(rect, false);
```

4.2.2. Inserting Text Objects

The sample application implements the following features:

- Insert Text button to add text boxes each time the pen touches the viewport.
- Custom mode for inserting a text box when there is a touch event.
- Listener for touch events in the View area.
- Text box object insertion and SpenPageDoc instance and viewport update. If another text box already exists at the coordinates where the touch event takes place, the SpenSurfaceView instance switches to text editing mode.

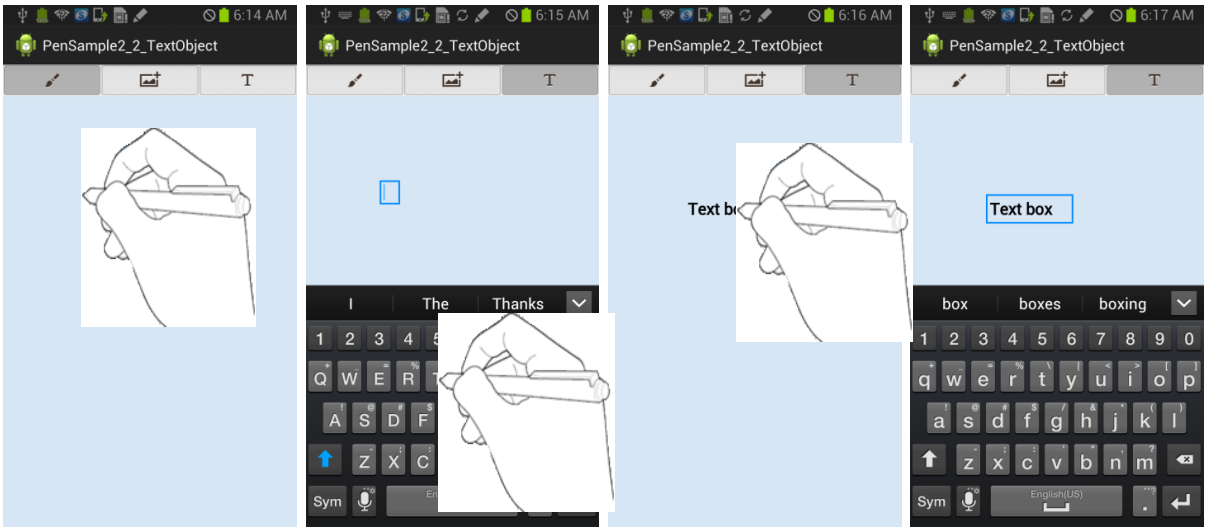


Figure 15: Text object

If you click the Insert Text button and the action for TOOL_SPEN is ACTION_TEXT, the text property settings window appears to allow you to change the font size, color, font and type. You can close the window by clicking the Insert Text button again when the text property window is open.

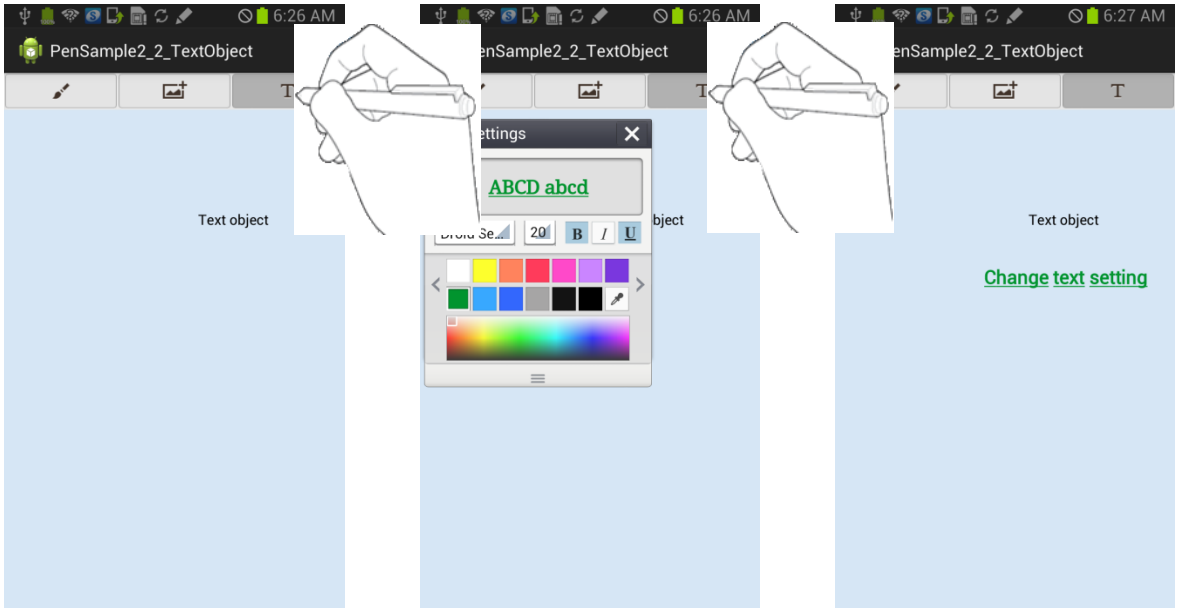


Figure 16: Text settings

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

              .........

        // Create TextSettingView.
        mTextSettingView =
            new SpenSettingTextLayout(mContext, new String(),
                new HashMap<String, String>(), spenViewLayout);
        if (mTextSettingView == null) {
            Toast.makeText(mContext, "Cannot create new TextSettingView.",
                Toast.LENGTH_SHORT).show();
            finish();
        }
        spenViewContainer.addView(mPenSettingView);
        spenViewContainer.addView(mTextSettingView);

              .........

        initSettingInfo();
        // Register the listeners.
        mSpenSurfaceView.setTouchListener(mPenTouchListener);
        mSpenSurfaceView.setColorPickerListener(mColorPickerListener);
        mSpenSurfaceView.setTextChangeListener(mTextChangeListener);

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

```
        mTextObjBtn = (ImageView) findViewById(R.id.textObjBtn);
        mTextObjBtn.setOnClickListener(mTextObjBtnClickListener);

        selectButton(mPenBtn);
}

private void initSettingInfo() {
        // Reset the settings for the pen.
        SpenSettingPenInfo penInfo = new SpenSettingPenInfo();
        penInfo.color = Color.BLUE;
        penInfo.size = 10;
        mSpenSurfaceView.setPenSettingInfo(penInfo);
        mPenSettingView.setInfo(penInfo);

        // Reset the settings for text.
        SpenSettingTextInfo textInfo = new SpenSettingTextInfo();
        textInfo.size = 30;
        mSpenSurfaceView.setTextSettingInfo(textInfo);
        mTextSettingView.setInfo(textInfo);
}

private SpenTouchListener mPenTouchListener = new SpenTouchListener() {

        @Override
        public boolean onTouch(View view, MotionEvent event) {
            if (event.getAction() == MotionEvent.ACTION_UP
                && event.getToolType(0) == mToolType) {
                // Checks whether the control is generated or not.
                SpenControlBase control = mSpenSurfaceView.getControl();
                if (control == null) {
                    // When touching the screen in Insert ObjectImage mode.
                    if (mMode == MODE_IMG_OBJ) {
                        // Set the Bitmap file to ObjectImage.

                        SpenObjectImage imgObj = new SpenObjectImage();
                        Bitmap imageBitmap =
                            BitmapFactory.decodeResource(
                                mContext.getResources(),
                                R.drawable.ic_launcher);
                        imgObj.setImage(imageBitmap);

                        // Specify the location where ObjectImage is inserted and add
                        // PageDoc.

                        float imgWidth = imageBitmap.getWidth();
                        float imgHeight = imageBitmap.getHeight();
                        RectF rect = getRealPoint(event.getX(), event.getY(),
                                imgWidth, imgHeight);
                        imgObj.setRect(rect, true);
                        mSpenPageDoc.appendObject(imgObj);
                        mSpenSurfaceView.update();

                        imageBitmap.recycle();
                        return true;
                    // When touching the screen in Insert ObjectTextBox mode.


                    } else if (mSpenSurfaceView.
```

```
                                getToolTypeAction(mToolType)
                        == SpenSurfaceView.ACTION_TEXT) {
                        // Specify the location where ObjectTextBox is inserted and add
PageDoc.

                        SpenObjectTextBox textObj = new SpenObjectTextBox();
                        RectF rect = getRealPoint(event.getX(), event.getY(),
                                0, 0);
                        rect.right += 200;
                        rect.bottom += 50;
                        textObj.setRect(rect, true);
                        mSpenPageDoc.appendObject(textObj);
                        mSpenPageDoc.selectObject(textObj);
                        mSpenSurfaceView.update();
                    }
                }
            }
            return false;
        }
};

private RectF getRealPoint(float x, float y, float width, float height) {
        float panX = mSpenSurfaceView.getPan().x;
        float panY = mSpenSurfaceView.getPan().y;
        float zoom = mSpenSurfaceView.getZoomRatio();
        width *= zoom;
        height *= zoom;
        RectF realRect = new RectF();
        realRect.set(
            (x - width / 2) / zoom + panX, (y - height / 2) / zoom + panY,
            (x + width / 2) / zoom + panX, (y + height / 2) / zoom + panY);
        return realRect;
}

              .........

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

                // When Pen is in text mode.
                if (mSpenSurfaceView.getToolTypeAction(mToolType) ==
                        SpenSurfaceView.ACTION_TEXT) {
                    // Close TextSettingView if TextSettingView is displayed.
                    if (mTextSettingView.isShown()) {
                        mTextSettingView.setVisibility(View.GONE);
                    // Display TextSettingView if TextSettingView is not displayed.
                    } else {
                        mTextSettingView
                            .setViewMode(SpenSettingTextLayout.VIEW_MODE_NORMAL);
                        mTextSettingView.setVisibility(View.VISIBLE);
                    }
                // Switch to text mode unless Pen is in text mode.
                } else {
                    mMode = MODE_TEXT_OBJ;
                    selectButton(mTextObjBtn);
                    mSpenSurfaceView.setToolTypeAction(mToolType,
```

```
                        SpenSurfaceView.ACTION_TEXT);
                }
            }
};

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

                    SpenSettingTextInfo textInfo =
                        mTextSettingView.getInfo();
                    textInfo.color = color;
                    mTextSettingView.setInfo(textInfo);
                }
            }
};

SpenTextChangeListener mTextChangeListener = new SpenTextChangeListener() {

        @Override
        public boolean onSelectionChanged(int arg0, int arg1) {
            return false;
        }

        @Override
        public void onMoreButtonDown(SpenObjectTextBox arg0) {
        }

        @Override
        public void onChanged(SpenSettingTextInfo info, int state) {
            if (mTextSettingView != null) {
                if (state == CONTROL_STATE_SELECTED) {
                    mTextSettingView.setInfo(info);
                }
            }
        }
        @Override
        public void onFocusChanged(boolean arg0) {

        }
};

private void selectButton(View v) {
        // Enable or disable buttons depending on the mode.
        mPenBtn.setSelected(false);
        mImgObjBtn.setSelected(false);
        mTextObjBtn.setSelected(false);

        v.setSelected(true);

        closeSettingView();
}
```

```
private void closeSettingView() {
        // Close all setting views.
        mPenSettingView.setVisibility(SpenSurfaceView.GONE);
        mTextSettingView.setVisibility(SpenSurfaceView.GONE);
}

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

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

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

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

For more information, see PenSample2_2_TextObject.java in PenSample2_2_TextObject.

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

4.2.2.1 Registering a Listener for the Insert Text Button

To handle Insert Text button events in your application:

1. Create an Insert Text Box button.

Create an `OnClickListener` instance for the Insert Text button, mTextObjBtnClickListener in the sample, and register it by calling `setOnClickListener()` on the button.

In the `onClick()` method, show the Text Properties view if `mToolType` is set to ACTION_TEXT. If `mToolType` is not set to ACTION_TEXT, switch to an internal mode for adding text box objects. Indicate that the button has been selected.

Call `SpenSurfaceView.setViewMode()` to show the Text Settings view. Use VIEW_MODE_NORMAL in `setViewMode` to show the font setting tab and paragraph tab.

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

            // When Pen is in text mode.
            if (mSpenSurfaceView.getToolTypeAction(mToolType) ==
                    SpenSurfaceView.ACTION_TEXT) {
                // Close PenSettingView if TextSettingView is displayed.
                if (mTextSettingView.isShown()) {
                    mTextSettingView.setVisibility(View.GONE);
                // Display PenSettingView unless TextSettingView is displayed.
                } else {
                    mTextSettingView
                            .setViewMode(SpenSettingTextLayout.VIEW_MODE_NORMAL);
                    mTextSettingView.setVisibility(View.VISIBLE);
                }
            // Switch to text mode unless Pen is in text mode.
            } else {
                mMode = MODE_TEXT_OBJ;
                selectButton(mTextObjBtn);
                mSpenSurfaceView.setToolTypeAction(mToolType,
                        SpenSurfaceView.ACTION_TEXT);
            }
}
```

Note

SpenSettingTextLayout offers you the following text view modes:

View mode	Value	Description
VIEW_MODE_NORMAL	0	To set fonts and paragraphs.
VIEW_MODE_MINIMUM	1	To set fonts.
VIEW_MODE_MINIMUM_WITHOUT_PREVIEW	2	To set fonts without displaying a preview that shows a sample of a selected font.
VIEW_MODE_STYLE	3	To set font styles such as bold, italic, and underline.
VIEW_MODE_COLOR	4	To set the font color.
VIEW_MODE_PARAGRAPH	5	To set alignment, indentation and line spacing.

4.2.2.2 Creating and Registering a Touch Event Listener

To handle touch events in your application when you are in text box mode:

1. Create an `SpenTouchListener` listener instance and implement the `onTouch()` callback function to handle touch events in the viewport.

Call SpenSurfaceView.setTouchListener() to register the listener.

If another text box already exists at the coordinates where the touch event takes place, SpenSurfaceView switches to text editing mode.

In the onTouch() method, add a text box object if the tool type is TOOL_SPEN and the action for the tool type is ACTION_TEXT.

Calculate the location of the text box considering zooming and rotating of the screen and call SpenObjectTextBox.setRect() to specify the location for inserting the text box.

Call SpenPageDoc.appendObject() to insert the text box object.

Call SpenPageDoc.selectObject() to select the new text box.

Call SpenSurfaceView.update() to refresh the screen.

```
public boolean onTouch(View view, MotionEvent event) {
            if (event.getAction() == MotionEvent.ACTION_UP
                && event.getToolType(0) == mToolType) {
                // Check whether the control is created or not.
                SpenControlBase control = mSpenSurfaceView.getControl();
                if (control == null) {

              .........

                    // When touching the screen in Insert ObjectTextBox mode.
                    } else if (mSpenSurfaceView.
                              getToolTypeAction(mToolType)
                        == SpenSurfaceView.ACTION_TEXT) {
                        // Specify the location where ObjectTextBox is inserted
                        // and add PageDoc.
                        SpenObjectTextBox textObj = new SpenObjectTextBox();
                        RectF rect = getRealPoint(event.getX(), event.getY(),
                                0, 0);
                        rect.right += 200;
                        rect.bottom += 50;
                        textObj.setRect(rect, true);
                        mSpenPageDoc.appendObject(textObj);
                        mSpenPageDoc.selectObject(textObj);
                        mSpenSurfaceView.update();
                    }
                }
            }
            return false;
        }
};

private RectF getRealPoint(float x, float y, float width, float height) {
        float panX = mSpenSurfaceView.getPan().x;
        float panY = mSpenSurfaceView.getPan().y;
        float zoom = mSpenSurfaceView.getZoomRatio();
        width *= zoom;
        height *= zoom;
        RectF realRect = new RectF();
        realRect.set(
```

```
            (x - width / 2) / zoom + panX, (y - height / 2) / zoom + panY,
            (x + width / 2) / zoom + panX, (y + height / 2) / zoom + panY);
        return realRect;
}
```

Note

When adding an ObjectTextBox instance, use `SpenObjectTextBox.setGravity()` to align the text horizontally or vertically. Pen supports the following gravity options:

Gravity	Value	Description
GRAVITY_TOP	0	Align text to the top edge of the text box.
GRAVITY_CENTER	1	Align text to the vertical and horizontal center of the text box.
GRAVITY_BOTTOM	2	Align text to the bottom edge of the text box.

Use `SpenObjectTextBox.setAutoFitOption()` to set the Rect Auto Fit type of Spentext box objects. Pen supports the following Auto Fit options:

AUTO_FIT_NONE	0	No auto-fit.
AUTO_FIT_HORIZONTAL	1	Automatically adjusts the horizontal size of the object Rect to accommodate the text.
AUTO_FIT_VERTICAL	2	Automatically adjusts the vertical size of the object Rect to accommodate the text.
AUTO_FIT_BOTH	3	Automatically adjusts the horizontal and vertical size of the object Rect to accommodate the text.

4.2.2.3 Adding a Color Picker Listener

To add a color picker for text in your application:

1. Create an `SpenColorPickerListener` listener instance, mColorPickerListener in the sample, for the color picker and handle the events it returns in the `onClick()` method.

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

Call `SpenSettingTextInfo.SetInfo()` to set the color returned by the color picker.

```
SpenSettingTextInfo textInfo = mTextSettingView.getInfo();
textInfo.color = color;
mTextSettingView.setInfo(textInfo);
```

4.2.2.4 Adding a Text Change Event Listener

To add a listener for text change events:

1. Create an SpenTextChangeListener listener instance, mTextChangeListener in the sample, to receive text settings change events.

Call `SpenSurfaceView.setTextChangeListener()` to register the listener.

If the SpenSettingPenLayout window is open when the cursor is placed in the SpenTextBox, call `SpenSettingTextInfo.SetInfo()` to add the settings of the text at the cursor to the SpenSettingPenLayout window.

```
SpenTextChangeListener mTextChangeListener = new SpenTextChangeListener() {
        @Override
        public void onChanged(SpenSettingTextInfo info, int state) {
            if (mTextSettingView != null) {
                if (state == CONTROL_STATE_SELECTED) {
                    mTextSettingView.setInfo(info);
                }
            }
        }
};
```

4.2.2.5 Preventing Memory Leaks

To prevent memory leaks:

1. Call `SpenSettingTextLayout.close()` in the `onDestroy()` method to prevent memory leaks when you application closes.

2. Call `SpenSurfaceView.closeControl()` to close the text control.

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

mSpenSurfaceView.closeControl();
```

Note
To set the basic properties of your ObjectTextBox instance, call methods such as `setFont()`, `setFontSize()`, `setTextStyle()`, `setTextColor()`, and `setBackgroundColor()`.Pen also provides classes for setting paragraphs and spans, and supports rich text format. The following classes inherit TextParagraphInfo for setting text paragraph properties:

Class	Description

Note

StyleParagraphInfo	Style paragraphs with templates.
IndentLevelParagraphInfo	Set the indentation of a paragraph.
AlignParagraphInfo	Set the alignment of a paragraph.
LineSpacingParagraphInfo	Set the line spacing of a paragraph.

The following classes inherit TextSpanInfo for setting text paragraph properties:

Class	Description
TextDirectionSpanInfo	Set the text direction.
FontSizeSpanInfo	Set the font size.
FontNameSpanInfo	Set text font.
ForegroundColorSpanInfo	Set text color.
BackgroundColorSpanInfo	Set text background color.
BoldStyleSpanInfo	Make text bold.
ItalicStyleSpanInfo	Make text italic.
UnderlineStyleSpanInfo	Underline text.

Note

The following simple sample code uses TextSpanInfo and TextParagraphInfo:

```
String string =
new String("This is rich text format\nHow does it look?");
SpenObjectTextBox text = new SpenObjectTextBox();
text.setText(string);

ArrayList<SpenObjectTextBox.TextParagraphInfo> paras =
                new ArrayList<TextParagraphInfo>();
SpenObjectTextBox.AlignParagraphInfo alignInfo =
                new SpenObjectTextBox.AlignParagraphInfo();
alignInfo.startPos = 26;
alignInfo.endPos = 41;
alignInfo.align = SpenObjectTextBox.ALIGN_RIGHT;
paras.add(alignInfo);

text.setParagraph(paras);

ArrayList<SpenObjectTextBox.TextSpanInfo> spans =
                new ArrayList<SpenObjectTextBox.TextSpanInfo>();
SpenObjectTextBox.FontSizeSpanInfo fontSize =
```

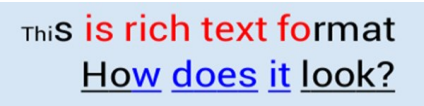
Note

```
                new SpenObjectTextBox.FontSizeSpanInfo();
fontSize.startPos = 0;
fontSize.endPos = 3;
fontSize.fontSize = 20;
spans.add(fontSize);

SpenObjectTextBox.UnderlineStyleSpanInfo underLine =
                new SpenObjectTextBox.UnderlineStyleSpanInfo();
underLine.startPos = 25;
underLine.endPos = text.getText().length();
underLine.isUnderline = true;
spans.add(underLine);

SpenObjectTextBox.ForegroundColorSpanInfo[] fontColor =
                { new SpenObjectTextBox.ForegroundColorSpanInfo(),
                new SpenObjectTextBox.ForegroundColorSpanInfo() };
fontColor[0].foregroundColor = Color.RED;
fontColor[0].startPos = 4;
fontColor[0].endPos = 20;
spans.add(fontColor[0]);
fontColor[1].foregroundColor = Color.BLUE;
fontColor[1].startPos = 27;
fontColor[1].endPos = 37;
spans.add(fontColor[1]);

text.setSpan(spans);
```



4.2.3. Inserting Stroke Objects

Pen 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.

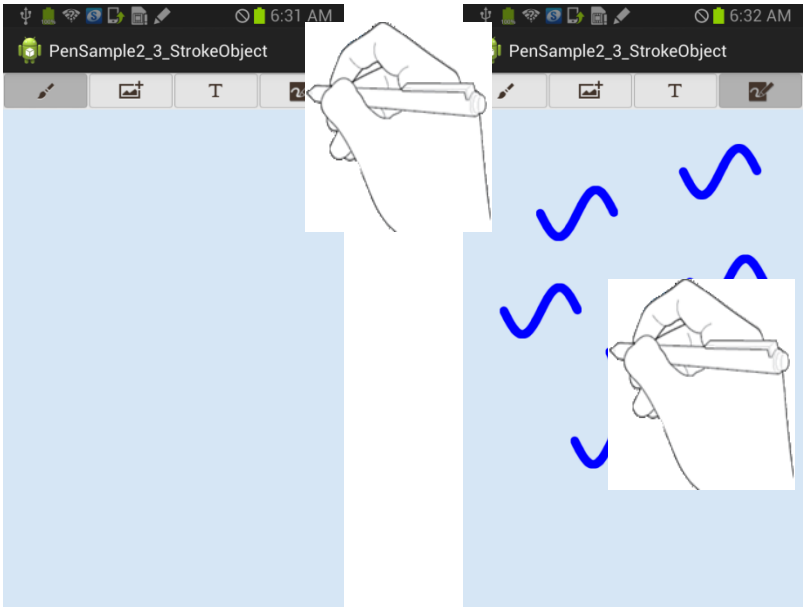


Figure 17: Stroke object

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

              .........

        mSpenSurfaceView.setTouchListener(mPenTouchListener);

              .........

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

        selectButton(mPenBtn);
}

              .........

private SpenTouchListener mPenTouchListener = new SpenTouchListener() {

        @Override
        public boolean onTouch(View view, MotionEvent event) {
            if (event.getAction() == MotionEvent.ACTION_UP
                && event.getToolType(0) == mToolType) {
                // Check whether the control is created or not.
                SpenControlBase control = mSpenSurfaceView.getControl();
                if (control == null) {

              .........

                    //If in Insert Stroke Object mode when a touch event occurs.

                    } else if (mMode == MODE_STROKE_OBJ) {
```

```
                        //Specify the location where ObjectStroke is inserted
                        // and add PageDoc.
                        RectF rect = getRealPoint(event.getX(), event.getY(),
                            0, 0);
                        float rectX = rect.centerX();
                        float rectY = rect.centerY();
                        int pointSize = 157;
                        float[][] strokePoint = new float[pointSize][2];
                        for(int i = 0; i < pointSize; i++) {
                            strokePoint[i][0] = rectX++;
                            strokePoint[i][1] =
                                (float) (rectY + Math.sin(.04 * i) * 50);
                        }
                        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 = strokePoint[i][0];
                            points[i].y = strokePoint[i][1];
                            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);
                        mSpenSurfaceView.update();
                    }
                }
            }
            return false;
        }
};

              .........

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

                mMode = MODE_STROKE_OBJ;
                selectButton(mStrokeObjBtn);
                mSpenSurfaceView.setToolTypeAction(mToolType,
                    SpenSurfaceView.ACTION_NONE);
            }
};
```

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

private void selectButton(View v) {
        // Enable or disable the buttons depending on the mode.
        mPenBtn.setSelected(false);
        mImgObjBtn.setSelected(false);
        mTextObjBtn.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.

4.2.3.1 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);
mSpenSurfaceView.setToolTypeAction(mToolType,
                    SpenSurfaceView.ACTION_NONE);
```

4.2.3.2 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 `SpenSurfaceView.setTouchListener()` to register the listener.

In the `onTouch ()` method, if the SpenSurfaceView 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 `SpenSurfaceView.update()` to refresh the screen.

```
if (mMode == MODE_STROKE_OBJ) {
              //Specify the location where ObjectStroke is inserted
              // and add PageDoc.
              RectF rect = getRealPoint(event.getX(), event.getY(),0, 0);
              float rectX = rect.centerX();
              float rectY = rect.centerY();
              int pointSize = 157;
              float[][] strokePoint = new float[pointSize][2];
              for(int i = 0; i < pointSize; i++) {
                  strokePoint[i][0] = rectX++;
                  strokePoint[i][1] = (float) (rectY + Math.sin(.04 * i) * 50);
              }
              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 = strokePoint[i][0];
                  points[i].y = strokePoint[i][1];
                  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);
              mSpenSurfaceView.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 `SpenSurfaceView.setPreTouchListener()` method to receive and consume pre-touch events.

4.2.4. Saving Files

The sample application saves the data created with Pen in a file. The application supports the SPD format for Pen 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 can read them in the SPD data format. When the image data is modified with common editing tools, Pen 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 PNG, a Bitmap is generated first and then it is saved. The SPD data behind the image is then added.

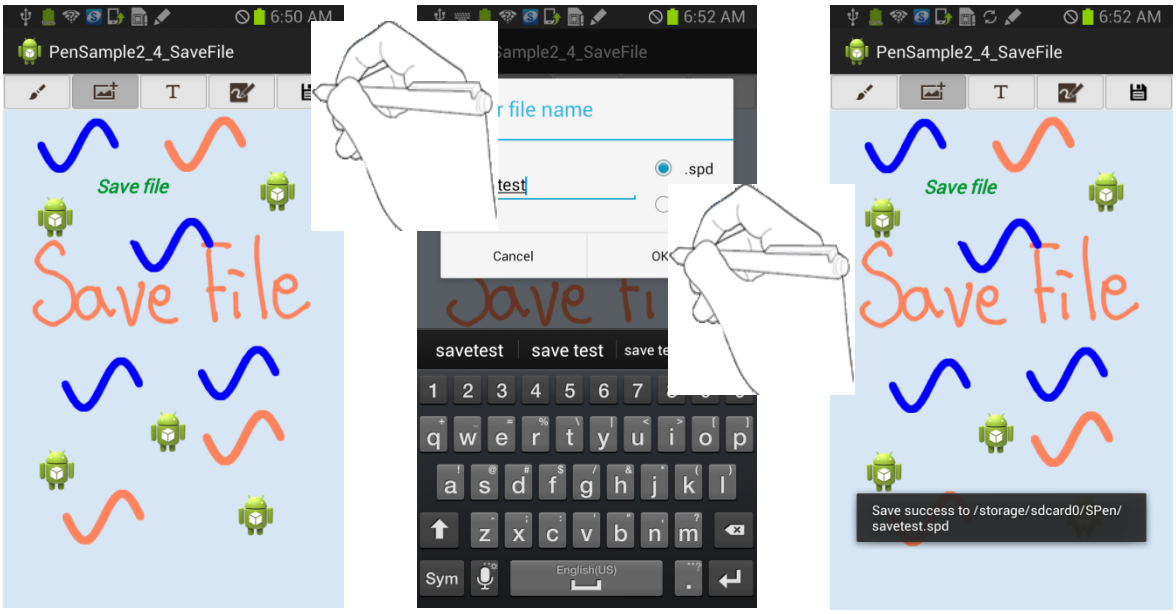


Figure 18: File save function

```
public class PenSample2_4_SaveFile extends Activity {
```

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

    private boolean isDiscard = false;

    @Override
    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) {
                mSpenSurfaceView.closeControl();

                closeSettingView();
                saveNoteFile(false);
            }
        };

    private boolean saveNoteFile(final boolean isClose) {
        // Display the File Save dialog to prompt users to enter file names
        // and let users select the file format: either 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);
```

```
                // Specify the path to the location where the files are saved.

                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;
                    switch (selectFileExt.getCheckedRadioButtonId()) {
                    // Save in note file format.
                    case R.id.radioNote:
                        saveFilePath += ".spd";
                        saveNoteFile(saveFilePath);

                        break;

                    // Save the file in image format.
                    case R.id.radioImage:
                        saveFilePath += ".png";
                        captureSpenSurfaceView(saveFilePath);

                        break;
                    }                       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 the NoteDoc.
```

```
            mSpenNoteDoc.save(strFileName);
            Toast.makeText(mContext, "Saved 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 captureSpenSurfaceView(String strFileName) {

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

        OutputStream out = null;
        try {
            // Create FileOutputStream and capture the image.
            out = new FileOutputStream(strFileName);
            imgBitmap.compress(CompressFormat.PNG, 100, out);
            // Save information about the note.
            mSpenNoteDoc.save(out);
            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();
                            isDiscard = true;
                            finish();
                        }
                    })
                .setNegativeButton("Cancel",
                    new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(
                        DialogInterface dialog, int which) {
                        dialog.dismiss();
                    }
                }).show();
            dlg = null;
        }
        else {
            super.onBackPressed();
        }
    }

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

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

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

        if(mSpenNoteDoc != null) {
            try {
                if (isDiscard)
```

```
                    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.

4.2.4.1 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()` on the 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 save a file in SPD format, pass the file name to the `SpenNoteDoc.save()` method. Pen stores the file in the “SPen/” folder in external storage.

```
private boolean saveNoteFile(String strFileName) {
        try {
            // Save NoteDoc
            mSpenNoteDoc.save(strFileName);
        } catch (IOException e) {
        } catch (Exception e) {
        }
        return true;
}
```

To save a file in PNG format:

- Call `SpenSurfaceView.captureCurrentView()` to get the SpenSurfaceView 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 captureSpenSurfaceView(String strFileName) {

        // View capture
        Bitmap imgBitmap = mSpenSurfaceView.captureCurrentView(true);
        if(imgBitmap == null) {
            return;
        }

        OutputStream out = null;
        try {
            // Create FileOutputStream and save the captured image.
            out = new FileOutputStream(strFileName);
            imgBitmap.compress(CompressFormat.PNG, 100, out);
            // Save information about the note.
            mSpenNoteDoc.save(out);
            out.close();
        } catch (IOException e) {
            File tmpFile = new File(strFileName);
            if (tmpFile.exists())
                tmpFile.delete();
            e.printStackTrace();
        } catch (Exception e) {
            File tmpFile = new File(strFileName);
            if (tmpFile.exists())
                tmpFile.delete();
            e.printStackTrace();
        }
        imgBitmap.recycle();
}
```

4.2.4.2 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.

```
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 saving?")
                .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();
                  isDiscard = 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 (isDiscard)
                        mSpenNoteDoc.discard();
                    else
                        mSpenNoteDoc.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
                mSpenNoteDoc = null;
            }
};
```

4.2.5. Loading SPD and +SPD Files

You can use Pen to load files saved in SPD (Pen data files) and +SPD formats (image files with 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.

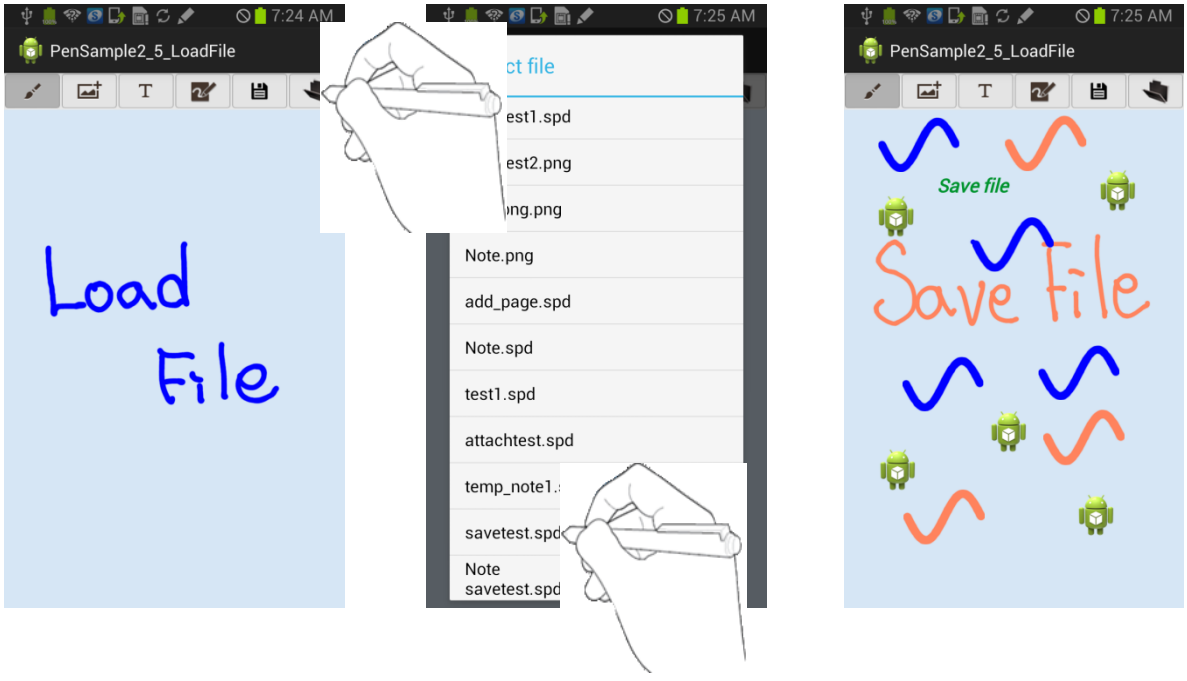


Figure 19: File load function

```
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);

        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) {
                mSpenSurfaceView.closeControl();

                closeSettingView();
                loadNoteFile();
            }
};

              .........

private void loadNoteFile() {
        //File list load.
        final String fileList[] = setFileList();
        if(fileList == null)
            return;

        // Display the file load 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 {
                        // Create NoteDoc with the selected file.
                        SpenNoteDoc tmpSpenNoteDoc = new SpenNoteDoc(mContext,
                            strFilePath, mScreenRect.width(),
                            SpenNoteDoc.MODE_WRITABLE);
                        mSpenNoteDoc.close();
                        mSpenNoteDoc = tmpSpenNoteDoc;
```

```
                        if (mSpenNoteDoc.getPageCount() == 0) {
                            mSpenPageDoc = mSpenNoteDoc.appendPage();
                        } else {
                            mSpenPageDoc =  mSpenNoteDoc.getPage(
                                mSpenNoteDoc.getLastEditedPageIndex());
                        }
                        mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);
                        mSpenSurfaceView.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 a version that is not supported.",
                            Toast.LENGTH_LONG).show();
                    } catch (Exception e) {
                        Toast.makeText(mContext, "Failed to load noteDoc.",
                            Toast.LENGTH_LONG).show();
                    }
                }
            }).show();
}

private String[] setFileList() {
        // Call the list of files located in mFilePath.
        if (!mFilePath.exists()) {
            if (!mFilePath.mkdirs()) {
                Toast.makeText(mContext, "Save Path Creation Error",
                    Toast.LENGTH_SHORT).show();
                return null;
            }
        }
        // Filter by 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.

4.2.5.1 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 settings view and call the file selection view.

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

In the file selection view, create a window to display a list of the SPD 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);
          mSpenNoteDoc.close();
          mSpenNoteDoc = tmpSpenNoteDoc;
          if (mSpenNoteDoc.getPageCount() == 0) {
                  mSpenPageDoc = mSpenNoteDoc.appendPage();
          } else {
                  mSpenPageDoc = mSpenNoteDoc.getPage(
                                  mSpenNoteDoc.getLastEditedPageIndex());
          }
          mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);
          mSpenSurfaceView.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 `SpenSurfaceView.setPageDoc()` to link the page information and your SpenSurfaceView instance.

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

- SpenUnsupportedTypeException is thrown if Pen 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 does not support the SPD file format version.

4.2.6. Attaching External Files

You can use Pen to attach external files to SpenNoteDoc to make them available in your applications.

The sample application implements the following features:

- Insert Image button for adding images. Section 4.3.1. Adding Image Objects also uses the image that the application attaches in this sample. When the Insert Image button is clicked, a dialog appears to ask whether you want to attach a file. When you select Yes, it displays the information that a large attachment may take a long time to insert.
- Displays a list of selectable image files.
- When the Insert Image button is selected, the application creates an image object with the attached data when you touch anywhere in the View area with your pen, inserts it in SpenPageDoc and refreshes the screen.

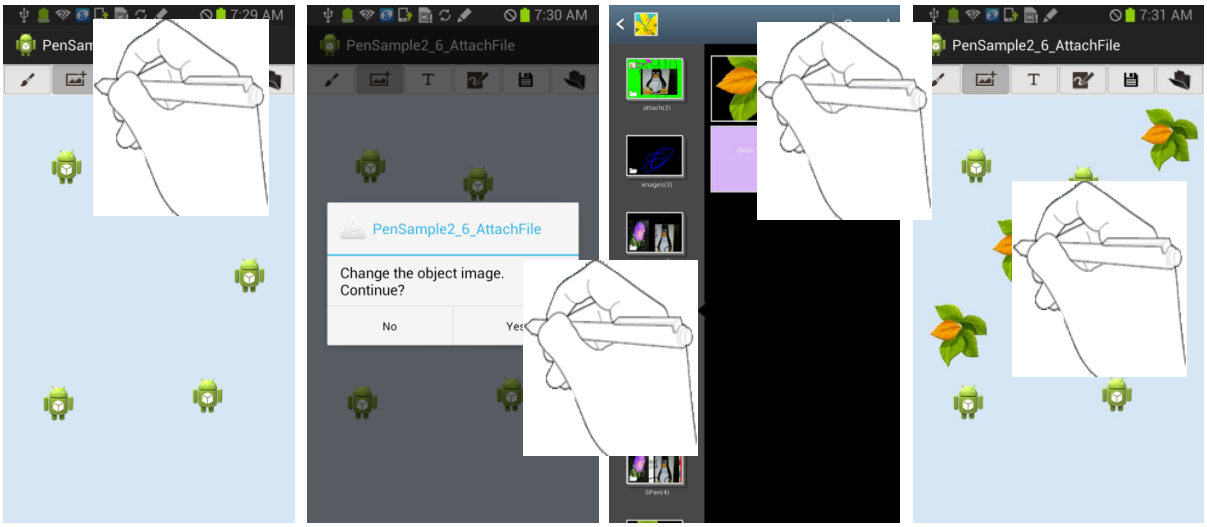


Figure 20: File attach function

```
public class PenSample2_6_AttachFile extends Activity {

    private final int REQUEST_CODE_ATTACH_IMAGE = 100;

    private final String ATTACH_IMAGE_KEY = "Attach Image Key";

              .........

    private SpenTouchListener mPenTouchListener = new SpenTouchListener() {

        @Override
        public boolean onTouch(View view, MotionEvent event) {
            if (event.getAction() == MotionEvent.ACTION_UP
                && event.getToolType(0) == mToolType) {
                // Check whether the control is created or not.
                SpenControlBase control = mSpenSurfaceView.getControl();
                if (control == null) {
                    // When touching the screen in Insert ObjectImage mode.
                    if (mMode == MODE_IMG_OBJ) {
                        SpenObjectImage imgObj = new SpenObjectImage();
                        Bitmap imageBitmap;
                        // Set the Bitmap file to ObjectImage.
                        // If there is any attached file, use this file as
                        // ObjectImage.

                        if (mSpenNoteDoc.hasAttachedFile(ATTACH_IMAGE_KEY)) {
                            imageBitmap =
                                BitmapFactory.decodeFile(mSpenNoteDoc
                                    .getAttachedFile(ATTACH_IMAGE_KEY));
                        // If there is no attached file, use the launcher icon as
                        // ObjectImage.

                        } else {
                            imageBitmap =
                                BitmapFactory.decodeResource(
                                    mContext.getResources(),
                                    R.drawable.ic_launcher);
```

```
                    }
                    imgObj.setImage(imageBitmap);

                    // Specify the location where ObjectImage is inserted and add
                    // PageDoc.
                    float imgWidth = imageBitmap.getWidth();
                    float imgHeight = imageBitmap.getHeight();
                    RectF rect = getRealPoint(event.getX(), event.getY(),
                            imgWidth, imgHeight);
                    imgObj.setRect(rect, true);
                    mSpenPageDoc.appendObject(imgObj);
                    mSpenSurfaceView.update();

                    imageBitmap.recycle();
                    return true;

            .........

    };

    private RectF
        getRealPoint(float x, float y, float width, float height) {
        float panX = mSpenSurfaceView.getPan().x;
        float panY = mSpenSurfaceView.getPan().y;
        float zoom = mSpenSurfaceView.getZoomRatio();
        width *= zoom;
        height *= zoom;
        RectF realRect = new RectF();
        realRect.set(
            (x - width / 2) / zoom + panX, (y - height / 2) / zoom + panY,
            (x + width / 2) / zoom + panX, (y + height / 2) / zoom + panY);
        return realRect;
    }

            .........

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

                if (mMode == MODE_IMG_OBJ) {
                    closeSettingView();
                    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("Change the object image. Continue?")
                        .setPositiveButton("Yes",
                            new DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(
                                    DialogInterface dialog, int which) {

                                    changeImgObj();
                                    // Finish the dialog.
                                    dialog.dismiss();
                                }
```

```
                        })
                    .setNegativeButton("No",
                        new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(
                                DialogInterface dialog, int which) {
                                dialog.dismiss();
                            }
                        }).show();
                dlg = null;
            } else {
                mMode = MODE_IMG_OBJ;
                selectButton(mImgObjBtn);
                mSpenSurfaceView.setToolTypeAction(mToolType,
                    SpenSurfaceView.ACTION_NONE);
            }
        }
};

          .........

private void changeImgObj() {
    // Print the warning message.
    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(
            "When you select an image, copy the image in NoteDoc data. \n" +
            "If the image is large," +
            " it can take a long time to save/load.")
        .setPositiveButton("OK",
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(
                    DialogInterface dialog, int which) {

                    callGalleryForInputImage(REQUEST_CODE_ATTACH_IMAGE);
                    // Finish the dialog.
                    dialog.dismiss();
                }
            }).show();
    dlg = null;
}

private void callGalleryForInputImage(int nRequestCode) {
    // Get the 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 activity.",
            Toast.LENGTH_SHORT).show();
        e.printStackTrace();
    }
}

@Override
```

```
protected void onActivityResult(int requestCode, int resultCode,
    Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

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

        // Process the attach image request.
        if (requestCode == REQUEST_CODE_ATTACH_IMAGE) {
            // Get the image URL to extract the file path and then attach the file.
            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));

            mSpenNoteDoc.attachFile(ATTACH_IMAGE_KEY, imagePath);
        }
    }
}

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

For more information, see PenSample2_6_AttachFile.java in PenSample2_6_AttachFile.

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

4.2.6.1 Adding a Listener for the Insert Image Button

To handle Insert Image button events:

1. Create an Insert Image button.
2. Create an `OnClickListener` listener instance for the Insert Image button, mImgObjBtnClickListener in the sample, and register it by calling `setOnClickListener()` on the button.
3. In the `onCreate()` method when the internal action mode is insert image, close the active settings views (if any) and create a dialog box to ask users if they want to change the image.
4. When users select yes, create a dialog to warn them that large images may take some time to load. If they continue, create an intent to call `startActivityForResult()` to select an image file from the device gallery application.

```
closeSettingView();
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("Change the object image. Continue?")
                    .setPositiveButton("Yes",
                    new DialogInterface.OnClickListener() {
                          @Override
                          public void onClick(DialogInterface dialog, int which) {
                              changeImgObj();
                              // Finish dialog.
                              dialog.dismiss();
                          }
                    })
                    .setNegativeButton("No",
                    new DialogInterface.OnClickListener() {
                          @Override
                          public void onClick(
                              DialogInterface dialog, int which) {
                              dialog.dismiss();
                          }
                    }).show();
dlg = null;
```

4.2.6.2 Handling Gallery Image Selection Events

To handle the events triggered when the user selects an image from the gallery:

1. Add an `onActivityResult()` callback method to handle the image returned from the gallery application.
2. Get the URI of the image file from the transferred intent and call `SpenNoteDoc.attachFile()` to attach the file. Use "Attach Image Key" as the key value to get the attached file.

If another attached file already exists, detach the old file and attach the new file.

Copy it to SpenNoteDoc.

```
if (requestCode == REQUEST_CODE_ATTACH_IMAGE) {
      // Get the image uri and extract the file path to attach the file.
      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));

      mSpenNoteDoc.attachFile(ATTACH_IMAGE_KEY, imagePath);
}
```

4.2.6.3 Creating and Registering a Touch Event Listener

To handle touch events in your application in Insert Image mode:

1. Create an `SpenTouchListener` listener instance and add the `onTouch()` callback method to handle S pen touch events in the View area.
2. Call `SpenSurfaceView.setTouchListener()` to register the listener.
3. If the tool type of SpenSurfaceView is S pen and the internal application action mode is Insert Image, in `onTouch ()` call `SpenNoteDoc.hasAttachFile()` to check whether a file exists that corresponds to the key value of "Attach Image Key".
4. If a file is attached, call `SpenNoteDoc.getAttachFile()` to change the acquired file into a bitmap.
5. Call `SpenObjectImage.setImage()` to set the bitmap as an image for Insert Image. Otherwise, change the default icon image natively supplied by Android into a bitmap, and call `SpenObjectImage.setImage()` to specify the bitmap as an image for the Insert Image feature.

```
// If an attached file is found, the file is used as
// ObjectImage.

if (mSpenNoteDoc.hasAttachedFile(ATTACH_IMAGE_KEY)) {
        imageBitmap = BitmapFactory.decodeFile(mSpenNoteDoc
                                    .getAttachedFile(ATTACH_IMAGE_KEY));
        // If no attached file is found, the launcher icon is used as
        // ObjectImage.
} else {
        imageBitmap = BitmapFactory.decodeResource(
                                    mContext.getResources(),
                                    R.drawable.ic_launcher);
}
imgObj.setImage(imageBitmap);
```

Note
Pen uses the file selected as an input variable for the `SpenNoteDoc.attachFile()` method and copies it to the active SpenNoteDoc. Pen takes a long time to save large files in the SPD format using `attachFile()`, or to load them. The key value specified in `attachFile()` can be used as an input variable to call `SpenNoteDoc.detachFile()` to remove the attached file from the SpenNoteDoc data.

4.2.7. Adding Pages

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

You can use `SpenNoteDoc.insertPage()` to insert new pages at a specified index and `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 button calls `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.

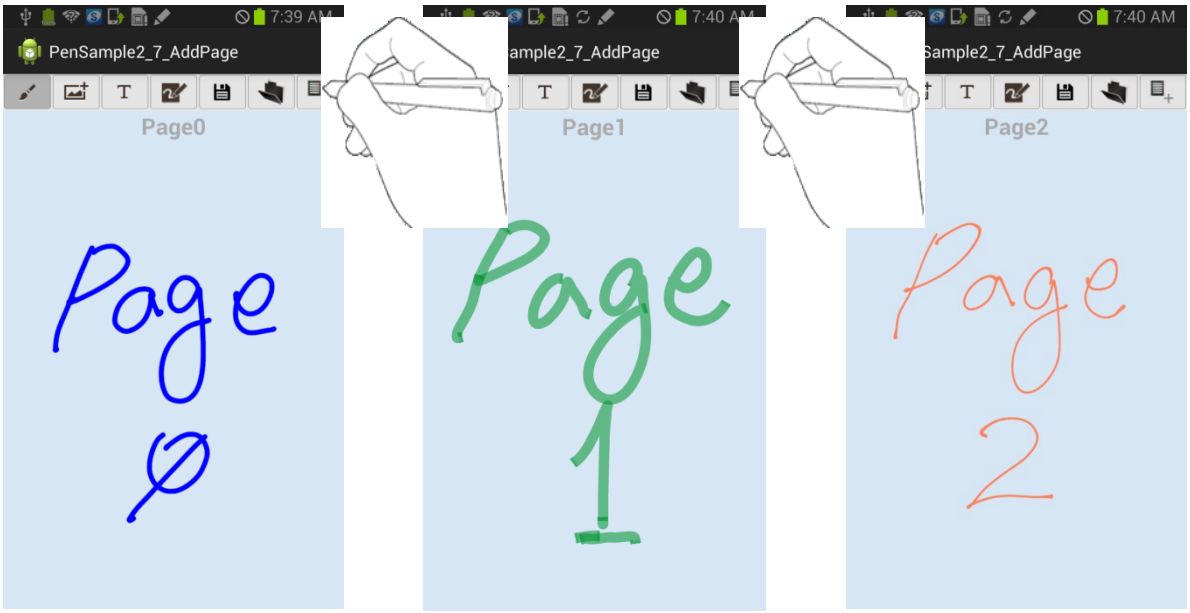


Figure 21: Page Add function

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

              .........

        mSpenSurfaceView.setFlickListener(mFlickListener);

              .........

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

        mTxtView = (TextView) findViewById(R.id.spen_page);
        mTxtView.setText("Page"
            + mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId()));

              .........

}
```

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

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

                closeSettingView();
                // Create a page after the current page.
                mSpenPageDoc = mSpenNoteDoc.insertPage(
                    mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId()) + 1);
                mSpenPageDoc.setBackgroundColor(0xFFD6E6F5);
                mSpenPageDoc.clearHistory();
                v.setClickable(false);
                mSpenSurfaceView.setPageDoc(mSpenPageDoc,
                    SpenSurfaceView.PAGE_TRANSITION_EFFECT_RIGHT,
                    SpenSurfaceView.PAGE_TRANSITION_EFFECT_TYPE_SHADOW, 0);
                mSpenSurfaceView.setPageEffectListener(
                  new SpenPageEffectListener() {
                    @Override
                    public void onFinish() {
                        mAddPageBtn.setClickable(true);

                    }
                });
                mTxtView = (TextView) findViewById(R.id.spen_page);
                mTxtView.setText("Page"
                    + mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId()));
            }
};

            .........

private SpenFlickListener mFlickListener = new SpenFlickListener() {

        @Override
        public boolean onFlick(int direction) {
            int pageIndex =
                mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId());
            int pageCount = mSpenNoteDoc.getPageCount();
            if(pageCount > 1) {
                // Flick to the left to go to the previous page.
                if (direction == DIRECTION_LEFT) {
                    mSpenPageDoc =
                        mSpenNoteDoc.getPage((pageIndex + pageCount - 1)
                            % pageCount);
                    mSpenSurfaceView.setPageDoc(mSpenPageDoc,
                        SpenSurfaceView.PAGE_TRANSITION_EFFECT_LEFT,
                        SpenSurfaceView.PAGE_TRANSITION_EFFECT_TYPE_SHADOW, 0);

                // Flick to the right to go to the next page.
                } else if (direction == DIRECTION_RIGHT) {
                    mSpenPageDoc =
                        mSpenNoteDoc.getPage((pageIndex + 1) % pageCount);
                    mSpenSurfaceView.setPageDoc(mSpenPageDoc,
                        SpenSurfaceView.PAGE_TRANSITION_EFFECT_RIGHT,
                        SpenSurfaceView.PAGE_TRANSITION_EFFECT_TYPE_SHADOW, 0);
```

```
                }
                mTxtView = (TextView) findViewById(R.id.spen_page);
                mTxtView.setText("Page"
                    + mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId()));
                return true;
            }
            return false;
        }
};

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

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.

4.2.7.1 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 settings views.
- 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 `SpenSurfaceView.setPageDoc()` to set the new page in your SpenSurfaceView 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 `SpenSurfaceView.setPageDoc()` is completed. This can cause problems in your application. To prevent this:
    - Disable the Add New Page button before calling `SpenSurfaceView.setPageDoc()`.
    - Register an SpenPageEffectListener instance.
    - In the `onFinish()` callback method, which is called on completion of a page effect, enable the button.
-

```
closeSettingView();
// Create a page after the current page.
mSpenPageDoc = mSpenNoteDoc.insertPage(
                    mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId()) + 1);
```

```
mSpenPageDoc.setBackgroundColor(0xFFD6E6F5);
mSpenPageDoc.clearHistory();
v.setClickable(false);
mSpenSurfaceView.setPageDoc(mSpenPageDoc,
              SpenSurfaceView.PAGE_TRANSITION_EFFECT_RIGHT,
              SpenSurfaceView.PAGE_TRANSITION_EFFECT_TYPE_SHADOW, 0);

mSpenSurfaceView.setPageEffectListener(
  new SpenPageEffectListener() {
    @Override
    public void onFinish() {
        mAddPageBtn.setClickable(true);
    }
});
mTxtView = (TextView) findViewById(R.id.spen_page);
mTxtView.setText("Page" + mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId()));
```

4.2.7.2 Registering a Flick Event Listener

To handle flick events in the View area in your application:

1. Create an `SpenFlickListener` listener interface for flick events in the View area.
2. Call `SpenSurfaceView.setFlickListener()` to register the listener.

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

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

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

```
public boolean onFlick(int direction) {
            int pageIndex =
                mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId());
            int pageCount = mSpenNoteDoc.getPageCount();
            if(pageCount > 1) {
                // Flick to the left to go to the previous page.
                if (direction == DIRECTION_LEFT) {
                    mSpenPageDoc =
                        mSpenNoteDoc.getPage((pageIndex + pageCount - 1)
                            % pageCount);
                    mSpenSurfaceView.setPageDoc(mSpenPageDoc,
                        SpenSurfaceView.PAGE_TRANSITION_EFFECT_LEFT,
                        SpenSurfaceView.PAGE_TRANSITION_EFFECT_TYPE_SHADOW, 0);

                // Flick to the right to go to the next page.
                } else if (direction == DIRECTION_RIGHT) {
                    mSpenPageDoc =
                        mSpenNoteDoc.getPage((pageIndex + 1) % pageCount);
                    mSpenSurfaceView.setPageDoc(mSpenPageDoc,
                        SpenSurfaceView.PAGE_TRANSITION_EFFECT_RIGHT,
                        SpenSurfaceView.PAGE_TRANSITION_EFFECT_TYPE_SHADOW, 0);
                }
                mTxtView = (TextView) findViewById(R.id.spen_page);
```

```
                mTxtView.setText("Page"
                    + mSpenNoteDoc.getPageIndexById(mSpenPageDoc.getId()));
                return true;
            }
            return false;
}
```

4.2.8. Using Extra Data

Pen provides methods to save any additional data required by your applications. As shown in the following sample code, Pen links a user-defined key value to the data, which enables Pen 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 does not record changes made to set up extra data in the history stack. You cannot restore extra data with the Undo command.

Method	Description

`setExtraDataString`
`setExtraDataInt`
`setExtraDataStringArray`
`setExtraDataByteArray`
	Sets extra data for the specified key.

`getExtraDataString`
`getExtraDataInt`
`getExtraDataStringArray`
`getExtraDataByteArray`
	Returns extra data that corresponds to the specified key.

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

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

4.3. Selecting Objects

Pen allows you to resize, relocate, or rotate objects added to the SpenSurfaceView instance, and to group and ungroup multiple objects. You can use `SpenPageDoc.moveObjectIndex()` to edit the order of objects in the SpenPageDoc instance.

4.3.1. Selecting Top Objects

The sample application implements the following features:

- Selection Tool button for selecting objects.
- Relocating, rotating, or resizing objects according to pen events.
- When the Selection Tool button is clicked, the `onClick()` callback method calls `SpenSurfaceView.setToolTypeAction()` to set the action for TOOL_SPEN to ACTION_SELECTION and the internal application action mode to Object Select.
- Listener for touch events.
- When a touch event takes place, `SpenPageDoc.findTopObjectAtPosition()` is called to bring the highest level object to the touch location.

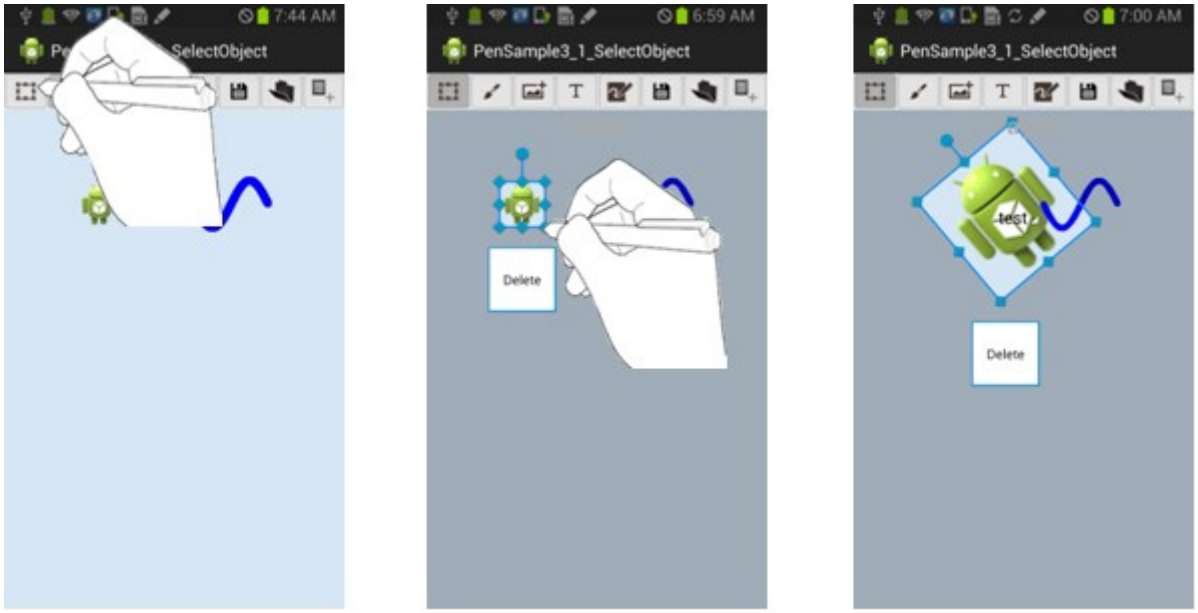


Figure 22: Object Select function

Note
If `SpenObjectBase.isMovable()` or `SpenObjectBase.isRotatable()` returns false, Pen does not allow the selected object to be relocated or rotated. The `setMovable()` or `setRotatable()` methods handle the relocation or rotation modes respectively. Use `setResizeOption()` to resize objects. Pen supports the following resize options:

Note

Resize option	Value	Description
RESIZE_OPTION_FREE	0	Ignores the aspect ratio when resizing objects.
RESIZE_OPTION_KEEP_RATIO	1	Keeps the aspect ratio when resizing objects.
RESIZE_OPTION_DISABLE	2	Disables object resizing.

Pen applies these option values when editing in SpenSurfaceView. For example, if `SpenObjectBase.isRotatable()` returns false, you can use `setRotation()` to allow rotation of the object.

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

              .........

        // Register the listener.
        mSpenSurfaceView.setTouchListener(mPenTouchListener);
        mSpenSurfaceView.setColorPickerListener(mColorPickerListener);
        mSpenSurfaceView.setTextChangeListener(mTextChangeListener);
        mSpenSurfaceView.setFlickListener(mFlickListener);
        mSpenSurfaceView.setControlListener(mControlListener);

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

              .........

        addImgObject(200, 200);
        addTextObject(300, 200, "test");
        addStrokeObject(400, 200);

              .........

}

              .........

private SpenTouchListener mPenTouchListener = new SpenTouchListener() {

        @Override
        public boolean onTouch(View view, MotionEvent event) {
            if (event.getAction() == MotionEvent.ACTION_UP
                && event.getToolType(0) == mToolType) {
                // Check whether to create the control.
                SpenControlBase control = mSpenSurfaceView.getControl();
                if (control == null) {
                // If in Insert ObjectImage mode when a touch event takes place.
                    if (mMode == MODE_IMG_OBJ) {
                        addImgObject(event.getX(), event.getY());
```

```
                        return true;

                    // If in Insert ObjectTextBox mode when a touch event occurs.
                    } else if (mSpenSurfaceView.
                                getToolTypeAction(mToolType)
                        == SpenSurfaceView.ACTION_TEXT) {
                                        SpenObjectTextBox obj =
                            addTextObject(event.getX(), event.getY(), null);
                        mSpenPageDoc.selectObject(obj);
                        mSpenSurfaceView.update();

                        return true;

                    // If in Insert ObjectStroke mode when a touch event occurs.
                    } else if (mMode == MODE_STROKE_OBJ) {
                        addStrokeObject(event.getX(), event.getY());

                        return true;
                    }
                }
            }
            return true;
        }
};

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

                // Change to selection mode.
                mMode = MODE_SELECTION;
                selectButton(mSelectionBtn);
                mSpenSurfaceView.setToolTypeAction(mToolType,
                    SpenSurfaceView.ACTION_SELECTION);
            }
};

              .........

private void addImgObject(float x, float y) {
        SpenObjectImage imgObj = new SpenObjectImage();
        Bitmap imageBitmap;
        // Set the Bitmap file to ObjectImage.
        // If an attached file exists, use this as ObjectImage.
        if (mSpenNoteDoc.hasAttachedFile(ATTACH_IMAGE_KEY)) {
            imageBitmap =
                BitmapFactory.decodeFile(mSpenNoteDoc
                    .getAttachedFile(ATTACH_IMAGE_KEY));
        // If no attached file exists, use the launcher icon as ObjectImage.
        } else {
            imageBitmap =
                BitmapFactory.decodeResource(
                    mContext.getResources(),
                    R.drawable.ic_launcher);
        }
        imgObj.setImage(imageBitmap);
```

```
        // Specify the location where ObjectImage is inserted and add PageDoc.
        float imgWidth = imageBitmap.getWidth();
        float imgHeight = imageBitmap.getHeight();
        RectF rect = getRealPoint(x, y, imgWidth, imgHeight);
        imgObj.setRect(rect, true);
        mSpenPageDoc.appendObject(imgObj);
        mSpenSurfaceView.update();

        imageBitmap.recycle();
}

private SpenObjectTextBox addTextObject(float x, float y, String str) {
        // Specify the location where ObjectTextBox is inserted and add PageDoc.
        SpenObjectTextBox textObj = new SpenObjectTextBox();
        RectF rect = getRealPoint(x, y, 0, 0);
        rect.right += 200;
        rect.bottom += 50;
        textObj.setRect(rect, true);
        textObj.setText(str);
        mSpenPageDoc.appendObject(textObj);
        mSpenSurfaceView.update();

        return textObj;
}

private void addStrokeObject(float x, float y) {
        // Specify the location where ObjectStroke is inserted and add PageDoc.
        RectF rect = getRealPoint(x, y, 0, 0);
        float rectX = rect.centerX();
        float rectY = rect.centerY();
        int pointSize = 157;
        float[][] strokePoint = new float[pointSize][2];
        for(int i = 0; i < pointSize; i++) {
            strokePoint[i][0] = rectX++;
            strokePoint[i][1] = (float) (rectY + Math.sin(.04 * i) * 50);
        }
        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 = strokePoint[i][0];
            points[i].y = strokePoint[i][1];
            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);
        mSpenSurfaceView.update();
}
```

```
private RectF
        getRealPoint(float x, float y, float width, float height) {
        float panX = mSpenSurfaceView.getPan().x;
        float panY = mSpenSurfaceView.getPan().y;
        float zoom = mSpenSurfaceView.getZoomRatio();
        width *= zoom;
        height *= zoom;
        RectF realRect = new RectF();
        realRect.set(
            (x - width / 2) / zoom + panX, (y - height / 2) / zoom + panY,
            (x + width / 2) / zoom + panX, (y + height / 2) / zoom + panY);
        return realRect;
}

              .........

private 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) {
            switch (itemId) {
            // Delete the selected object.

                case CONTEXT_MENU_DELETE_ID:
                    mSpenPageDoc.removeSelectedObject();
                    mSpenSurfaceView.closeControl();
                    mSpenSurfaceView.update();
                    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.
            menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_DELETE_ID,
                                                 "Delete", true));

            return true;
        }

        @Override
```

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

private void selectButton(View v) {
        // Enable or disable the buttons depending on the mode.
        mSelectionBtn.setSelected(false);
        mPenBtn.setSelected(false);
        mImgObjBtn.setSelected(false);
        mTextObjBtn.setSelected(false);
        mStrokeObjBtn.setSelected(false);

        v.setSelected(true);

        closeSettingView();
}

          .....
```

For more information, see PenSample3_1_SelectObject.java in PenSample3_1_SelectObject.

The following sections provide more details on the steps involved in selecting an object.

4.3.1.1 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, mSelectionBtnClickListener in the sample, and register it by calling `setOnClickListener()` on the button.

In the `onClick()` method:

- Set the internal application action mode to object selection.
- Indicate the button has been selected.
- Call `SpenSurfaceView.setToolTypeAction()` to set the action for `mToolType` to ACTION_SELECTION.

```
mMode = MODE_SELECTION;
selectButton(mSelectionBtn);
mSpenSurfaceView.setToolTypeAction(mToolType,
                                   SpenSurfaceView.ACTION_SELECTION);
```

Note
If the action for the tool type is set to ACTION_SELECTION, you do not have to implement SpenTouchListener to select objects because the touched object is auto-selected. You can get the objects by calling `findObjectAtPosition()`, `findObjectInClosedCurve()`, `findObjectInRect()` and `findTopObjectAtPosition()`. Pen provides the following view

Note

modes:

Type filter	Value	Description
FIND_TYPE_STROKE	1	To get stroke objects.
FIND_TYPE_TEXT_BOX	2	To get text box objects.
FIND_TYPE_IMAGE	4	To get image objects.
FIND_TYPE_CONTAINER	8	To get object containers.
FIND_TYPE_ALL	31	To get all types of objects.

4.3.1.2 Creating and Registering a Control Event Listener

To handle control events in the View area in your application:

1. Create an `SpenControlListener` listener instance to handle state changes in the View area.
2. Call `SpenSurfaceView.setControlListener()` to register the listener.

In the `onCreated()` callback method, which is called when a control is displayed in the View area, add a "Delete" item to the context menu to delete a selected object.

In the `onMenuSelected()` callback method, which is called when an item is selected from a context menu appearing on a control, call the following methods:

- `SpenPageDoc.removeSelectedObject()` to delete the selected object.
- `SpenSurfaceView.closeControl()` to close the control.
- `SpenSurfaceView.update()` to refresh the screen.

```
private SpenControlListener mControlListener = new SpenControlListener() {

              .........

       public boolean onMenuSelected(
           ArrayList<SpenObjectBase> objectList, int itemId) {
           switch (itemId) {
           // Delete the selected object.
           case CONTEXT_MENU_DELETE_ID:
               mSpenPageDoc.removeSelectedObject();
               mSpenSurfaceView.closeControl();
               mSpenSurfaceView.update();
               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.
            menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_DELETE_ID,
                "Delete", true));

            return true;
        }

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

Note

Pen offers easy-to-format context menus depending on the selected object. As shown in the following sample code, SpenControlListener.onCreated() creates an SpenContextMenuItemInfo instance to add your menu items to the context menu. When you select an item from the context menu, onMenuSelected() is called to execute the selected item(command).

```
private SpenControlListener mControlListener = new
SpenControlListener() {
     public boolean onCreated(ArrayList<SpenObjectBase> objectList,
            ArrayList<Rect> relativeRectList,
            ArrayList<SpenContextMenuItemInfo> menu,
            ArrayList<Integer> styleList, int pressType) {
           menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_DELETE_ID,
                                "Delete", true));
           menu.add(new SpenContextMenuItemInfo(
                                CONTEXT_MENU_MOVE_TO_TOP_ID,
                                "Move to top", true));
     }
     public boolean onMenuSelected(
            ArrayList<SpenObjectBase> objectList, int itemId) {
            SpenObjectBase object = objectList.get(0);
            switch (itemId) {
               case CONTEXT_MENU_DELETE_ID:
                     mSpenPageDoc.removeSelectedObject();
                     mSpenSurfaceView.update();
                     mSpenSurfaceView.closeControl();
                     break;
               case CONTEXT_MENU_MOVE_TO_TOP_ID:
                     mSpenPageDoc.moveObjectIndex(
```

Note

```
                        object, mSpenPageDoc.getObjectCount(true) - 1
                        - mSpenPageDoc.getObjectIndex(object), true);
                    mSpenSurfaceView.update();
                    mSpenSurfaceView.closeControl();
                    break;
            }
}
```

4.3.2. Using the Rectangle and Lasso Selection Tool

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

Pen 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 view to let users specify the selection type: Lasso or Rectangle.
- One or multiple object selection.

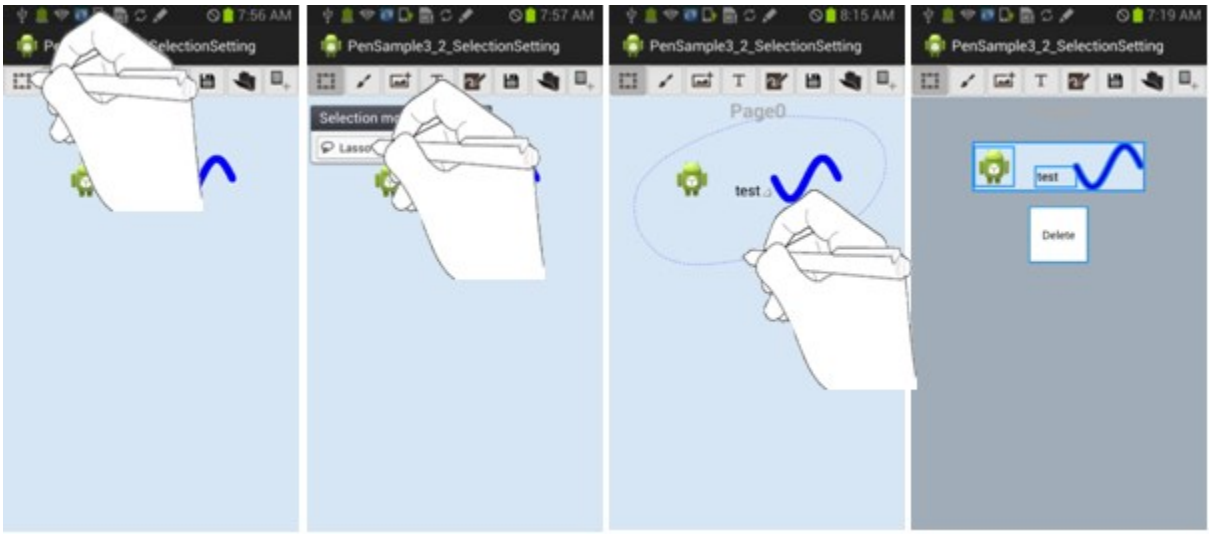


Figure 23: Selection settings

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

              .........

        // Create SelectionSettingView.
        mSelectionSettingView =
            new SpenSettingSelectionLayout(mContext, new String(),
                spenViewLayout);
        if (mSelectionSettingView == null) {
            Toast.makeText(mContext, "Cannot create new SelectionSettingView.",
                Toast.LENGTH_SHORT).show();
            finish();
        }
        spenViewContainer.addView(mPenSettingView);
        spenViewContainer.addView(mTextSettingView);
        spenViewContainer.addView(mSelectionSettingView);

        // Create Pen View.
        mSpenSurfaceView = new SpenSurfaceView(mContext);
        if (mSpenSurfaceView == null) {
            Toast.makeText(mContext, "Cannot create new SpenSurfaceView.",
                Toast.LENGTH_SHORT).show();
            finish();
        }
        spenViewLayout.addView(mSpenSurfaceView);
        mPenSettingView.setCanvasView(mSpenSurfaceView);
        mTextSettingView.setCanvasView(mSpenSurfaceView);
        mSelectionSettingView.setCanvasView(mSpenSurfaceView);

              .........

        mSpenSurfaceView.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) {
                mSpenSurfaceView.closeControl();

                // When Pen is in selection mode.
                if (mSpenSurfaceView.getToolTypeAction(mToolType) ==
                        SpenSurfaceView.ACTION_SELECTION) {
                    // Close the SelectionSettingView if SelectionSettingView is
                    // displayed.
                    if (mSelectionSettingView.isShown()) {
                        mSelectionSettingView.setVisibility(View.GONE);
```

```
                    // Show the SelectionSettingView if SelectionSettingView is not
                    // displayed.
                    } else {
                        mSelectionSettingView
                             .setViewMode(SpenSettingSelectionLayout.VIEW_MODE_NORMAL);
                        mSelectionSettingView.setVisibility(View.VISIBLE);
                    }
                // Change to selection mode if Pen is in selection mode.
                } else {
                    mMode = MODE_SELECTION;
                    selectButton(mSelectionBtn);
                    mSpenSurfaceView.setToolTypeAction(mToolType,
                                                SpenSurfaceView.ACTION_SELECTION);
                }
            }
};

              .........

private SpenSelectionChangeListener mSelectionListener =
        new SpenSelectionChangeListener() {

            @Override
            public void onChanged(SpenSettingSelectionInfo info) {
                // Close the setting view if the selection type changes.
                mSelectionSettingView.setVisibility(SpenSurfaceView.GONE);
            }
};

private void selectButton(View v) {
        // Enable or disable the buttons depending on the mode.
        mSelectionBtn.setSelected(false);
        mPenBtn.setSelected(false);
        mImgObjBtn.setSelected(false);
        mTextObjBtn.setSelected(false);
        mStrokeObjBtn.setSelected(false);

        v.setSelected(true);

        closeSettingView();
}

private void closeSettingView() {
        // Close all settings view.
        mPenSettingView.setVisibility(SpenSurfaceView.GONE);
        mTextSettingView.setVisibility(SpenSurfaceView.GONE);
        mSelectionSettingView.setVisibility(SpenSurfaceView.GONE);
}

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

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

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

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

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

For more information, see PenSample3_2_SelectionSetting.java in PenSample3_2_SelectionSetting.

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

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

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

              .........

mSelectionSettingView.setCanvasView(mSpenSurfaceView);
```

4.3.2.2 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()` on the 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 (mSpenSurfaceView.getToolTypeAction(mToolType) ==
        SpenSurfaceView.ACTION_SELECTION) {
        // Close the SelectionSettingView if SelectionSettingView is
        // displayed.
        if (mSelectionSettingView.isShown()) {
            mSelectionSettingView.setVisibility(View.GONE);
            // Show the SelectionSettingView if SelectionSettingView is not
            // displayed.
        } else {
            mSelectionSettingView
                        .setViewMode(SpenSettingSelectionLayout.VIEW_MODE_NORMAL);
            mSelectionSettingView.setVisibility(View.VISIBLE);
        }
```

4.3.2.3 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 `SpenSurfaceView.setSelectionChangeListener()` to register the listener.

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

```
public void onChanged(SpenSettingSelectionInfo info) {
     // Close the setting view if the selection type changes.
     mSelectionSettingView.setVisibility(SpenSurfaceView.GONE);
}
```

4.3.2.4 Preventing Memory Leaks

To prevent memory leaks:

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

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

4.3.3. Grouping and Ungrouping Objects

You can use Pen to select multiple objects in your application.

The sample application implements the following features:

- Context menu for grouping multiple objects. The menu appears when multiple objects are located within the selected area.
- Registers the group of objects internally in SpenObjectContainer.
- Context menu for ungrouping grouped objects. This menu appears when the area where the selected objects are grouped is touched
- When users select Ungroup from the context menu, `SpenPageDoc.ungroupObject()` is called to ungroup the objects.

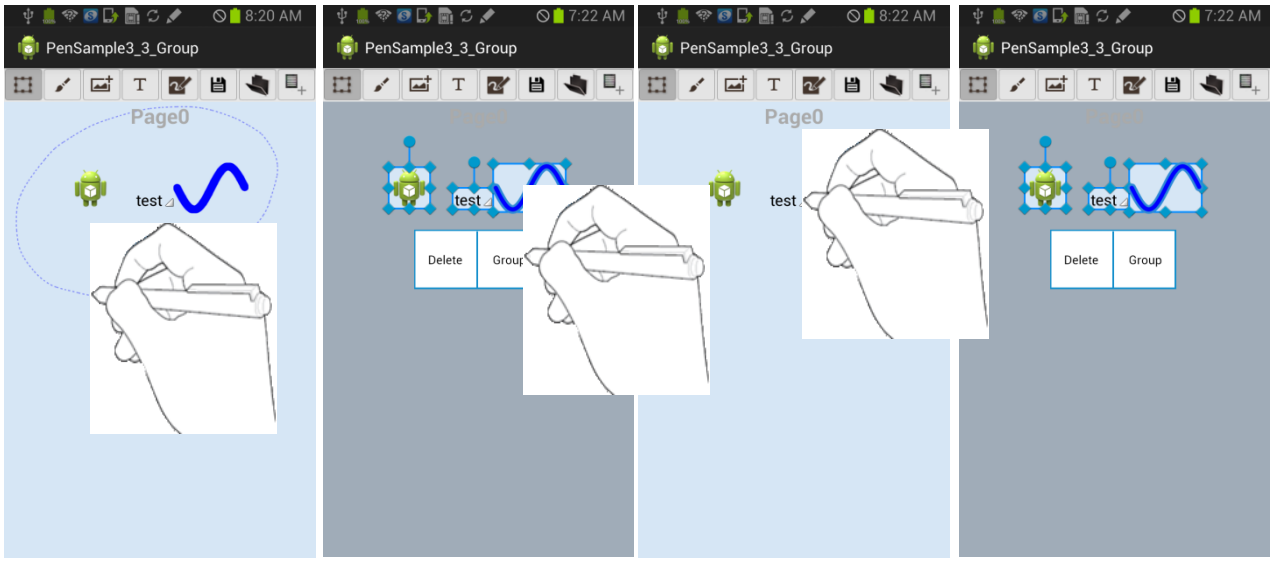


Figure 24: Group and Ungroup

```
public class PenSample3_3_Group extends Activity {

    private final int CONTEXT_MENU_DELETE_ID = 10;
    private final int CONTEXT_MENU_GROUP_ID = 20;
    private final int CONTEXT_MENU_UNGROUP_ID = 21;
```

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

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

            .........

        mSpenSurfaceView.setControlListener(mControlListener);
        mSpenSurfaceView.setSelectionChangeListener(mSelectionListener);

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

            .........

    }

            .........

    private 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;
            switch (itemId) {
            // Delete the selected object.
            case CONTEXT_MENU_DELETE_ID:
                mSpenPageDoc.removeSelectedObject();
                mSpenSurfaceView.closeControl();
                mSpenSurfaceView.update();
                break;

            // Group the objects.
            case CONTEXT_MENU_GROUP_ID:
                objContainer = mSpenPageDoc.groupObject(objectList, false);
                mSpenSurfaceView.closeControl();
                mSpenPageDoc.selectObject(objContainer);
                mSpenSurfaceView.update();
                break;

            // Ungroup the grouped objects.
            case CONTEXT_MENU_UNGROUP_ID:
```

```
                ArrayList<SpenObjectBase> objList = new ArrayList<SpenObjectBase>();
                for(SpenObjectBase selectedObj : objectList) {
                    if(selectedObj.getType() == SpenObjectBase.TYPE_CONTAINER) {
                        objContainer = (SpenObjectContainer) selectedObj;
                        for(SpenObjectBase obj : objContainer.getObjectList()) {
                            objList.add(obj);
                        }
                        mSpenPageDoc.ungroupObject((SpenObjectContainer) selectedObj,
                                                   false);
                    }
                }
                mSpenSurfaceView.closeControl();
                mSpenPageDoc.selectObject(objList);
                mSpenSurfaceView.update();
            }

            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.
            menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_DELETE_ID,
                                                 "Delete", true));
            // Display the Group menu item when there is more than 1 selected object.
            if(objectList.size() > 1) {
                menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_GROUP_ID,
                                                     "Group", true));
            }
            // Display the Ungroup menu item when there is an ObjectContainer among
            // the selected objects.
            for(SpenObjectBase obj : objectList) {
                if(obj.getType() == SpenObjectBase.TYPE_CONTAINER) {
                    menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_UNGROUP_ID,
                                                         "Ungroup", true));
                    break;
                }
            }            if(objectList.size() == 1)
                return true;
            // Create a control for each object.
            SpenControlList controlList = new SpenControlList(mContext, mSpenPageDoc);
            controlList.setObject(objectList);
            controlList.setGroup(false);
            mSpenSurfaceView.setControl(controlList);
            controlList.setContextMenu(menu);

            return false;
        }

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

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

For more information, see PenSample3_3_Group.java in PenSample3_3_Group.

The following sections provide more details on the steps involved in using the group and ungroup object features.

4.3.3.1 Creating a Context Menu in a Control

To create a context menu when control events occur in your application:

1. Create an `SpenControlListener` listener instance.

Call `SpenSurfaceView.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 as follows:

- Add the “Delete” menu item to the context menu so that users can select and delete an object.
- Add the “Group” menu item to the context menu when users select more than one object.
- Add the “Ungroup” menu item to the context menu when users select objects with at least one SpenObjectContainer, which indicates that grouped objects are present.
- Create SpenContextMenuItemInfo to register these commands in the context menu.
- Create an `SpenControlList` instance and call `setObject()` to link controls for each object when multiple objects are selected.
- Call `SpenControlList.setGroup()` and pass the Boolean value false.
- Call `SpenSurfaceView.setControl()`.
- Return the Boolean value false to link controls for selected objects to one another.

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

            // Set the context menu.
            menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_DELETE_ID,
                                                 "Delete", true));
            // Display the Group menu item when there is more than 1 selected object.
            if(objectList.size() > 1) {
                menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_GROUP_ID,
                                                     "Group", true));
            }
            // Display the Ungroup menu item when there is any ObjectContainer among
            // the selected objects.
            for(SpenObjectBase obj : objectList) {
                if(obj.getType() == SpenObjectBase.TYPE_CONTAINER) {
```

```
                    menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_UNGROUP_ID,
                                                         "Ungroup", true));
                    break;
                }
            }
            if(objectList.size() == 1)
                return true;
            // Create a control for each object.
            SpenControlList controlList = new SpenControlList(mContext, mSpenPageDoc);
            controlList.setObject(objectList);
            controlList.setGroup(false);
            mSpenSurfaceView.setControl(controlList);
            controlList.setContextMenu(menu);

            return false;
}
```

4.3.3.2 Handling Context Menu Events in a Control

To handle the context menu selection events:

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 “Delete” menu item is selected, do the following:

- `SpenPageDoc.removeSelectedObject()` to remove the selected object.
- `SpenSurfaceView.closeControl()` to close the control.
- `SpenSurfaceView.update()` to refresh the screen.

When the user selects the “Group” menu item, do the following:

- Call `SpenPageDoc.groupObject()` and pass the list of selected objects to get an instance of SpenObjectContainer for the grouped objects.
- Pass this instance when calling `SpenPageDoc.selectObject()`.
- Call `SpenSurfaceView.update()` to refresh the screen.

When the “Ungroup” menu item is selected, do the following:

- Browse the object list to check for an object of the type SpenObjectContainer and call `SpenPageDoc.unGroup()` to ungroup the object.
- Add each of the ungrouped objects to an SpenObjectBase array.
- Call `SpenPageDoc.selectObject()` and pass the array to have each object remain selected.
- Call `SpenSurfaceView.update()` to refresh the screen.

```
public boolean onMenuSelected(
            ArrayList<SpenObjectBase> objectList, int itemId) {
        SpenObjectContainer objContainer;
```

```
        switch (itemId) {
            // Delete the selected object.
            case CONTEXT_MENU_DELETE_ID:
                mSpenPageDoc.removeSelectedObject();
                mSpenSurfaceView.closeControl();
                mSpenSurfaceView.update();
                break;

            // Group the objects.
            case CONTEXT_MENU_GROUP_ID:
                objContainer = mSpenPageDoc.groupObject(objectList, false);
                mSpenSurfaceView.closeControl();
                mSpenPageDoc.selectObject(objContainer);
                mSpenSurfaceView.update();
                break;

            // Ungroup the grouped objects.
            case CONTEXT_MENU_UNGROUP_ID:
                ArrayList<SpenObjectBase> objList = new ArrayList<SpenObjectBase>();
                for(SpenObjectBase selectedObj : objectList) {
                    if(selectedObj.getType() == SpenObjectBase.TYPE_CONTAINER) {
                        objContainer = (SpenObjectContainer) selectedObj;
                        for(SpenObjectBase obj : objContainer.getObjectList()) {
                            objList.add(obj);
                        }
                        mSpenPageDoc.ungroupObject((SpenObjectContainer) selectedObj,
false);
                    }
                }
                mSpenSurfaceView.closeControl();
                mSpenPageDoc.selectObject(objList);
                mSpenSurfaceView.update();
        }

        return true;
}
```

4.3.4. Bringing Objects Forward and Backward

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

This functionality is added to the sample application that implemented Grouping and Ungrouping Objects.

The sample application implements the following features:

- Adds the following menu items to the control created for Grouping and Ungrouping 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.

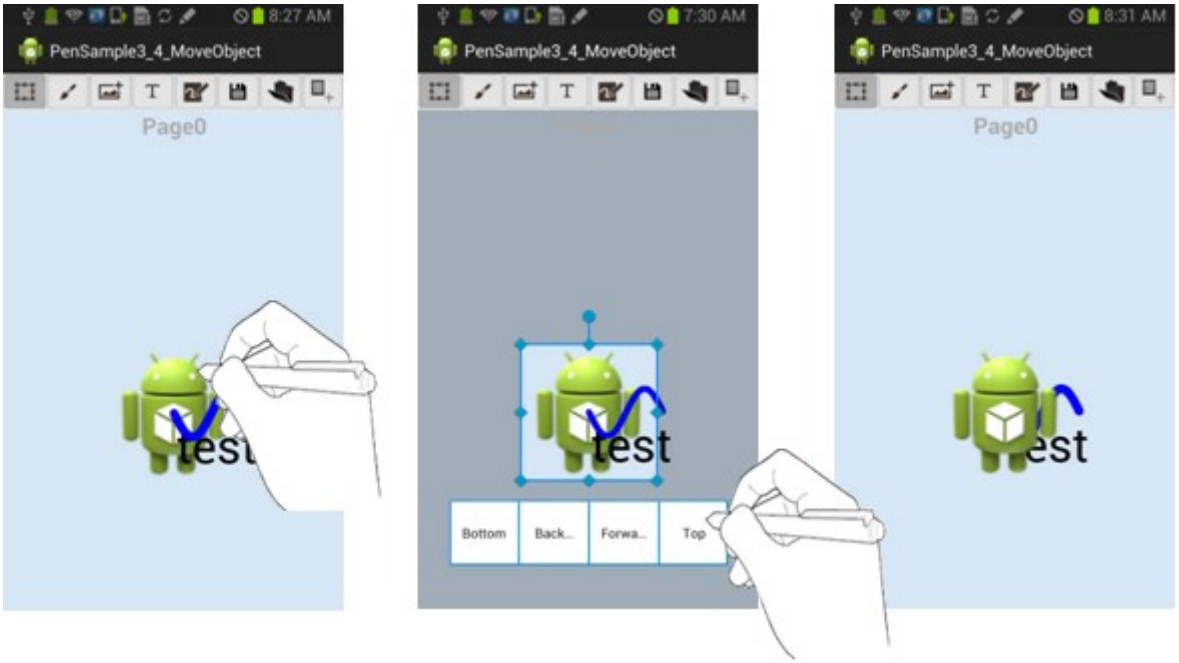


Figure 25: Moving an object

```
public class PenSample3_4_MoveObject extends Activity {

    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_move_object);
        mContext = this;

              .........

        mSpenSurfaceView.setControlListener(mControlListener);
        mSpenSurfaceView.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);
```

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

    }

            .........

    private 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) {
            // Delete the object.
            case CONTEXT_MENU_DELETE_ID:
                mSpenPageDoc.removeSelectedObject();
                mSpenSurfaceView.closeControl();
                mSpenSurfaceView.update();
                break;

            // Group the objects.
            case CONTEXT_MENU_GROUP_ID:
                objContainer = mSpenPageDoc.groupObject(objectList, false);
                mSpenSurfaceView.closeControl();
                mSpenPageDoc.selectObject(objContainer);
                mSpenSurfaceView.update();
                break;

            // Ungroup the grouped objects.
            case CONTEXT_MENU_UNGROUP_ID:
                ArrayList<SpenObjectBase> objList = new ArrayList<SpenObjectBase>();
                for(SpenObjectBase selectedObj : objectList) {
                    if(selectedObj.getType() == SpenObjectBase.TYPE_CONTAINER) {
                        objContainer = (SpenObjectContainer) selectedObj;
                        for(SpenObjectBase obj : objContainer.getObjectList()) {
                            objList.add(obj);
                        }
                        mSpenPageDoc.ungroupObject((SpenObjectContainer)    selectedObj
false);
                    }
                }
                mSpenSurfaceView.closeControl();
                mSpenPageDoc.selectObject(objList);
                mSpenSurfaceView.update();
                break;
```

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

            // Send the selected object back one index.
            case CONTEXT_MENU_MOVE_TO_BACKWARD_ID:
                if (mSpenPageDoc.getObjectIndex(object) > 0) {
                    mSpenPageDoc.moveObjectIndex(object, -1, true);
                    mSpenSurfaceView.update();
                }
                break;

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

            // Bring the selected object forward to the top.
            case CONTEXT_MENU_MOVE_TO_TOP_ID:
                mSpenPageDoc.moveObjectIndex(
                    object, mSpenPageDoc.getObjectCount(true) - 1
                    - mSpenPageDoc.getObjectIndex(object), true);
                mSpenSurfaceView.update();
                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.
            menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_DELETE_ID,
                "Delete", true));
            // Display the Group menu item when there is more than 1 selected object.
            if(objectList.size() > 1) {
                menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_GROUP_ID,
                    "Group", true));
            }
            // Display the Ungroup menu item when there is an ObjectContainer among
            // the selected objects.
            for(SpenObjectBase obj : objectList) {
                if(obj.getType() == SpenObjectBase.TYPE_CONTAINER) {
                    menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_UNGROUP_ID,
                        "Ungroup", true));
                    break;
                }
            }
```

```
            // Display the menu items for object placement when a single 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;
            }
            // Create a control for each object.
            SpenControlList controlList = new SpenControlList(mContext, mSpenPageDoc);
            controlList.setObject(objectList);
            controlList.setGroup(false);
            mSpenSurfaceView.setControl(controlList);
            controlList.setContextMenu(menu);

            return false;
        }

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

          .....
```

For more information, see PenSample3_4_MoveObject.java in PenSample3_4_MoveObject.

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

4.3.4.1 Creating a Context Menu in a Control

To create a context menu for control events in your application:

1. Create an `SpenControlListener` listener instance.
2. Call `SpenSurfaceView.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) {

              .........

            // Display the menu items for object placement when single 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));

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

4.3.4.2 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 `SpenSurfaceView.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 back to the bottom.
           case CONTEXT_MENU_MOVE_TO_BOTTOM_ID:
               mSpenPageDoc.moveObjectIndex(object,
                   -mSpenPageDoc.getObjectIndex(object), true);
               mSpenSurfaceView.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 `SpenSurfaceView.update()` to refresh the screen.

```
// Send the selected object back one index.
case CONTEXT_MENU_MOVE_TO_BACKWARD_ID:
      if (mSpenPageDoc.getObjectIndex(object) > 0) {
               mSpenPageDoc.moveObjectIndex(object, -1, true);
               mSpenSurfaceView.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 `SpenSurfaceView.update()` to refresh the screen.

```
// Bring the selected object forward one index.
case CONTEXT_MENU_MOVE_TO_FORWARD_ID:
      if (mSpenPageDoc.getObjectIndex(object) <
              mSpenPageDoc.getObjectCount(true) - 1) {
                    mSpenPageDoc.moveObjectIndex(object, 1, true);
                    mSpenSurfaceView.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 `SpenSurfaceView.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);
```

```
        mSpenSurfaceView.update();
        break;
```

4.4. Working with SOR (S-Pen Object Runtime)

An S-Pen Object Runtime (SOR) is a plug-in for expanding the capabilities of default preloaded objects such as SpenObjectStroke, SpenObjectText, SpenObjectImage, and SpenObjectContainer that are natively supplied by Pen in real-time.

You can create an SOR with SpenObjectRuntimeManager by calling `SpenObjectRuntimeManager.createObjectRuntime()`. Pen returns the SpenObjectRuntime instance that you create.

You need an SpenObjectRuntimeInfo object or the class name value of an SOR to call `createObjectRuntime()`. If the SOR has a private key, the private key should be sent.

Pen provides the following key classes for the SOR functionality:

Class	Description
SpenObjectRuntimeInfo	Provides information to make SpenObjectRuntime available. • Name: The name of ObjectRuntime (String) • className: The class name of ObjectRuntime (String) • version: Version information for ObjectRuntime (Integer) • iconImageURI: The path to the icon image (String) • hasPrivateKey: Whether a private key exists or not (boolean)
SpenObjectRuntime	Provides functions that manipulate SpenObjectRuntime. Prior to using the SpenObjectRuntime class, you should create an instance using ObjectRuntimeManager.
SpenObjectRuntimeManager	Manages instances of SpenObjectRuntime. You can use SpenObjectRuntimeManager to create or delete an instance of SpenObjectRuntime.

4.4.1. Adding Video Objects

The sample application implements the following features:

- Video button to play a video using a preloaded SpenObjectRuntime from Pen.

- Adds video objects using the following classes:
    - SpenObjectRuntime
    - SpenObjectRuntimeInfo
    - SpenObjectRuntimeManager
- On application start up, calls `SpenObjectRuntimeManager.getObjectRuntimeInfoList()` to get the list of the available SpenObjectRuntime objects.
- When the Add Video button is clicked, the sample calls `SpenObjectRuntimeManager.createObjectRuntime()` to create an SpenObjectRuntime instance. The SpenObjectRuntime type is set to SpenObjectRuntimeInterface.TYPE_IMAGE. This creates an image object that is passed when calling `SpenObjectRuntime.start()`.
- Adds a “Run” menu item to the context menu that appears when the user selects a new video object. When the user selects “Run” from the context menu, it calls `SpenObjectRuntime.start()` to run the object. This method is also called to add the SOR.

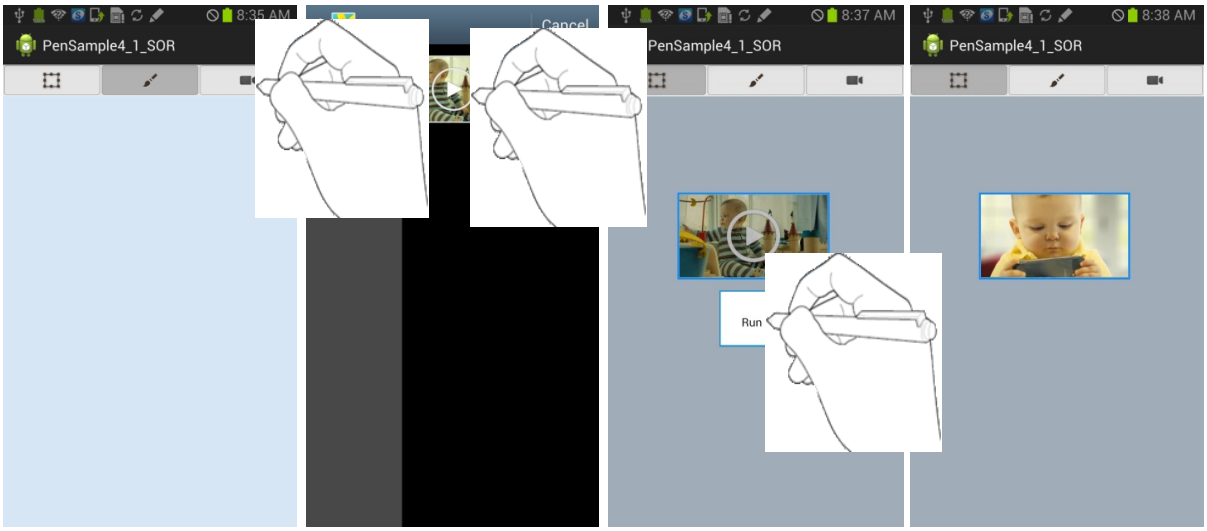


Figure 26: Video SOR

```
public class PenSample4_1_SOR extends Activity {

    private final int CONTEXT_MENU_RUN_ID = 0;

    private Context mContext;
    private Activity mActivity;
    private SpenNoteDoc mSpenNoteDoc;
    private SpenPageDoc mSpenPageDoc;
    private SpenSurfaceView mSpenSurfaceView;
    RelativeLayout mSpenViewLayout;

    private ImageView mSelectionBtn;
    private ImageView mPenBtn;
    private ImageView mVideoBtn;

    private SpenObjectRuntimeManager mSpenObjectRuntimeManager;
    private List<SpenObjectRuntimeInfo> mSpenObjectRuntimeInfoList;
    private SpenObjectRuntimeInfo mObjectRuntimeInfo;
```

```
    private SpenObjectRuntime mVideoRuntime;

    private int mToolType = SpenSurfaceView.TOOL_SPEN;

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

            // Initialize Pen.

            boolean isSpenFeatureEnabled = false;
            Spen spenPackage = new Spen();
            try {
                spenPackage.initialize(this);

                isSpenFeatureEnabled =
                    spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
            } catch (SsdkUnsupportedException e) {
                if( SDKUtils.processUnsupportedException(this, e) == true) {
                    return;
                }
            } catch (Exception e1) {
                Toast.makeText(mContext, "Cannot initialize Pen.",
                               Toast.LENGTH_SHORT).show();
                e1.printStackTrace();
                finish();
            }

            mSpenViewLayout =
                (RelativeLayout) findViewById(R.id.spenViewLayout);

            // Create Pen View.
            mSpenSurfaceView = new SpenSurfaceView(mContext);
            if (mSpenSurfaceView == null) {
                Toast.makeText(mContext, "Cannot create new SpenSurfaceView.",
                    Toast.LENGTH_SHORT).show();
                finish();
            }
            mSpenSurfaceView.setZoomable(false);
            mSpenViewLayout.addView(mSpenSurfaceView);
            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);

            // Get the dimensions of the screen of the device.
            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 and then get the instance to use as an input
        // variable.
        mSpenPageDoc = mSpenNoteDoc.appendPage();
        mSpenPageDoc.setBackgroundColor(0xFFD6E6F5);
        mSpenPageDoc.clearHistory();
        // Set PageDoc in the View.
        mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);

        initPenSettingInfo();
        // Register a listener.
        mSpenSurfaceView.setControlListener(mControlListener);

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

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

        mVideoBtn = (ImageView) findViewById(R.id.videoBtn);
        mVideoBtn.setOnClickListener(mVideoBtnClickListener);

        selectButton(mPenBtn);

        // Set up the ObjectRuntimeManager.
        mSpenObjectRuntimeManager = new SpenObjectRuntimeManager(mActivity);
        mSpenObjectRuntimeInfoList =
            new ArrayList<SpenObjectRuntimeInfo>();
        mSpenObjectRuntimeInfoList =
            mSpenObjectRuntimeManager.getObjectRuntimeInfoList();

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

private void initPenSettingInfo() {
    // Initialize settings for the pen.
    SpenSettingPenInfo penInfo = new SpenSettingPenInfo();
    penInfo.color = Color.BLUE;
    penInfo.size = 10;
    mSpenSurfaceView.setPenSettingInfo(penInfo);
}

private final OnClickListener mSelectionBtnClickListener =
    new OnClickListener() {
        @Override
        public void onClick(View v) {
            selectButton(mSelectionBtn);
            mSpenSurfaceView.setToolTypeAction(mToolType,
```

```
                                                SpenSurfaceView.ACTION_SELECTION);
            }
        };

    private final OnClickListener mPenBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                selectButton(mPenBtn);
                mSpenSurfaceView.setToolTypeAction(mToolType,
                                                SpenSurfaceView.ACTION_STROKE);
            }
        };

    private final OnClickListener mVideoBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                mVideoBtn.setClickable(false);
                mSpenSurfaceView.closeControl();
                createObjectRuntime();
            }
        };

    SpenControlListener mControlListener = new SpenControlListener() {

        @Override
        public boolean onCreated(ArrayList<SpenObjectBase> objectList,
            ArrayList<Rect> relativeRectList,
            ArrayList<SpenContextMenuItemInfo> menu,
            ArrayList<Integer> styleList, int pressType, PointF point) {
            if (objectList == null) {
                return false;
            }
            // Display the context menu if any SOR information is found.
            if (objectList.get(0).getSorInfo() != null) {
                menu.add(new SpenContextMenuItemInfo(
                    CONTEXT_MENU_RUN_ID, "Run", true));
                return true;
            }
            return true;
        }

        @Override
        public boolean onMenuSelected(
            ArrayList<SpenObjectBase> objectList, int itemId) {
            if (objectList == null) {
                return true;
            }

            if (itemId == CONTEXT_MENU_RUN_ID) {
                SpenObjectBase object = objectList.get(0);
                mSpenSurfaceView.getControl().setContextMenuVisible(false);

                mSpenSurfaceView.getControl().
                    setStyle(SpenControlBase.STYLE_BORDER_STATIC);

                // Set up listener and make it play.
                mVideoRuntime.setListener(objectRuntimelistener);
```

```
                mVideoRuntime.start(object, getRealRect(object.getRect()),
                    mSpenSurfaceView.getPan(), mSpenSurfaceView.getZoomRatio(),
                    mSpenSurfaceView.getFrameStartPosition(), mSpenViewLayout);

                mSpenSurfaceView.update();
            }
            return false;
        }

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

        @Override
        public void onRectChanged(RectF rect, SpenObjectBase object) {
        }

        @Override
        public void onRotationChanged(float angle,
            SpenObjectBase objectBase) {
        }

        @Override
        public boolean onClosed(ArrayList<SpenObjectBase> objectList) {
            if(mVideoRuntime != null)
                 mVideoRuntime.stop(true);
            return false;
        }
    };

    void createObjectRuntime() {
        if (mSpenObjectRuntimeInfoList == null
            || mSpenObjectRuntimeInfoList.size() == 0) {
            return;
        }

        try {
            for(SpenObjectRuntimeInfo info : mSpenObjectRuntimeInfoList) {
                if (info.name.equalsIgnoreCase("Video")) {
                    mVideoRuntime =
                        mSpenObjectRuntimeManager.createObjectRuntime(info);

                    mObjectRuntimeInfo = info;
                    startObjectRuntime();

                    return;
                }
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            Toast.makeText(mContext, "ObjectRuntimeInfo class not found.",
                Toast.LENGTH_SHORT).show();
        } catch (InstantiationException e) {
            e.printStackTrace();
            Toast.makeText(mContext,
              " Failed to access the ObjectRuntimeInfo constructor.",

                Toast.LENGTH_SHORT).show();
        } catch (IllegalAccessException e) {
```

```
            e.printStackTrace();
            Toast.makeText(mContext,
              " Failed to access the ObjectRuntimeInfo field or method.",

                Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(mContext, "ObjectRuntimeInfo is not loaded.",
                Toast.LENGTH_SHORT).show();
        }
    }

    void startObjectRuntime() {
        if (mVideoRuntime == null) {
            Toast.makeText(mContext,
                "ObjectRuntime is not loaded \n Load Plug-in First !!",
                Toast.LENGTH_SHORT).show();
            return;
        }

        SpenObjectBase objectBase = null;
        switch (mVideoRuntime.getType()) {
            case SpenObjectRuntimeInterface.TYPE_NONE:
                return;
            case SpenObjectRuntimeInterface.TYPE_IMAGE:
                objectBase = new SpenObjectImage();
                break;
            case SpenObjectRuntimeInterface.TYPE_STROKE:
                objectBase = new SpenObjectStroke();
                break;
            case SpenObjectRuntimeInterface.TYPE_CONTAINER:
                objectBase = new SpenObjectContainer();
                break;
            default:
                break;
        }

        if(objectBase == null) {
            Toast.makeText(mContext, "Has no selected object.",
                           Toast.LENGTH_SHORT).show();
            return;
        }

        objectBase.setSorInfo(mObjectRuntimeInfo.className);
        objectBase.setOutOfViewEnabled(false);

        mVideoRuntime.setListener(objectRuntimelistener);
        mSpenPageDoc.appendObject(objectBase);
        mSpenPageDoc.selectObject(objectBase);
        mSpenSurfaceView.update();
        mSpenSurfaceView.getControl().setContextMenuVisible(false);
        mVideoRuntime.start(objectBase,
            new RectF(0, 0, mSpenPageDoc.getWidth(), mSpenPageDoc.getHeight()),
            mSpenSurfaceView.getPan(), mSpenSurfaceView.getZoomRatio(),
            mSpenSurfaceView.getFrameStartPosition(), mSpenViewLayout);

    }

    SpenObjectRuntime.UpdateListener objectRuntimelistener =
```

```
        new SpenObjectRuntime.UpdateListener() {

            @Override
            public void onCompleted(Object objectBase) {
                if ( mSpenSurfaceView != null ) {
                    SpenControlBase control = mSpenSurfaceView.getControl();
                    if (control != null) {
                        control.setContextMenuVisible(true);
                        mSpenSurfaceView.updateScreenFrameBuffer();
                        mSpenSurfaceView.update();
                    }
                }
                mVideoBtn.setClickable(true);
            }

            @Override
            public void onObjectUpdated(RectF rect, Object objectBase) {
                if ( mSpenSurfaceView != null ) {
                    SpenControlBase control = mSpenSurfaceView.getControl();
                    if(control != null) {
                        control.fit();
                        control.invalidate();
                        mSpenSurfaceView.update();
                    }
                }
            }

            @Override
            public void onCanceled(int state, Object objectBase) {
                if (state == SpenObjectRuntimeInterface.CANCEL_STATE_INSERT) {
                    mSpenPageDoc.removeObject((SpenObjectBase) objectBase);
                    mSpenPageDoc.removeSelectedObject();
                    mSpenSurfaceView.closeControl();
                    mSpenSurfaceView.update();
                } else if (state == SpenObjectRuntimeInterface.CANCEL_STATE_RUN) {
                    mSpenSurfaceView.closeControl();
                    mSpenSurfaceView.update();
                }
                mVideoBtn.setClickable(true);
            }
        };

    private void selectButton(View v) {
        // Enable or disable the button depending on the mode.
        mSelectionBtn.setSelected(false);
        mPenBtn.setSelected(false);
        v.setSelected(true);
    }

    private RectF getRealRect(RectF rect) {
        float panX = mSpenSurfaceView.getPan().x;
        float panY = mSpenSurfaceView.getPan().y;
        float zoom = mSpenSurfaceView.getZoomRatio();
        PointF startPoint = mSpenSurfaceView.getFrameStartPosition();
        RectF realRect = new RectF();
        realRect.set(
            (rect.left - panX) * zoom + startPoint.x,
            (rect.top - panY) * zoom + startPoint.y,
            (rect.right - panX) * zoom + startPoint.x,
```

```
            (rect.bottom - panY) * zoom + startPoint.y
        );
        return realRect;
    }

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

        if(mSpenObjectRuntimeManager != null) {
            if(mVideoRuntime != null) {
                mVideoRuntime.stop(true);
                mSpenObjectRuntimeManager.unload(mVideoRuntime);
            }
            mSpenObjectRuntimeManager.close();
        }

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

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

For more information, see PenSample4_1_SOR.java in PenSample4_1_SOR.

The following sections provide more details on the steps involved in working with SORs.

4.4.1.1 Getting SOR Data

To prepare to create an SOR object:

1. In the `onCreate()` method, create an SpenObjectRuntimeManager instance and call `getObjectRuntimeInfoList()` to get a list (mSpenObjectRuntimeInfoList in the sample) of the available SpenObjectRuntime objects.

```
mSpenObjectRuntimeManager = new SpenObjectRuntimeManager(mActivity);
mSpenObjectRuntimeInfoList =
            new ArrayList<SpenObjectRuntimeInfo>();
mSpenObjectRuntimeInfoList = mSpenObjectRuntimeManager.getObjectRuntimeInfoList();
```

4.4.1.2 Registering a Listener for the Insert Video Button

To handle Insert Video button events:

1. Add an Insert Video button.

Create an `OnClickListener` listener instance for the Insert Video button, mVideoBtnClickListener in the sample, and register it by calling `setOnClickListener()` on the button.

In the `onClick()` method, get the SOR information for “Video” from the list of SOR objects acquired when you initialized your application.

Call `mSpenObjectRuntimeManager.createObjectRuntime()` and pass this SOR information to create a video.

```
public void onClick(View v) {
      mVideoBtn.setClickable(false);
      mSpenSurfaceView.closeControl();
      createObjectRuntime();
}

              .........

void createObjectRuntime() {
        if (mSpenObjectRuntimeInfoList == null
            || mSpenObjectRuntimeInfoList.size() == 0) {
            return;
        }

        try {
            for(SpenObjectRuntimeInfo info : mSpenObjectRuntimeInfoList) {
                if (info.name.equalsIgnoreCase("Video")) {
                    mVideoRuntime =
                        mSpenObjectRuntimeManager.createObjectRuntime(info);

                    mObjectRuntimeInfo = info;
                    startObjectRuntime();

                    return;
                }
            }
        } catch (ClassNotFoundException e) {
              .........
        } catch (InstantiationException e) {
              .........
        } catch (IllegalAccessException e) {
              .........
        } catch (Exception e) {
              .........
        }
}
```

The SOR type for video is SpenObjectRuntimeInterface.TYPE_IMAGE. Create an image object for the SOR, and set the SOR information to the image object by calling `SpenObjectBase.setSorInfo()`.

Call `SpenObjectBase.setOutOfViewEnabled(fasle)` to disable the user to move the object outside SpenView.

Create and register an `SpenControlListener` listener instance for the SOR events and call the following methods:

- `SpenPageDoc.appendObject()` to register the object.
- `SpenPageDoc.selectObject()` to indicate the object has been selected.
- `SpenSurfaceView.update()` to refresh the screen.
- `SpenObjectRuntime.start()` to register the SOR.

```
void startObjectRuntime() {
        if (mVideoRuntime == null) {
            Toast.makeText(mContext,
                "ObjectRuntime is not loaded \n Load Plug-in First !!",
                Toast.LENGTH_SHORT).show();
            return;
        }

        SpenObjectBase objectBase = null;
        switch (mVideoRuntime.getType()) {
            case SpenObjectRuntimeInterface.TYPE_NONE:
                return;
            case SpenObjectRuntimeInterface.TYPE_IMAGE:
                objectBase = new SpenObjectImage();
                break;
            case SpenObjectRuntimeInterface.TYPE_STROKE:
                objectBase = new SpenObjectStroke();
                break;
            case SpenObjectRuntimeInterface.TYPE_CONTAINER:
                objectBase = new SpenObjectContainer();
                break;
            default:
                break;
        }
        if(objectBase == null) {
            Toast.makeText(mContext, "Has no selected object.",
                    Toast.LENGTH_SHORT).show();
            return;
        }

        objectBase.setSorInfo(mObjectRuntimeInfo.className);
        objectBase.setOutOfViewEnabled(false);

        mVideoRuntime.setListener(objectRuntimelistener);
        mSpenPageDoc.appendObject(objectBase);
        mSpenPageDoc.selectObject(objectBase);
        mSpenSurfaceView.update();
        mSpenSurfaceView.getControl().setContextMenuVisible(false);
        mVideoRuntime.start(objectBase,
            new RectF(0, 0, mSpenPageDoc.getWidth(), mSpenPageDoc.getHeight()),
            mSpenSurfaceView.getPan(), mSpenSurfaceView.getZoomRatio(),
            mSpenSurfaceView.getFrameStartPosition(), mSpenViewLayout);

  }
```

4.4.1.3 Creating a Context Menu in a Control

To add a context menu in your application:

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

In the `onCreated()` callback method, which is called when there is a control in the View area, add a “Run” menu item to the context menu that appears when the selected object contains SOR information.

```
public boolean onCreated(ArrayList<SpenObjectBase> objectList,
            ArrayList<Rect> relativeRectList,
            ArrayList<SpenContextMenuItemInfo> menu,
            ArrayList<Integer> styleList, int pressType, PointF point) {
            if (objectList == null) {
                return false;
            }
            // Display the context menu if any SOR information is found.
            if (objectList.get(0).getSorInfo() != null) {
                menu.add(new SpenContextMenuItemInfo(
                                    CONTEXT_MENU_RUN_ID, "Run", true));
                return true;
            }
            return true;
}
```

4.4.1.4 Handling Context Menu Events in a Control

To handle context menu events:

1. In the `onMenuSelected()` callback method, which is called when a menu item is selected from the context menu in a control, do the following:

- Close the context menu.
- To prevent the SOR from resizing while it is being played, call `SpenSurfaceView.getControl().setStyle(SpenControlBase.STYLE_BORDER_STATIC)`.
- Register an SOR event listener.
- Call `start()` to play the video.

```
public boolean onMenuSelected(
       ArrayList<SpenObjectBase> objectList, int itemId) {
       if (objectList == null) {
           return true;
       }
```

```
        if (itemId == CONTEXT_MENU_RUN_ID) {
            SpenObjectBase object = objectList.get(0);
            mSpenSurfaceView.getControl().setContextMenuVisible(false);

            mSpenSurfaceView.getControl().setStyle(SpenControlBase.STYLE_BORDER_STATIC);

            // Set up the listener and make it play.
            mVideoRuntime.setListener(objectRuntimelistener);
            mVideoRuntime.start(object, getRealRect(object.getRect()),
                    mSpenSurfaceView.getPan(), mSpenSurfaceView.getZoomRatio(),
                    mSpenSurfaceView.getFrameStartPosition(), mSpenViewLayout);

            mSpenSurfaceView.update();
        }
        return false;
}
```

4.4.1.5 Registering an SOR Event Listener

To handle SOR events in your application:

1. Create an `objectRuntimelistener` listener instance for SOR events.
2. Call `SpenObjectRuntime.UpdateListener()` to register the listener before playing the video.

In the `onCompleted()` callback method, which is called when the SOR is done, call the following methods:

- `SpenControlBase.setContextMenuVisible()` to make the context menu re-appear.
- `SpenSurfaceView.updateScreenFrameBuffer()` to update the screen frame buffer.
- `SpenSurfaceView.update()` to refresh the screen.

In the `onObjectUpdated()` callback method, which is called for update events, resize the SpenControl to fit to the updated object and refresh the screen.

In the `onCanceled()` callback method, which is called for cancellation events, do the following:

- When an event to cancel the registration of a new SOR occurs, delete the corresponding object from SpenPageDoc, close the control, and refresh the screen.
- When an event to cancel an SOR action occurs, close the control and refresh the screen.

```
SpenObjectRuntime.UpdateListener objectRuntimelistener =
        new SpenObjectRuntime.UpdateListener() {

            @Override
            public void onCompleted(Object objectBase) {
                if ( mSpenSurfaceView != null ) {
                    SpenControlBase control = mSpenSurfaceView.getControl();
                    if (control != null) {
                        control.setContextMenuVisible(true);
                        mSpenSurfaceView.updateScreenFrameBuffer();
```

```
                    mSpenSurfaceView.update();
                }
            }
            mVideoBtn.setClickable(true);
        }

        @Override
        public void onObjectUpdated(RectF rect, Object objectBase) {
            if ( mSpenSurfaceView != null ) {
                SpenControlBase control = mSpenSurfaceView.getControl();
                if(control != null) {
                    control.fit();
                    control.invalidate();
                    mSpenSurfaceView.update();
                }
            }
        }

        @Override
        public void onCanceled(int state, Object objectBase) {
            if (state == SpenObjectRuntimeInterface.CANCEL_STATE_INSERT) {
                mSpenPageDoc.removeObject((SpenObjectBase) objectBase);
                mSpenPageDoc.removeSelectedObject();
                mSpenSurfaceView.closeControl();
                mSpenSurfaceView.update();
            } else if (state == SpenObjectRuntimeInterface.CANCEL_STATE_RUN) {
                mSpenSurfaceView.closeControl();
                mSpenSurfaceView.update();
            }
            mVideoBtn.setClickable(true);
        }
};
```

4.4.2. Working with SOR Lists

The sample application implements the following features in the sample application for Adding Video Objects:

- Add SOR button to add an SOR.
- When the Add SOR button is clicked, the sample application shows the list of available SORs returned by the `SpenObjectRuntimeManager.getObjectRuntimeInfoList()` method when you initialize your application.
- The sample application then inserts the SOR selected from the list on the screen.

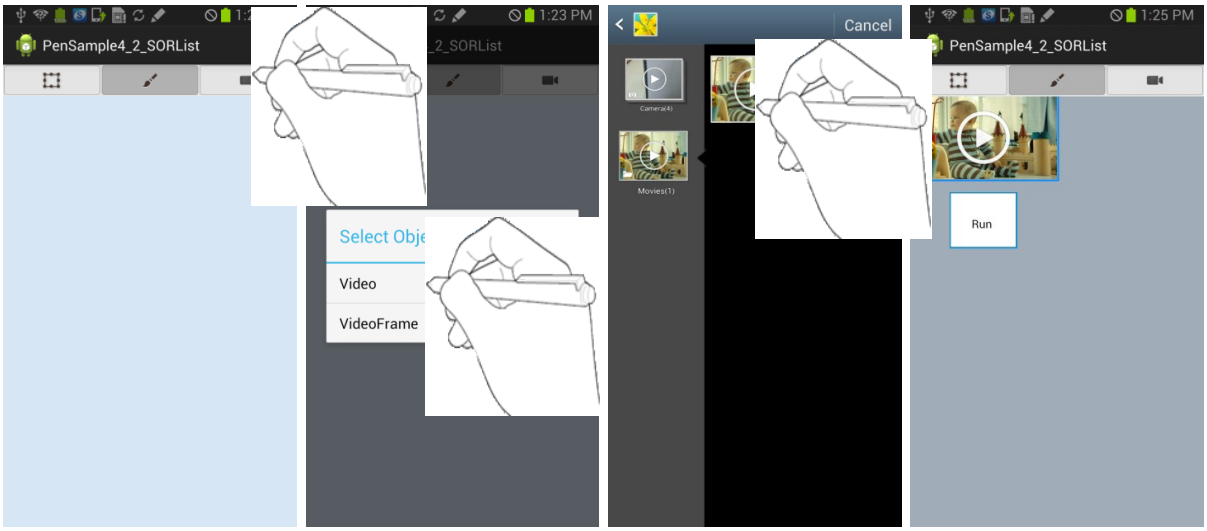


Figure 27: SOR list

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

              .........

        mVideoBtn = (ImageView) findViewById(R.id.videoBtn);
        mVideoBtn.setOnClickListener(mVideoBtnClickListener);

        selectButton(mPenBtn);

        // Set up ObjectRuntimeManager.
        mSpenObjectRuntimeManager = new SpenObjectRuntimeManager(mActivity);
        mSpenObjectRuntimeInfoList =
            new ArrayList<SpenObjectRuntimeInfo>();
        mSpenObjectRuntimeInfoList =
            mSpenObjectRuntimeManager.getObjectRuntimeInfoList();


              .........

private final OnClickListener mVideoBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                if(mObjectRuntime != null)
                    mObjectRuntime.stop(true);
                createObjectRuntime();
            }
};

              .........

void createObjectRuntime() {
        if (mSpenObjectRuntimeInfoList == null
```

```
            || mSpenObjectRuntimeInfoList.size() == 0) {
            return;
        }

        String objectRuntimeInfoList[] = new String[mSpenObjectRuntimeInfoList.size()];
        for (int i = 0; i < mSpenObjectRuntimeInfoList.size(); i++) {
            objectRuntimeInfoList[i] = mSpenObjectRuntimeInfoList.get(i).name;
        }

        new AlertDialog.Builder(mContext)
            .setTitle("Select Object Runtime")
            .setItems(objectRuntimeInfoList, new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    try {
                        SpenObjectRuntimeInfo                    info                    =
mSpenObjectRuntimeInfoList.get(which);
                        mObjectRuntime =
                            mSpenObjectRuntimeManager.createObjectRuntime(info);

                        mObjectRuntimeInfo = info;
                        startObjectRuntime();

                        return;
                    } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                        Toast.makeText(mContext, "ObjectRuntimeInfo class not found.",
                            Toast.LENGTH_SHORT).show();
                    } catch (InstantiationException e) {
                        e.printStackTrace();

                        Toast.makeText(mContext,
                          " Failed to access the ObjectRuntimeInfo constructor.",
                            Toast.LENGTH_SHORT).show();
                    } catch (IllegalAccessException e) {
                        e.printStackTrace();

                        Toast.makeText(mContext,
                          " Failed to access the ObjectRuntimeInfo field or method.",
                            Toast.LENGTH_SHORT).show();
                    } catch (Exception e) {
                        e.printStackTrace();
                        Toast.makeText(mContext, "ObjectRuntimeInfo is not loaded.",
                            Toast.LENGTH_SHORT).show();
                    }
                }
            }).show();
}

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

For more information, see PenSample4_2_SORList.java in PenSample4_2_SORList.

The following sections provide more details on the steps involved in working with SOR lists.

4.4.2.1 Showing SOR Lists

To show an SOR list in your application:

1. In the `onCreate()` method, create an `SpenObjectRuntimeManager` instance to call `getObjectRuntimeInfoList()` to get the list of available SpenObjectRuntime objects.
2. Extract the SOR names from the list to create a dialog showing the names in a list.

In the `onClick()` callback method, which is called when users select an SOR, get the SpenObjectRuntimeInfo of the selected object. Call `mSpenObjectRuntimeManager.createObjectRuntime()` and pass the information to create the selected SOR.

```
if (mSpenObjectRuntimeInfoList == null || mSpenObjectRuntimeInfoList.size() == 0) {
       return;
}

String objectRuntimeInfoList[] = new String[mSpenObjectRuntimeInfoList.size()];
for (int i = 0; i < mSpenObjectRuntimeInfoList.size(); i++) {
       objectRuntimeInfoList[i] = mSpenObjectRuntimeInfoList.get(i).name;
}

new AlertDialog.Builder(mContext)
       .setTitle("Select Object Runtime")
       .setItems(objectRuntimeInfoList, new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    try {
                        SpenObjectRuntimeInfo info =
                             mSpenObjectRuntimeInfoList.get(which);
                        mObjectRuntime =
                            mSpenObjectRuntimeManager.createObjectRuntime(info);
```

4.5. Using Advanced Pen Features

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

- Smart Scroll
- Smart Zoom
- Translucent View

4.5.1. Using Smart Scroll

You can use Smart Scroll to enable automatic horizontal or vertical scrolling when a S pen hovers near the edge of the screen.

The sample application implements the following features:

- Smart Scroll button for turning Smart Scroll on and off.
- Listener for button click events.
- Hover zone where Smart Scroll is available.
- Enabling and disabling horizontal and vertical scrolling.

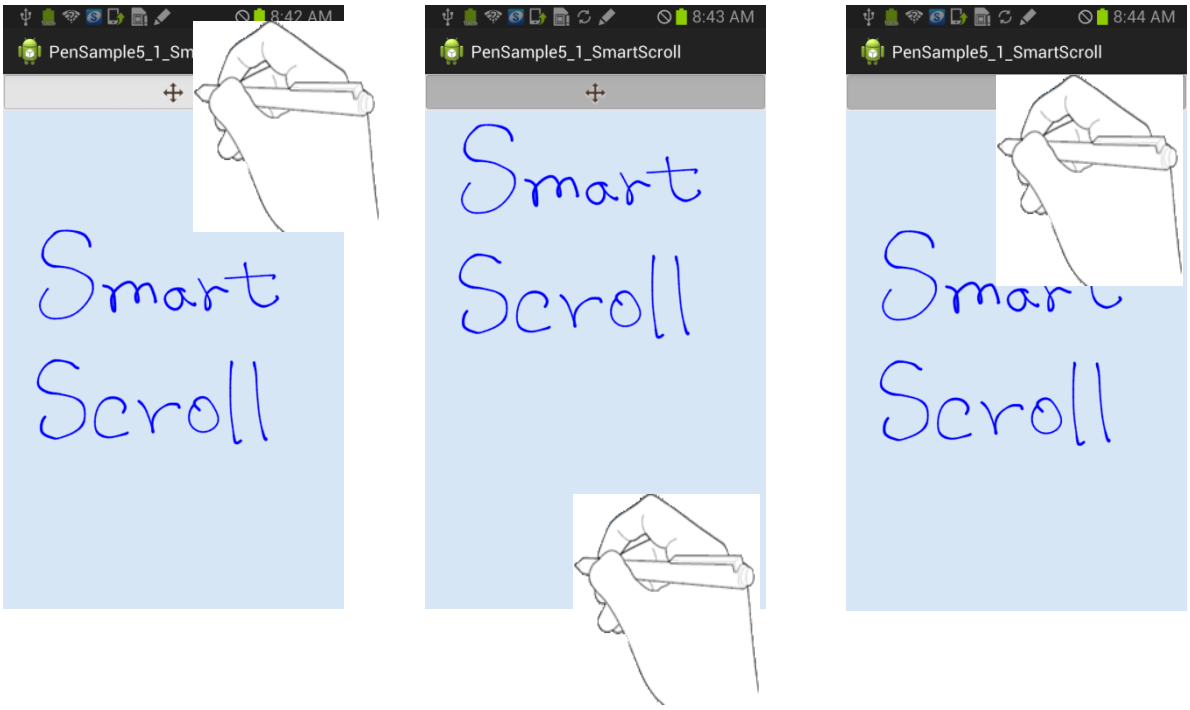


Figure 28: Smart Scroll

```
public class PenSample5_1_SmartScroll extends Activity {

    private Context mContext;
    private SpenNoteDoc mSpenNoteDoc;
    private SpenPageDoc mSpenPageDoc;
    private SpenSurfaceView mSpenSurfaceView;

    private ImageView mSmartScrollBtn;

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

        // Initialize Pen.

        boolean isSpenFeatureEnabled = false;
        Spen spenPackage = new Spen();
        try {
            spenPackage.initialize(this);

            isSpenFeatureEnabled =
                spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
        } catch (SsdkUnsupportedException e) {
```

```
            if( SDKUtils.processUnsupportedException(this, e) == true) {
                return;
            }
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Pen.",
                Toast.LENGTH_SHORT).show();
            e1.printStackTrace();
            finish();
        }

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

        // Create Pen View.
        mSpenSurfaceView = new SpenSurfaceView(mContext);
        if (mSpenSurfaceView == null) {
            Toast.makeText(mContext, "Cannot create new SpenSurfaceView.",
                Toast.LENGTH_SHORT).show();
            finish();
        }
        spenViewLayout.addView(mSpenSurfaceView);

        // Get the dimensions 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 and get an instance to use as a member variable.
        mSpenPageDoc = mSpenNoteDoc.appendPage();
        mSpenPageDoc.setBackgroundColor(0xFFD6E6F5);
        mSpenPageDoc.clearHistory();
        // Set the PageDoc in the View.
        mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);

        initPenSettingInfo();

        // Set up the button.
        mSmartScrollBtn = (ImageView) findViewById(R.id.smartScrollBtn);
        mSmartScrollBtn.setOnClickListener(mSmartScrollBtnClickListener);
    }

    if(isSpenFeatureEnabled == false) {
        mSpenSurfaceView.setToolTypeAction(SpenSurfaceView.TOOL_FINGER,
                SpenSurfaceView.ACTION_STROKE);
        Toast.makeText(mContext,
                       "Device does not support S pen. \n You can draw strokes with
                        your finger",
```

```
                        Toast.LENGTH_SHORT).show();
    }

    private void initPenSettingInfo() {
        // Initialize the settings for the pen.
        SpenSettingPenInfo penInfo = new SpenSettingPenInfo();
        penInfo.color = Color.BLUE;
        penInfo.size = 10;
        mSpenSurfaceView.setPenSettingInfo(penInfo);
    }

    private void setSmartScroll(boolean enable) {

        // Define the region for Smart Scroll.
        int width, height, w1, h1, w9, h9;
        width = mSpenSurfaceView.getWidth();
        height = mSpenSurfaceView.getHeight();
        w1 = (int) (width * 0.1);
        h1 = (int) (height * 0.1);
        w9 = (int) (width * 0.9);
        h9 = (int) (height * 0.9);

        // Define the region for horizontal Smart Scroll.
        Rect leftRegion = new Rect(0, 0, w1, height);
        Rect rightRegion = new Rect(w9, 0, width, height);
        mSpenSurfaceView.setHorizontalSmartScrollEnabled(enable,
            leftRegion, rightRegion, 500, 10);

        // Define the region for vertical Smart Scroll.
        Rect topRegion = new Rect(0, 0, width, h1);
        Rect bottomRegion = new Rect(0, h9, width, height);
        mSpenSurfaceView.setVerticalSmartScrollEnabled(enable,
            topRegion, bottomRegion, 500, 10);
    }

    private final OnClickListener mSmartScrollBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                boolean isSmartScrollEnabled =
                    !mSpenSurfaceView.isVerticalSmartScrollEnabled();
                mSmartScrollBtn.setSelected(isSmartScrollEnabled);

                setSmartScroll(isSmartScrollEnabled);
            }
        };

    @Override
    public void onConfigurationChanged(Configuration newConfig) {

        super.onConfigurationChanged(newConfig);
        if(mSmartScrollBtn.isSelected() == false) {
            return;
        }

        ViewTreeObserver observer = mSpenSurfaceView.getViewTreeObserver();
        observer.addOnGlobalLayoutListener(new OnGlobalLayoutListener() {

            @SuppressLint("NewApi")
```

```
            @SuppressWarnings("deprecation")
            @Override
            public void onGlobalLayout() {
                setSmartScroll(true);

                if (Build.VERSION.SDK_INT >= 16) {
                    mSpenSurfaceView.getViewTreeObserver().
                        removeOnGlobalLayoutListener(this);
                } else {
                    // This method was deprecated in API level 16.
                    mSpenSurfaceView.getViewTreeObserver().
                        removeGlobalOnLayoutListener (this);
                }
            }
        });
    }


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

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

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

For more information, see PenSample5_1_SmartScroll.java in PenSample5_1_SmartScroll.

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

4.5.1.1 Registering a Listener for the Smart Scroll Button

To handle Small Scroll button events in your application:

1. Create a Smart Scroll button.
2. Create an `OnClickListener` listener instance for the Smart Scroll button, mSmartScrollBtnClickListener in the sample, and register it by calling `setOnClickListener()` on the button.
3. In the onClick() method, call `SpenSurfaceView.isVerticalSmartScrollEnabled()` to check if vertical Smart Scroll is enabled. You can enable or disable the button by using the logical NOT

operator (!) with the return value of `isVerticalSmartScrollEnabled()` to enable or disable vertical and horizontal scrolling when users click the button.

```
boolean isSmartScrollEnabled = !mSpenSurfaceView.isVerticalSmartScrollEnabled();
mSmartScrollBtn.setSelected(isSmartScrollEnabled);
```

Set the region where Smart Scroll is available. Typically, Pen enables Smart Scroll when it generates a hovering event in the 10% of the width from the left or right edge and 10% of the width from the top or bottom edge.

```
int width, height, w1, h1, w9, h9;
width = mSpenSurfaceView.getWidth();
height = mSpenSurfaceView.getHeight();
w1 = (int) (width * 0.1);
h1 = (int) (height * 0.1);
w9 = (int) (width * 0.9);
h9 = (int) (height * 0.9);
```

Set the area for horizontal Smart Scroll.

Call `SpenSurfaceView.setHorizontalSmartScrollEnabled()` and pass the return value of `isSmartScrollEnabled()`, the area where Smart Scroll is enabled, the response time, and the speed after a hover event.

For vertical Smart Scroll, set the area for vertical Smart Scroll. Call `SpenSurfaceView.setVerticalSmartScrollEnabled()` and pass the corresponding input variables.

```
// Set up the HorizontalSmartScroll.
Rect leftRegion = new Rect(0, 0, w1, height);
Rect rightRegion = new Rect(w9, 0, width, height);
mSpenSurfaceView.setHorizontalSmartScrollEnabled(enable,
                    leftRegion, rightRegion, 500, 10);

// Set up the VerticalSmartScroll.
Rect topRegion = new Rect(0, 0, width, h1);
Rect bottomRegion = new Rect(0, h9, width, height);
mSpenSurfaceView.setVerticalSmartScrollEnabled(enable,
                    topRegion, bottomRegion, 500, 10);
```

4.5.1.2 Redefining a Smart Scroll Region Based on Page Orientation

To redefine the Smart Scroll region when the page orientation (landscape mode and portrait mode) changes, override the `onConfigurationChanged()` method of the activity.

```
@Override
```

```
public void onConfigurationChanged(Configuration newConfig) {

        super.onConfigurationChanged(newConfig);
        if(mSmartScrollBtn.isSelected() == false) {
            return;
        }

        ViewTreeObserver observer = mSpenSurfaceView.getViewTreeObserver();
        observer.addOnGlobalLayoutListener(new OnGlobalLayoutListener() {

            @SuppressLint("NewApi")
            @SuppressWarnings("deprecation")
            @Override
            public void onGlobalLayout() {
                setSmartScroll(true);

                if (Build.VERSION.SDK_INT >= 16) {
                    mSpenSurfaceView.getViewTreeObserver().
                        removeOnGlobalLayoutListener(this);
                } else {
                    // This method was deprecated in API level 16.
                    mSpenSurfaceView.getViewTreeObserver().
                        removeGlobalOnLayoutListener (this);
                }
            }
        });
}
```

4.5.2. Using Smart Zoom

You can use Smart Zoom in your application to implement auto-zoom in features when a S pen hovers over a specific region on the screen.

The sample application implements the following features:

- Smart Zoom button to zoom in when a S pen hovers over the specified Smart Zoom region.
- Listener for button click events.
- Enabling and disabling Smart Zoom when the button is clicked.
- Setting the Smart Zoom region.

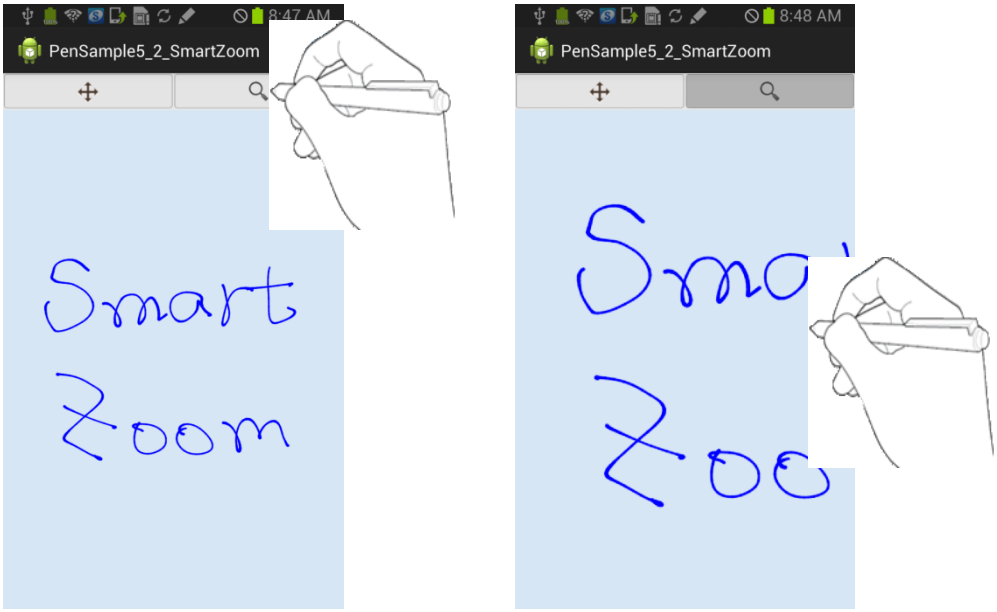


Figure 29: Smart Zoom

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

              .........

        mSmartZoomBtn = (ImageView) findViewById(R.id.smartZoomBtn);
        mSmartZoomBtn.setOnClickListener(mSmartZoomBtnClickListener);
}

              .........

private void setSmartScale(boolean enable) {

    // Define the region for Smart Zoom.
    int width, height, w1, h1, w9, h9;
    width = mSpenSurfaceView.getWidth();
    height = mSpenSurfaceView.getHeight();
    w1 = (int) (width * 0.1);
    h1 = (int) (height * 0.1);
    w9 = (int) (width * 0.9);
    h9 = (int) (height * 0.9);

    // The settings for Smart Scale.
    Rect zoomRegion = new Rect(w1, h1, w9, h9);
    mSpenSurfaceView.setSmartScaleEnabled(enable, zoomRegion, 8, 500, 1.5f);
}

          .........

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

```
        public void onClick(View v) {
            boolean isSmartScaleEnabled =
                    !mSpenSurfaceView.isSmartScaleEnabled();
            mSmartZoomBtn.setSelected(isSmartScaleEnabled);

            setSmartScale(isSmartScaleEnabled);
        }
    };

@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    if(mSmartScrollBtn.isSelected() == false
            && mSmartZoomBtn.isSelected() == false) {
        return;
    }

    ViewTreeObserver observer = mSpenSurfaceView.getViewTreeObserver();
    observer.addOnGlobalLayoutListener(new OnGlobalLayoutListener() {

        @SuppressLint("NewApi")
        @SuppressWarnings("deprecation")
        @Override
        public void onGlobalLayout() {
            if(mSmartScrollBtn.isSelected() == true) {
                setSmartScroll(true);
            }
            if(mSmartZoomBtn.isSelected() == true){
                setSmartScale(true);
            }
            if (Build.VERSION.SDK_INT >= 16) {
                mSpenSurfaceView.getViewTreeObserver().
                    removeOnGlobalLayoutListener(this);
            } else {
                // This method was deprecated in API level 16.
                mSpenSurfaceView.getViewTreeObserver().
                    removeGlobalOnLayoutListener (this);
            }
        }
    });
}

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

For more information, see PenSample5_2_SmartZoom.java in PenSample5_2_SmartZoom.

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

4.5.2.1 Registering a Listener for the Smart Zoom Button

To handle Smart Zoom button events in your application:

1. Create a Smart Zoom button.

2. Create an `OnClickListener` listener instance for the Smart Zoom button, mSmartZoomBtnClickListener in the sample, and register it by calling `setOnClickListener()` on the button.

3. In the `onClick()` method, call `SpenSurfaceView.isSmartScaleEnabled()` to check whether Smart Zoom is enabled. You can enable or disable the button by using the logical NOT operator (!) with the return value of `isSmartScaleEnabled()` to turn Smart Zoom on or off when the button is clicked.

```
boolean isSmartScaleEnabled =!mSpenSurfaceView.isSmartScaleEnabled();
mSmartZoomBtn.setSelected(isSmartScaleEnabled);
```

Set the region where Smart Zoom is available. The sample application enables Smart Zoom when it generates a hover event outside the 10% of the width from the left or right edge and 10% of the width from the top or bottom edge.

Call `SpenSurfaceView.setSmartScaleEnabled()` and pass the return value of `isSmartScaleEnabled()`, the response time after the hover event, and the zoom scale.

```
int width, height, w1, h1, w9, h9;
width = mSpenSurfaceView.getWidth();
height = mSpenSurfaceView.getHeight();
w1 = (int) (width * 0.1);
h1 = (int) (height * 0.1);
w9 = (int) (width * 0.9);
h9 = (int) (height * 0.9);

// Set SmartScale.
Rect zoomRegion = new Rect(w1, h1, w9, h9);
mSpenSurfaceView.setSmartScaleEnabled(enable,zoomRegion, 8, 500, 1.5f);
```

4.5.3. Displaying Translucent Pen Views

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

The sample application implements the following features:

- Simple View button for creating an SpenSimpleView instance with a transparent background.
- Stroke creation and saving in an image file with a specified name.

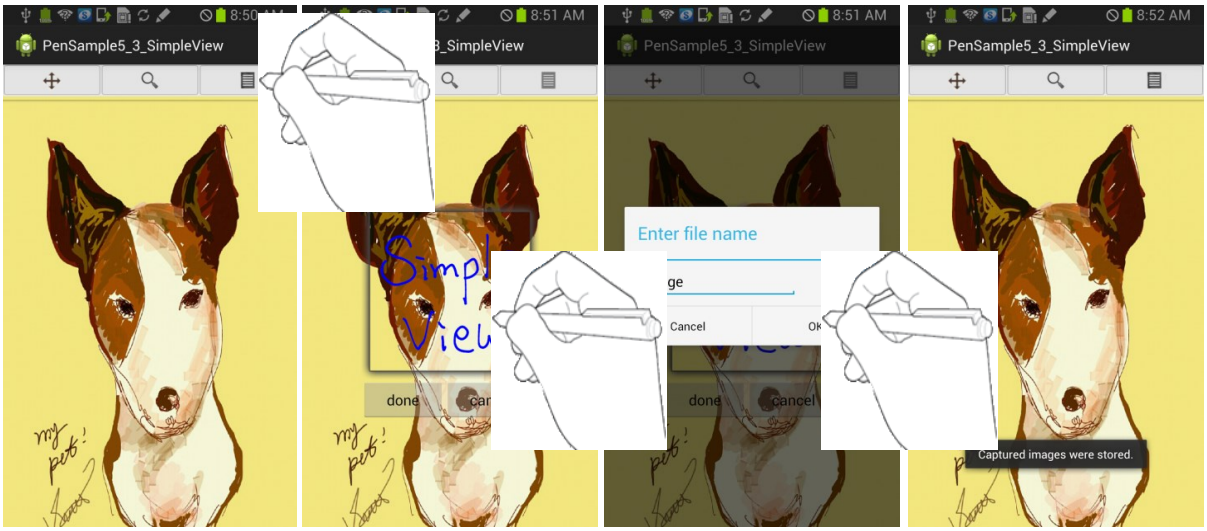


Figure 30: Simple View

```
public class PenSample5_3_SimpleView extends Activity {

	private Context mContext;
	private SpenNoteDoc mSpenNoteDoc;
	private SpenPageDoc mSpenPageDoc;
	private SpenSurfaceView mSpenSurfaceView;
	private SpenSimpleView mSpenSimpleView;
	private RelativeLayout mSpenSimpleViewContainer;

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

	@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) {
                        inputFileName();
                    }
                });

                Button cancelBtn = (Button) findViewById(R.id.cancel_btn);
                cancelBtn.setOnClickListener(new OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        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) {
                }
            });
        AlertDialog 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(mSpenSurfaceView != null) {
                mSpenSurfaceView.close();
                mSpenSurfaceView = null;
            }

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

For more information, see PenSample5_3_SimpleView.java in PenSample5_3_SimpleView.

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

4.5.3.1 Setting Background Images

To set a background image in your application:

1. Specify the background image of the main SpenSurfaceView 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 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.

4.5.3.2 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()` on the 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();
```

4.5.3.3 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();
}
```

4.5.3.4 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;
}
```

4.5.4. Using Temporary Stroke Mode

You can use the Temporary Stroke mode in your application, which scales the user input strokes down by 50% and moves the strokes to the upper part of the viewport.

The sample application implements the following features:

- When the application starts, it calls the `SpenSurfaceView.startTemporaryStroke()` method to process the user input strokes in Temporary Stroke mode. Temporary strokes are drawn on the viewport but are not saved to SpenPageDoc.
- If there is no user input for a second, it calls `SpenSurfaceView.stopTemporaryStroke()` to turn the Temporary Stroke mode off.
- The application calls `SpenSurfaceView.getTemporaryStroke()` to get the user input strokes, and scales the strokes down by 50%.
- The sample application registers the strokes as an object in the current page.

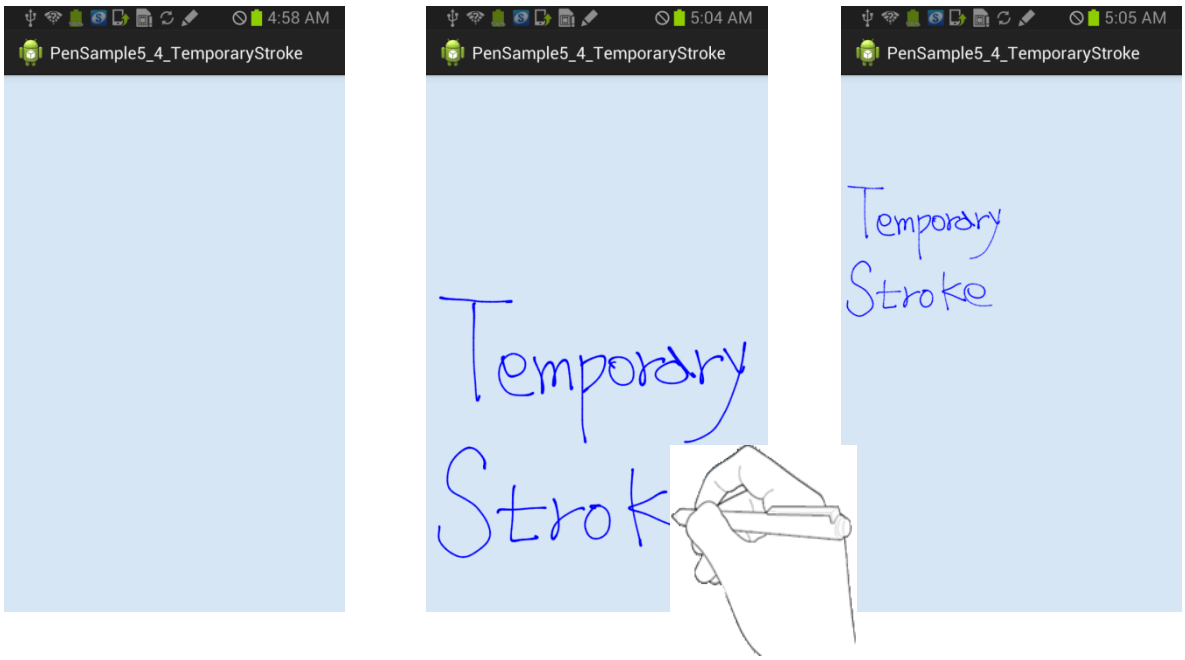


Figure 31: Temporary Stroke

```
public class PenSample5_4_TemporaryStroke extends Activity {

    private Context mContext;
    private SpenNoteDoc mSpenNoteDoc;
    private SpenPageDoc mSpenPageDoc;
    private SpenSurfaceView mSpenSurfaceView;

    private Handler mStrokeHandler;

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

        // Initialize Pen.
        boolean isSpenFeatureEnabled = false;
        Spen spenPackage = new Spen();
        try {
            spenPackage.initialize(this);

            isSpenFeatureEnabled =
                        spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
        } catch (SsdkUnsupportedException e) {
            if( SDKUtils.processUnsupportedException(this, e) == true) {
                return;
            }
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Pen.",
                Toast.LENGTH_SHORT).show();
            e1.printStackTrace();
            finish();
        }

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

        // Get the dimensions of the 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();
        }
        // 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.
        mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);

        initPenSettingInfo();
        // Register the listener.
        mSpenSurfaceView.setTouchListener(mPenTouchListener);

        mSpenSurfaceView.startTemporaryStroke();
        if(isSpenFeatureEnabled == false) {
            mSpenSurfaceView.setToolTypeAction(SpenSurfaceView.TOOL_FINGER,
                                    SpenSurfaceView.ACTION_STROKE);
            Toast.makeText(mContext,
                "Device does not support S pen. \n You can draw strokes with your
finger",
                 Toast.LENGTH_SHORT).show();
        }
    }

    private void initPenSettingInfo() {
        // Initialize pen settings.
        SpenSettingPenInfo penInfo = new SpenSettingPenInfo();
        penInfo.color = Color.BLUE;
        penInfo.size = 10;
        mSpenSurfaceView.setPenSettingInfo(penInfo);
    }

    private final SpenTouchListener mPenTouchListener = new SpenTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                // When ACTION_DOWN occurs before mStrokeRunnable is set
                // in a queue, the mStrokeRunnable that waits is removed.
                if (mStrokeHandler != null) {
                    mStrokeHandler.removeCallbacks(mStrokeRunnable);
                    mStrokeHandler = null;
                }
                break;
            case MotionEvent.ACTION_UP:
                // Generate Handler to put mStrokeRunnable in a queue when it
                // takes 1000 milliseconds after ACTION_UP occurred.
                mStrokeHandler = new Handler();
                mStrokeHandler.postDelayed(mStrokeRunnable, 1000);
                break;
            }
            return true;
        }
    };
```

```
    private final Runnable mStrokeRunnable = new Runnable() {
        @Override
        public void run() {
            // Get TemporaryStroke to resize the object by 1/2.
            ArrayList<SpenObjectStroke> objList = mSpenSurfaceView
                .getTemporaryStroke();

            for(SpenObjectStroke obj : objList) {
                RectF rect = obj.getRect();
                rect.set(rect.left / 2, rect.top / 2,
                    rect.right / 2, rect.bottom / 2);
                obj.setRect(rect, false);
                mSpenPageDoc.appendObject(obj);
            }
            mSpenSurfaceView.stopTemporaryStroke();
            mSpenSurfaceView.startTemporaryStroke();
            mSpenSurfaceView.update();
        }
    };

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

        if (mStrokeHandler != null) {
            mStrokeHandler.removeCallbacks(mStrokeRunnable);
            mStrokeHandler = null;
        }

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

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

For more information, see PenSample5_4_TemporaryStroke.java in PenSample5_4_TemporaryStroke.

4.5.4.1 Activating the Temporary Stroke Mode

To create a Temporary Stroke board:

1. In the `onCreate()` method, call `SpenSurfaceView.startTemporaryStroke()` to activate the Temporary Stroke mode.

```
mSpenPageDoc = mSpenNoteDoc.appendPage();
mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);
```

```
mSpenSurfaceView.startTemporaryStroke();
```

4.5.4.2 Registering a Touch Event Listener

To handle touch events in your application:

1. Create an `SpenTouchListener` instance, mPenTouchListener in the sample, for touch events on SpenSurfaceView and register it by calling `SpenSurfaceView.setTouchListener()`.

In the `onTouch()` method, do the following:

- When an ACTION_DOWN event occurs, call `Handler.removeCallbacks()` to remove any pending Runnable from the queue.
- If no stroke input is attempted for one second after an ACTION_UP event, generate a handler for the `postDelayed()` call, which executes a registered Runnable.

```
public boolean onTouch(View v, MotionEvent event) {
      switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                // When ACTION_DOWN occurs before mStrokeRunnable is set
                // to a queue, the mStrokeRunnable that waits is removed.
                if (mStrokeHandler != null) {
                    mStrokeHandler.removeCallbacks(mStrokeRunnable);
                    mStrokeHandler = null;
                }
                break;
            case MotionEvent.ACTION_UP:
                // Generate Handler to put mStrokeRunnable in a queue when it takes
                // 1000 milliseconds after ACTION_UP occurred.
                mStrokeHandler = new Handler();
                mStrokeHandler.postDelayed(mStrokeRunnable, 1000);
                break;
      }
      return true;
}
```

4.5.4.3 Registering Temporary Stroke Data as an Object

To register the Temporary Stroke data as an object:

1. Call `SpenObjectStroke.setRect()` to set the position of the stroke data that is fetched by SpenSurfaceView.getTemporaryStroke() and cut the data in half.

Call `SpenPageDoc.appendObject()` to register the stroke data as an object of the current page.

Call `SpenSurfaceView.stopTemporaryStroke()` to disable the Temporary Stroke mode.

Call `SpenSurfaceView.startTemporaryStroke()` to activate a new Temporary Stroke mode.

Call `SpenSurfaceView.update()` to refresh the viewport.

```
public void run() {
            // Get TemporaryStroke to resize the object by 1/2.
```

```
            ArrayList<SpenObjectStroke> objList = mSpenSurfaceView
                .getTemporaryStroke();

            for(SpenObjectStroke obj : objList) {
                RectF rect = obj.getRect();
                rect.set(rect.left / 2, rect.top / 2,
                    rect.right / 2, rect.bottom / 2);
                obj.setRect(rect, false);
                mSpenPageDoc.appendObject(obj);
            }
            mSpenSurfaceView.stopTemporaryStroke();
            mSpenSurfaceView.startTemporaryStroke();
            mSpenSurfaceView.update();
}
```

4.5.4.4 Preventing Memory Leaks

To prevent memory leaks:

1. Call `Handler.removeCallbacks()` to purge any pending callback methods in the queue.
2. Call `SpenSurfaceView.close()` and `SpenNoteDoc.close()` to close the SpenSurfaceView and SpenNoteDoc instances.

```
if (mStrokeHandler != null) {
       mStrokeHandler.removeCallbacks(mStrokeRunnable);
       mStrokeHandler = null;
}

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

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

4.5.5. Working Only with Pen

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

The sample application implements the following features:

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

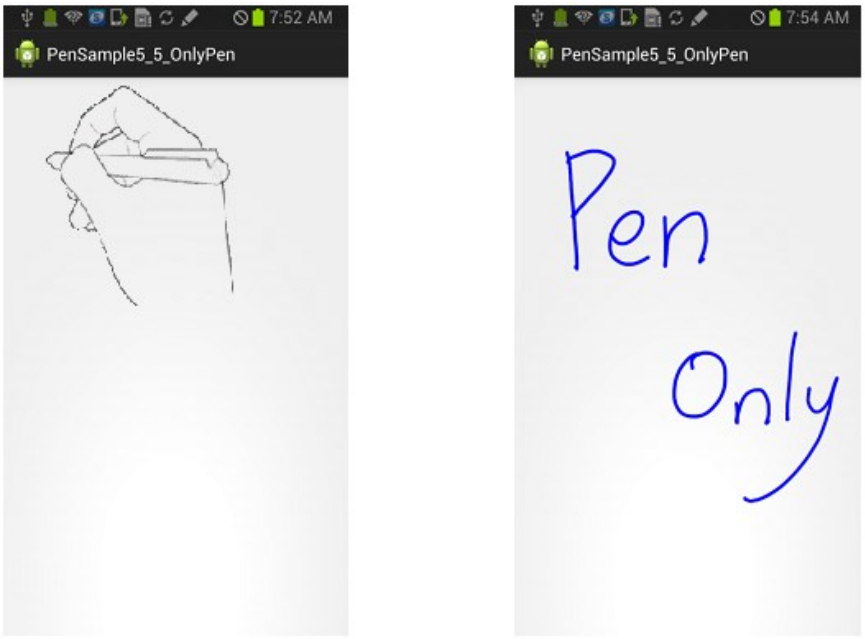


Figure 32: Pen drawing on Android View

```
public class PenSample5_5_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 (SsdkUnsupportedException e) {
            if( SDKUtils.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);
    }

    protected class MyView extends View {

        private RectF bitmapRect = new RectF();

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

        @Override
        public boolean onTouchEvent(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 enable pen 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, see PenSample5_2_OnlyPen.java in PenSample5_5_OnlyPen.

4.5.5.1 Loading Pen Plug-ins

To load a pen plug-in:

1. Create an SpenPenManager instance.

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

Select an appropriate pen 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 supports the following pre-loaded pen 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);
```

4.5.5.2 Linking the Pen 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.

```
protected class 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);
        }
```

4.5.5.3 Handling Drawing

To handle touch events:

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

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

In the `onDraw()` method, call `canvas.drawBitmap()` to display the bitmap drawn by the pen on the canvas.

```
public boolean onTouchEvent(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;
}
```

4.5.5.4 Preventing Memory Leaks

To prevent memory leaks:

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

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

4.5.6. Using Text Recognition Plug-ins

You can use Pen for text recognition and use pre-loaded recognition engines.

The sample application implements the following features:

- It creates an SpenSurfaceView instance.
- The application calls `SpenTextRecognitionManager.createRecognition()` to load a text recognition plug-in.
- It adds stroke objects to SpenSurfaceView. When the Selection Tool button is pressed, and the stroke object to be recognized is selected, the sample application calls `SpenTextRecognition.request()` to request text recognition.
- When the text recognition output is calculated, the sample application replaces the selected stroke object with the SpenObjectTextBox recognized as a textual component.

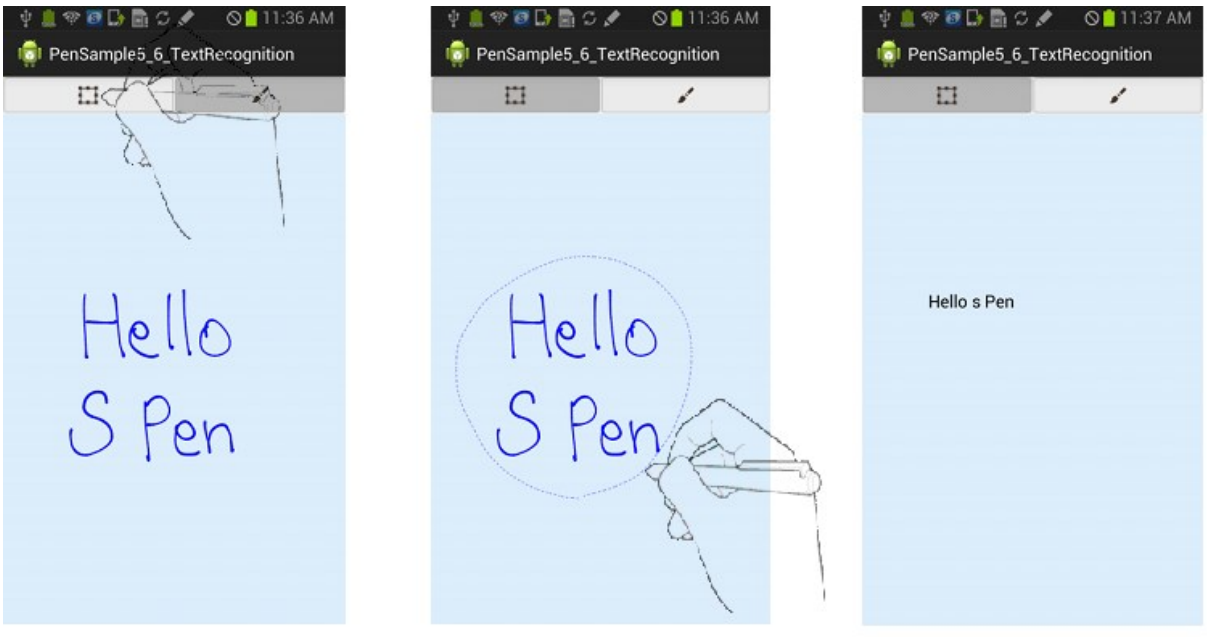


Figure 33: Text recognition

```
public class PenSample5_6_TextRecognition extends Activity {

    private Context mContext;
```

```
    private SpenNoteDoc mSpenNoteDoc;
    private SpenPageDoc mSpenPageDoc;
    private SpenSurfaceView mSpenSurfaceView;
    RelativeLayout mSpenViewLayout;

    private SpenTextRecognition mTextRecognition = null;
    private SpenTextRecognitionManager mSpenTextRecognitionManager = null;
    private boolean mIsProcessingRecognition = false;

    private ImageView mSelectionBtn;
    private ImageView mPenBtn;

    private Rect mScreenRect;

    private int mToolType = SpenSurfaceView.TOOL_SPEN;

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

        // Initialize Pen.

        boolean isSpenFeatureEnabled = false;
        Spen spenPackage = new Spen();
        try {
            spenPackage.initialize(this);

            isSpenFeatureEnabled =
                  spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
        } catch (SsdkUnsupportedException e) {
            if( SDKUtils.processUnsupportedException(this, e) == true) {
                return;
            }
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Pen.",
                           Toast.LENGTH_SHORT).show();
            e1.printStackTrace();
            finish();
        }

        mSpenViewLayout = (RelativeLayout) findViewById(R.id.spenViewLayout);

        // Create Pen View.
        mSpenSurfaceView = new SpenSurfaceView(mContext);
        if (mSpenSurfaceView == null) {
            Toast.makeText(mContext, "Cannot create new SpenSurfaceView.",
                           Toast.LENGTH_SHORT).show();
            finish();
        }
        mSpenViewLayout.addView(mSpenSurfaceView);

        // Get the dimensions of the screen.
        Display display = getWindowManager().getDefaultDisplay();
        mScreenRect = new Rect();
        display.getRectSize(mScreenRect);
        // Create SpenNoteDoc.
        try {
```

```
            mSpenNoteDoc = new SpenNoteDoc(mContext,
                mScreenRect.width(), mScreenRect.height());
        } catch (IOException e) {
            Toast.makeText(mContext, "Cannot create new NoteDoc.",
                           Toast.LENGTH_SHORT).show();
            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.
        mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);

        initPenSettingInfo();
        // Register the listener.
        mSpenSurfaceView.setControlListener(mControlListener);

        // Define the buttons.
        mSelectionBtn = (ImageView) findViewById(R.id.selectionBtn);
        mSelectionBtn.setOnClickListener(mSelectionBtnClickListener);

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

        selectButton(mPenBtn);

        setTextRecognition();

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

    private void initPenSettingInfo() {
        // Initialize pen settings.
        SpenSettingPenInfo penInfo = new SpenSettingPenInfo();
        penInfo.color = Color.BLUE;
        penInfo.size = 10;
        mSpenSurfaceView.setPenSettingInfo(penInfo);
    }



    private void setTextRecognition() {
        // Set TextRecognition
        mSpenTextRecognitionManager = new SpenTextRecognitionManager(mContext);
```

```
        List<SpenRecognitionInfo> textRecognitionList =
               mSpenTextRecognitionManager.getInfoList(
                 SpenObjectBase.TYPE_STROKE, SpenObjectBase.TYPE_CONTAINER);

        try {
            if (textRecognitionList.size() > 0) {
                for (SpenRecognitionInfo info : textRecognitionList) {
                    if (info.name.equalsIgnoreCase("SpenText")) {
                        mTextRecognition = mSpenTextRecognitionManager
                            .createRecognition(info);
                        break;
                    }
                }
            } else {
                finish();
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            Toast.makeText(mContext,
                  "SpenTextRecognitionManager class not found.",
                Toast.LENGTH_SHORT).show();
            return;
        } catch (InstantiationException e) {
            e.printStackTrace();
            Toast.makeText(mContext,
               "Failed to access the SpenTextRecognitionManager constructor.",
                Toast.LENGTH_SHORT).show();
            return;
        } catch (IllegalAccessException e) {
            e.printStackTrace();
            Toast.makeText(mContext,
               "Failed to access the SpenTextRecognitionManager field or method.",
                Toast.LENGTH_SHORT).show();
            return;
        } catch (SpenCreationFailureException e) {
            // End the application if text recognition is not available.
            e.printStackTrace();
            AlertDialog.Builder ad = new AlertDialog.Builder(this);
            ad.setIcon(this.getResources().getDrawable(
                android.R.drawable.ic_dialog_alert));
            ad.setTitle(this.getResources().getString(R.string.app_name))
                .setMessage(
                    "This device does not support Recognition.")
                .setPositiveButton("OK",
                    new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog,
                            int which) {
                            // finish dialog
                            dialog.dismiss();
                            finish();
                        }
                    }).show();
            ad = null;
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(mContext,
                  "SpenTextRecognitionManager engine not loaded.",
                Toast.LENGTH_SHORT).show();
```

```
            return;
        }

        // Select the language to be recognized: kor, eng or chn
        List<String> languageList = mTextRecognition.getSupportedLanguage();
        if (textRecognitionList.size() > 0) {
            for (String language : languageList) {
                if (language.equalsIgnoreCase("kor")) {
                    mTextRecognition.setLanguage(language);
                    break;
                }
            }
        }

        try {
            mTextRecognition.setResultListener(new ResultListener() {
                @Override
                public void onResult(List<SpenObjectBase> input,
                    List<SpenObjectBase> output) {
                    // Set rect that will draw text recognized by calculating the
                    // rect ranges of the selected objects, and purge the selected
                    // objects and append the recognized object to pageDoc.
                    RectF rect = new RectF(mScreenRect.width(),
                        mScreenRect.height(), 0, 0);

                    for (SpenObjectBase obj : input) {
                        if (rect.contains(obj.getRect()) == false) {
                            RectF objRect = obj.getRect();
                            rect.left = rect.left < objRect.left
                                ? rect.left : objRect.left;
                            rect.top = rect.top < objRect.top
                                ? rect.top : objRect.top;
                            rect.right = rect.right > objRect.right
                                ? rect.right : objRect.right;
                            rect.bottom = rect.bottom > objRect.bottom
                                ? rect.bottom : objRect.bottom;
                        }
                        mSpenPageDoc.removeObject(obj);
                    }

                    for (SpenObjectBase obj : output) {
                        if (obj instanceof SpenObjectTextBox) {
                            obj.setRect(rect, false);
                            mSpenPageDoc.appendObject(obj);
                        }
                    }
                    mIsProcessingRecognition = false;
                    mSpenSurfaceView.closeControl();
                    mSpenSurfaceView.update();
                }
            });
        } catch (IllegalStateException e) {
            e.printStackTrace();
            Toast.makeText(mContext, "SpenTextRecognition is not loaded.",
                Toast.LENGTH_SHORT).show();
            return;
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(mContext, "SpenTextRecognition is not loaded.",
```

```
                Toast.LENGTH_SHORT).show();
            return;
        }
    }

    private final OnClickListener mSelectionBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                selectButton(mSelectionBtn);
                mSpenSurfaceView.setToolTypeAction(mToolType,
                                                   SpenSurfaceView.ACTION_SELECTION);
            }
    };

    private final OnClickListener mPenBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                selectButton(mPenBtn);
                mSpenSurfaceView.setToolTypeAction(mToolType,
                                                   SpenSurfaceView.ACTION_STROKE);
            }
    };

    private SpenControlListener mControlListener = new SpenControlListener() {

        @Override
        public boolean onCreated(ArrayList<SpenObjectBase> selectedList,
                ArrayList<Rect> arg1,
                ArrayList<SpenContextMenuItemInfo> arg2,
                ArrayList<Integer> arg3, int arg4, PointF arg5) {
            if (selectedList.size() > 0 && !mIsProcessingRecognition) {
                // Incorporate the selected stroke object into the list and
                 // send it as a request.
                ArrayList<SpenObjectBase> inputList = new
                                                    ArrayList<SpenObjectBase>();
                for (int i = 0; i < selectedList.size(); i++) {
                    if (selectedList.get(i).getType() == SpenObjectBase.TYPE_STROKE)
{
                        inputList.add(selectedList.get(i));
                    }
                }

                if (inputList.size() <= 0) {
                    return false;
                }
                mIsProcessingRecognition = true;
                try {
                    mTextRecognition.request(inputList);
                } catch (IllegalStateException e) {
                    e.printStackTrace();
                    Toast.makeText(mContext,
                          "SpenTextRecognition is not loaded.",
                        Toast.LENGTH_SHORT).show();
                    return false;
                } catch (Exception e) {
                    e.printStackTrace();
                    Toast.makeText(mContext,
```

```
                        "SpenTextRecognition engine not loaded.",
                    Toast.LENGTH_SHORT).show();
                return false;
            }
            return true;
        }
        return false;
    }

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

    @Override
    public boolean onMenuSelected(ArrayList<SpenObjectBase> arg0,
            int arg1) {
        return false;
    }

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

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

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

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

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

    if (mTextRecognition != null) {
        mSpenTextRecognitionManager.destroyRecognition(mTextRecognition);
        mSpenTextRecognitionManager.close();
    }

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

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

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

For more information, see PenSample5_6_TextRecognition.java in PenSample5_6_TextRecognition.

4.5.6.1 Loading Text Recognition Plug-ins

To load text recognition plug-ins:

1. Create an SpenTextRecognitionManager instance.
2. Call `SpenTextRecognitionManager.getInfoList()` to get the list of text recognition information objects for the available recognition plug-ins with the specified input and output types. The sample uses Stroke and Container.

Select an appropriate text recognition plug-in from the list and call `SpenTextRecognitionManager.createRecognition()` with the associated text recognition information object to create a text recognition instance.

Call `SpenTextRecognition.getSupportedLanguage()` to get the list of supported languages.

Call `SpenTextRecognition.setLanguage()` and pass a supported language to the text recognition plug-in for text recognition.

```
mSpenTextRecognitionManager = new SpenTextRecognitionManager(mContext);

List<SpenRecognitionInfo> textRecognitionList =
               mSpenTextRecognitionManager.getInfoList(
                 SpenObjectBase.TYPE_STROKE, SpenObjectBase.TYPE_CONTAINER);

try {
            if (textRecognitionList.size() > 0) {
                for (SpenRecognitionInfo info : textRecognitionList) {
                    if (info.name.equalsIgnoreCase("SpenText")) {
                        mTextRecognition = mSpenTextRecognitionManager
                            .createRecognition(info);
                        break;
                    }
                }
            } else {
                finish();
            }
} catch (ClassNotFoundException e) {
} catch (InstantiationException e) {
} catch (IllegalAccessException e) {
} catch (SpenCreationFailureException e) {;
}

// Select the language to be recognized: kor, eng or chn
List<String> languageList = mTextRecognition.getSupportedLanguage();
```

```
if (textRecognitionList.size() > 0) {
            for (String language : languageList) {
                if (language.equalsIgnoreCase("kor")) {
                    mTextRecognition.setLanguage(language);
                    break;
                }
            }
}
```

4.5.6.2 Handling Insert Stroke Button Events

To handle Insert Stroke button events:

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

In the `onClick()` method of the Insert Stroke button, indicate that the button is selected and set `mToolType` to ACTION_STROKE .

```
private final OnClickListener mPenBtnClickListener =
        new OnClickListener() {
            public void onClick(View v) {
                selectButton(mPenBtn);
                mSpenSurfaceView.setToolTypeAction(mToolType,
                    SpenSurfaceView.ACTION_STROKE);
            }
};
```

4.5.6.3 Handling Selection Button Events

To handle Selection button events:

1. Create a Selection button.
2. Create an OnClickListener instance, mTextObjBtnClickListener in this sample, for the Selection button, mSelectionBtnClickListener in the sample, and register it by calling `setOnClickListener()` on the button.

In the `onClick()`method of the Selection button, set `mToolType` to ACTION_SELECTION and indicate that the button is selected.

```
private final OnClickListener mSelectionBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                mSpenSurfaceView.setToolTypeAction(mToolType,
                    SpenSurfaceView.ACTION_SELECTION);
                selectButton(mSelectionBtn);
            }
};
```

4.5.6.4 Selecting Objects for Text Recognition

To select an object for text recognition:

1. Create a control event listener for stroke object selection in SpenSurfaceView and register it by calling `SpenSurfaceView.setControlListener()`.

In the `onCreated()` method that is called when a control appears on the View, call `SpenTextRecognition.request()` to request text recognition and pass the list of selected stroke objects.

Set `isProcessingRecognition` to true to avoid duplicate requests while the text is being recognized.

```
private SpenControlListener mControlListener = new SpenControlListener() {

        @Override
        public boolean onCreated(ArrayList<SpenObjectBase> selectedList,
                ArrayList<Rect> arg1,
                ArrayList<SpenContextMenuItemInfo> arg2,
                ArrayList<Integer> arg3, int arg4, PointF arg5) {
            if (selectedList.size() > 0 && !mIsProcessingRecognition) {
                // Incorporate the selected stroke object into the list and
                // send it as a request.
                ArrayList<SpenObjectBase> inputList = new ArrayList<SpenObjectBase>();
                for (int i = 0; i < selectedList.size(); i++) {
                    if (selectedList.get(i).getType() == SpenObjectBase.TYPE_STROKE) {
                        inputList.add(selectedList.get(i));
                    }
                }

                if (inputList.size() <= 0) {
                    return false;
                }
                mIsProcessingRecognition = true;
                try {
                    mTextRecognition.request(inputList);
                } catch (IllegalStateException e) {
                } catch (Exception e) {
                }
                return true;
            }
            return false;
}
```

4.5.6.5 Handling Text Recognition Events

To handle text recognition events:

1. Create a ResultListener instance for text recognition events and register it by calling `SpenTextRecognition.setResultListener ()`.

In the `onResult()` method, do the following:

- Calculate the Rect of the selected stroke objects
- Set Rect to have the recognized text drawn at the calculated position

- Call `SpenPageDoc.removeObject()` to delete the selected stroke objects.
- Call `SpenObjectBase.setRect()` and pass the Boolean value false as the second parameter. The Rect of objects, which is sent as the recognition output, is already set in the recognition engine.
- Call `SpenPageDoc.appendObject()` to add the SpenObjectTextBox objects that are recognized as text to the current page.
- Call `SpenSurfaceView.update()` to refresh the viewport.
- Set `isProcessingRecognition` to false to prevent duplicate recognition requests.
- Call `SpenSurfaceView.closeControl()` to close the control.

```
try {
            mTextRecognition.setResultListener(new ResultListener() {
                @Override
                public void onResult(List<SpenObjectBase> input,
                    List<SpenObjectBase> output) {
                    // Set rect that will draw text recognized by calculating the
                    // rect ranges of the selected objects, and purge the selected
                    // objects and append the recognized object to pageDoc.
                    RectF rect = new RectF(mScreenRect.width(),
                        mScreenRect.height(), 0, 0);

                    for (SpenObjectBase obj : input) {
                        if (rect.contains(obj.getRect()) == false) {
                            RectF objRect = obj.getRect();
                            rect.left = rect.left < objRect.left
                                ? rect.left : objRect.left;
                            rect.top = rect.top < objRect.top
                                ? rect.top : objRect.top;
                            rect.right = rect.right > objRect.right
                                ? rect.right : objRect.right;
                            rect.bottom = rect.bottom > objRect.bottom
                                ? rect.bottom : objRect.bottom;
                        }
                        mSpenPageDoc.removeObject(obj);
                    }

                    for (SpenObjectBase obj : output) {
                        if (obj instanceof SpenObjectTextBox) {
                            obj.setRect(rect, false);
                            mSpenPageDoc.appendObject(obj);
                        }
                    }
                    mIsProcessingRecognition = false;
                    mSpenSurfaceView.closeControl();
                    mSpenSurfaceView.update();
                }
            });
} catch (IllegalStateException e) {
} catch (Exception e) {
}
```

4.5.6.6 Preventing Memory Leaks

To prevent memory leaks:

1. Call `SpenTextRecognitionManager.destroyRecognition()` and `SpenTextRecognitionManager.close()` to unload the text recognition plug-in
2. Call `SpenSurfaceView.closeControl()` to close the control.

Call `SpenSurfaceView.close()` and `SpenNoteDoc.close()` to close the SpenSurfaceView and SpenNoteDoc instances.

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

        if (mTextRecognition != null) {
            mSpenTextRecognitionManager.destroyRecognition(mTextRecognition);
            mSpenTextRecognitionManager.close();
        }

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

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

4.5.7. Verifying Signatures

You can use Pen for signature verification with pre-loaded signature recognition plug-ins.

The sample application implements the following features:

- ‘Check Signature’ to verify whether a signature is registered or not.
- ‘Registration’ to register signatures.
- ‘Verification’ to verify a registered signature.
- ‘Delete Signature’ to delete a registered signature.

For the Registration menu, the sample application implements the following features:

- It creates an SpenSurfaceView instance to allow users to create signatures.
- The application calls `SpenSignatureVerificationManager.createSignatureVerification()` to load a signature verification plug-in.

- When the user creates a signature on SPenSurfaceView and clicks Registration, the sample application calls `SpenSignatureVerificationManager.register()` three times to register the signature.
- The sample application then calls `finish()` to go the top-level menu.

For the Verification menu, the sample application implements the following features:

- It creates an SpenSurfaceView instance to allow users to create signatures.
- The application calls `SpenSignatureVerificationManager.createSignatureVerification()` to load a signature verification plug-in.
- It calls `SpenSignatureVerification.setResultListener()` to register the callback method that receives stroke recognition events, which are returned as a response to `SpenSignatureVerification.request()`.

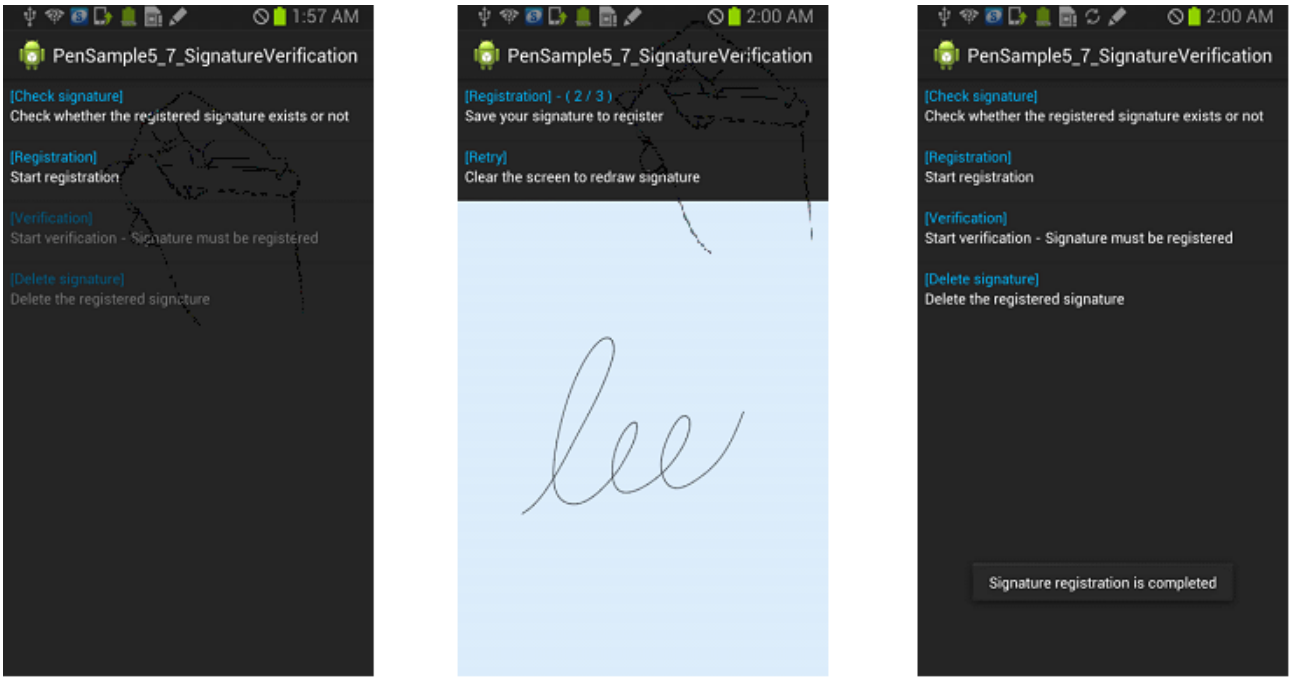


Figure 34: Signature registration

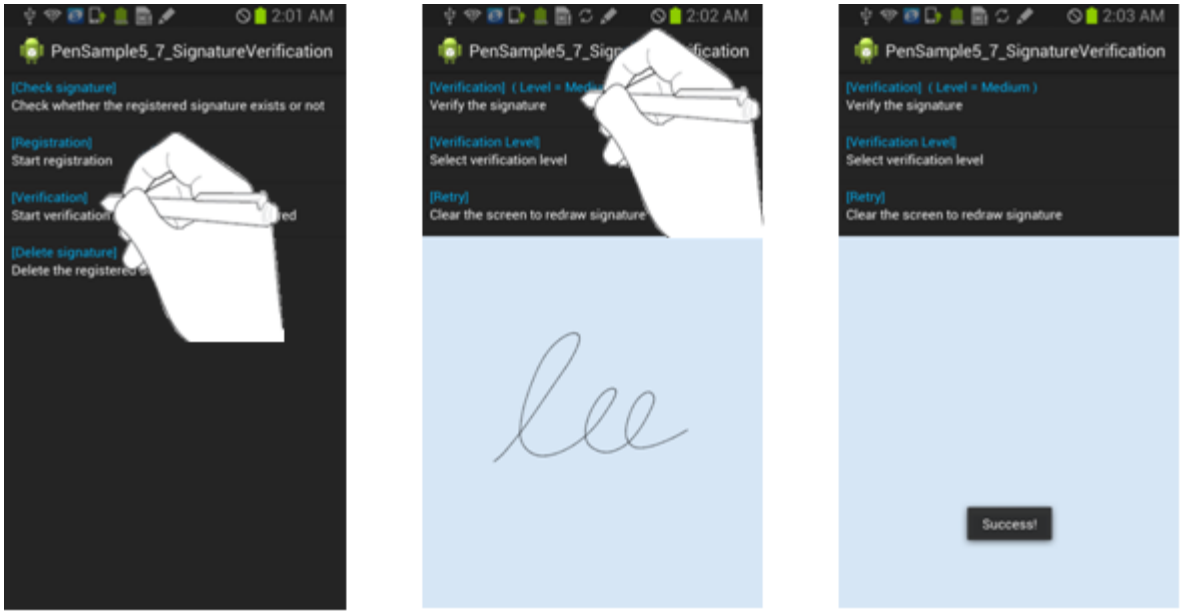


Figure 35: Signature verification

```
public class PenSample5_7_Signature extends Activity {

    private Context mContext = null;

    public ListView mSignatureList;
    public ArrayList<ListItem> mSignatureListItem;

    private SpenSignatureVerificationManager mSpenSignatureVerificationManager;
    private SpenSignatureVerification mSpenSignatureVerification;
    public ListAdapter mSignatureAdapter;

    private final int LIST_CHECK_SIGNATURE = 0;
    private final int LIST_REGISRTATION = 1;
    private final int LIST_VERIFICATION = 2;
    private final int LIST_DELETE_SIGNATURE = 3;

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

        // Initialize Pen.
        Spen spenPackage = new Spen();
        try {
            spenPackage.initialize(this);
        } catch (SsdkUnsupportedException e) {
            if( SDKUtils.processUnsupportedException(this, e) == true) {
                return;
            }
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Pen.",
                           Toast.LENGTH_SHORT).show();
            e1.printStackTrace();
```

```
            finish();
        }

        // Set list
        mSignatureListItem = new ArrayList<ListItem>();
        mSignatureListItem.add(new ListItem("[Check signature]",
            "Check whether the registered signature exists or not"));
        mSignatureListItem.add(new ListItem("[Registration]",
            "Start registration"));
        mSignatureListItem.add(new ListItem("[Verification]",
            "Start verification - Signature must be registered"));
        mSignatureListItem.add(new ListItem("[Delete signature]",
            "Delete the registered signature"));

        mSignatureAdapter = new ListAdapter(this);

        mSignatureList = (ListView) findViewById(R.id.signature_list);
        mSignatureList.setAdapter(mSignatureAdapter);

        // Set Verification
        mSpenSignatureVerificationManager =
            new SpenSignatureVerificationManager(mContext);

        List<SpenSignatureVerificationInfo> signatureVerificationList =
            mSpenSignatureVerificationManager.getInfoList();
        try {
            if (signatureVerificationList.size() > 0) {
                for (SpenSignatureVerificationInfo info : signatureVerificationList)
{
                    if (info.name.equalsIgnoreCase("SpenSignature")) {
                        mSpenSignatureVerification =
mSpenSignatureVerificationManager
                                .createSignatureVerification(info);
                        break;
                    }
                }
            } else {
                finish();
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            Toast.makeText(mContext,
                           "SpenSignatureVerificationManager class not found.",
                           Toast.LENGTH_SHORT).show();
            return;
        } catch (InstantiationException e) {
            e.printStackTrace();
            Toast.makeText(mContext,
                "Failed to access the SpenSignatureVerificationManager constructor.",
                Toast.LENGTH_SHORT).show();
            return;
        } catch (IllegalAccessException e) {
            e.printStackTrace();
            Toast.makeText(mContext,
             "Failed to access the SpenSignatureVerificationManager field or method.",
                Toast.LENGTH_SHORT).show();
            return;
        } catch (SpenCreationFailureException e) {
            // End the application unless the application supports
```

```
            // verification.
            e.printStackTrace();
            AlertDialog.Builder ad = new AlertDialog.Builder(this);
            ad.setIcon(this.getResources().getDrawable(
                android.R.drawable.ic_dialog_alert));
            ad.setTitle(this.getResources().getString(R.string.app_name))
                .setMessage(
                    "This device does not support Signature Recognition.")
                .setPositiveButton("OK",
                    new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog,
                            int which) {
                            // finish dialog
                            dialog.dismiss();
                            finish();
                        }
                    }).show();
            ad = null;
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(mContext,
               "SpenSignatureVerificationManager engine not loaded.",
                Toast.LENGTH_SHORT).show();
            return;
        }

        mSignatureList.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                int position, long id) {
                int registeredCount =
mSpenSignatureVerification.getRegisteredCount();
                int minimumRequiredCount =
                  mSpenSignatureVerification.getMinimumRequiredCount();
                if (position == LIST_CHECK_SIGNATURE) {
                    // Check whether any signature is registered or not.
                    if (registeredCount == minimumRequiredCount)
                        Toast.makeText(mContext, "Registered signatures exist.",
                            Toast.LENGTH_SHORT).show();
                    else
                        Toast.makeText(mContext,
                         "Registered Signature is less than minimum required count.",
                            Toast.LENGTH_SHORT).show();
                } else if (position == LIST_REGISRTATION) {
                    // Go to the Registration menu
                    Intent intent = new Intent(PenSample5_7_Signature.this,
                                        PenSample5_7_SignatureRegistration.class);
                    startActivity(intent); // create RegistrationActivity
                } else if (position == LIST_VERIFICATION) {
                    // Go to the Verification menu if any signature is found.
                    if (registeredCount == minimumRequiredCount) {
                        Intent intent = new Intent(PenSample5_7_Signature.this,
                                            PenSample5_7_SignatureVerification.class);
                        startActivity(intent);
                    } else
                        Toast.makeText(mContext,
                              "Registered Signature is less than minimum required count.",
                               Toast.LENGTH_SHORT).show();
```

```
                } else if (position == LIST_DELETE_SIGNATURE) {
                    // Delete the registered signature.
                    if (registeredCount == 0) {
                        Toast.makeText(mContext, "Signature is not registered.",
                                       Toast.LENGTH_SHORT).show();
                    } else {
                        mSpenSignatureVerification.unregisterAll();
                        if (mSpenSignatureVerification.getRegisteredCount() == 0)
                            Toast.makeText(mContext,
                                           "Registered signature is deleted.",
                                           Toast.LENGTH_SHORT).show();
                    }
                }
                mSignatureAdapter.notifyDataSetChanged();
            }
        });
    }

    class ListItem {
        ListItem(String iTitle, String isubTitle) {
            Title = iTitle;
            subTitle = isubTitle;
        }

        String Title;
        String subTitle;
    }

    class ListAdapter extends BaseAdapter {
        LayoutInflater Inflater;

        public ListAdapter(Context context) {
            Inflater =
                (LayoutInflater) context
                    .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        }

        @Override
        public int getCount() {
            return mSignatureListItem.size();
        }

        @Override
        public Object getItem(int position) {
            return null;
        }

        @Override
        public long getItemId(int position) {
            return 0;
        }

        @Override
        public View getView(int position, View convertView,
            ViewGroup parent) {
            if (convertView == null) {
                convertView =
                    Inflater.inflate(R.layout.signature_list_item, parent,
                        false);
```

```
            }
            TextView title = (TextView) convertView
                    .findViewById(R.id.signature_list_title);
            title.setText(mSignatureListItem.get(position).Title);
            TextView subtitle = (TextView) convertView
                    .findViewById(R.id.signature_list_subtitle);
            subtitle.setText(mSignatureListItem.get(position).subTitle);
            if ((position == LIST_VERIFICATION || position == LIST_DELETE_SIGNATURE)
                && mSpenSignatureVerification.getRegisteredCount()
                  != mSpenSignatureVerification
                    .getMinimumRequiredCount()) {
                title.setTextColor(0xFF005D87);
                subtitle.setTextColor(0xFF777777);
            } else {
                title.setTextColor(0xFF00B8FF);
                subtitle.setTextColor(0xFFFFFFFF);
            }
            return convertView;
        }
    }

    @Override
    protected void onResume() {
        if(mSignatureAdapter != null) {
            mSignatureAdapter.notifyDataSetChanged();
        }
        mSignatureAdapter.notifyDataSetChanged();
        super.onResume();
    }

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

        if (mSpenSignatureVerification != null) {
                mSpenSignatureVerificationManager
                            .destroySignatureVerification(mSpenSignatureVerification);
                mSpenSignatureVerificationManager.close();
        }

    }
}
```

For more information, see PenSample5_7_Signature.java in PenSample5_7_Signature.

```
public class PenSample5_7_SignatureRegistration extends Activity {

    private Context mContext = null;
    private SpenNoteDoc mSpenNoteDoc;
    private SpenPageDoc mSpenPageDoc;
    private SpenSurfaceView mSpenSurfaceView;

    public ListAdapter mSignatureAdapter;
    public ArrayList<ListItem> mSignatureListItem;

    public int mSignatureRegistrationNum = 0;
    public int mSignatureRegistrationNumMax;
    public ListView mSignatureList;
```

```
    private int mResult = 0;

    private SpenSignatureVerificationManager mSpenSignatureVerificationManager;
    private SpenSignatureVerification mSpenSignatureVerification;

    private final int LIST_REGISRTATION = 0;
    private final int LIST_RETRY = 1;

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

        // Initialize Pen.

        boolean isSpenFeatureEnabled = false;
        Spen spenPackage = new Spen();
        try {
            spenPackage.initialize(this);

            isSpenFeatureEnabled =
                        spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
        } catch (SsdkUnsupportedException e) {
            if( SDKUtils.processUnsupportedException(this, e) == true) {
                return;
            }
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Pen.",
                Toast.LENGTH_SHORT).show();
            e1.printStackTrace();
            finish();
        }

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

        // Get the dimensions of the 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();
        }
        // 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.
        mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);

        // Set Callback Listener(Interface)
        Toast.makeText(mContext, "Draw your signature to register.",
            Toast.LENGTH_SHORT).show();

        // Set the list
        mSignatureListItem = new ArrayList<ListItem>();
        mSignatureListItem.add(new ListItem("[Registration]",
            "Save your signature to register"));
        mSignatureListItem.add(new ListItem("[Retry]",
            "Clear the screen to redraw signature"));

        mSignatureAdapter = new ListAdapter(this);

        mSignatureList = (ListView) findViewById(R.id.signature_list);
        mSignatureList.setAdapter(mSignatureAdapter);

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

        // Set Verification
        mSpenSignatureVerificationManager =
            new SpenSignatureVerificationManager(mContext);

        List<SpenSignatureVerificationInfo> signatureVerificationList =
            mSpenSignatureVerificationManager.getInfoList();
        try {
            if (signatureVerificationList.size() > 0) {
                for (SpenSignatureVerificationInfo info : signatureVerificationList)
{
                    if (info.name.equalsIgnoreCase("SpenSignature")) {
                        mSpenSignatureVerification =
                                mSpenSignatureVerificationManager
                                .createSignatureVerification(info);
                        break;
                    }
                }
            } else {
                finish();
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
```

```
            Toast.makeText(mContext,
               "SpenSignatureVerificationManager class not found.",
                Toast.LENGTH_SHORT).show();
            return;
        } catch (InstantiationException e) {
            e.printStackTrace();
            Toast.makeText(mContext,
               "Failed to access the SpenSignatureVerificationManager constructor.",
                Toast.LENGTH_SHORT).show();
            return;
        } catch (IllegalAccessException e) {
            e.printStackTrace();
            Toast.makeText(mContext,
             "Failed to access the SpenSignatureVerificationManager field or method.",
                Toast.LENGTH_SHORT).show();
            return;
        } catch (SpenCreationFailureException e) {
            e.printStackTrace();
            Toast.makeText(mContext, "This device does not support Signature
Recognition.",
                Toast.LENGTH_SHORT).show();
            finish();
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(mContext,
               "mSpenSignatureVerificationManager engine not loaded.",
                Toast.LENGTH_SHORT).show();
            return;
        }

        mSignatureRegistrationNumMax =
            mSpenSignatureVerification.getMinimumRequiredCount();

        mSignatureList.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                int position, long id) {
                if (position == LIST_REGISRTATION) {
                        ArrayList<SpenObjectBase> strokeList =
                              mSpenPageDoc.getObjectList(SpenObjectBase.TYPE_STROKE);
                        if (strokeList.size() > 0) {
                        // Add the object on the view to the list.
                        ArrayList<SpenObjectStroke> list =
                                                new ArrayList<SpenObjectStroke>();
                        for (int i = 0; i < strokeList.size(); i++) {
                            list.add((SpenObjectStroke) strokeList.get(i));
                        }


                        // Register the object list as a signature.
                        try {
                            mSpenSignatureVerification.register(list);
                        } catch (IllegalStateException e) {
                            e.printStackTrace();
                            Toast.makeText(mContext,
                             "SpenSignatureVerification is not loaded.",
                                Toast.LENGTH_SHORT).show();
                            return;
                        } catch (IllegalArgumentException e) {
```

```
                            e.printStackTrace();
                            Toast.makeText(mContext, "SpenObjectStroke list is null.",
                                Toast.LENGTH_SHORT).show();
                            return;
                        } catch (Exception e) {
                            e.printStackTrace();
                            Toast.makeText(mContext,
                                "This signature is invalid for registration\n" +
                                "Please try again!", Toast.LENGTH_SHORT).show();
                            return;
                        }
                        int registeredCount =
                          mSpenSignatureVerification.getRegisteredCount();
                        mResult = registeredCount;
                        if (mResult == mSignatureRegistrationNumMax) {
                            // Move to the higher menu if enough signatures are
                            // registered to match the MinimunRequiredCount
                            Toast.makeText(mContext,
                                "Signature registration is completed",
                                Toast.LENGTH_SHORT).show();
                            finish();
                        } else if (mResult > 0) {
                            mSignatureRegistrationNum = mResult;
                            Toast.makeText(mContext, "Signature has been stored.",
                                Toast.LENGTH_SHORT).show();
                        } else { // Signature registration error
                            Toast.makeText(mContext,
                                "This signature is invalid for registration\n" +
                                "Please try again!", Toast.LENGTH_SHORT).show();
                        }
                        // After registering the signatures, delete the
                        // objects for the next registration.
                        mSpenPageDoc.removeAllObject();
                        mSpenSurfaceView.update();
                    }
                    mSignatureAdapter.notifyDataSetChanged();
                } else if (position == LIST_RETRY) {
                    // Delete the object for new input.
                    mSpenPageDoc.removeAllObject();
                    mSpenSurfaceView.update();
                    Toast.makeText(mContext, "Draw your signature to register.",
                        Toast.LENGTH_SHORT).show();
                }
            }
        });

    }

    class ListItem {
        ListItem(String iTitle, String isubTitle) {
            Title = iTitle;
            subTitle = isubTitle;
        }

        String Title;
        String subTitle;
    }

    class ListAdapter extends BaseAdapter {
```

```
        LayoutInflater Inflater;

        public ListAdapter(Context context) {
            Inflater = (LayoutInflater) context
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        }

        @Override
        public int getCount() {
            return mSignatureListItem.size();
        }

        @Override
        public Object getItem(int position) {
            return null;
        }

        @Override
        public long getItemId(int position) {
            return 0;
        }

        @Override
        public View getView(int position, View convertView,
            ViewGroup parent) {
            if (convertView == null) {
                convertView = Inflater.inflate(
                    R.layout.signature_list_item, parent, false);
            }

            if (position == LIST_REGISRTATION) {
                TextView title = (TextView) convertView
                    .findViewById(R.id.signature_list_title);
                title.setText(mSignatureListItem.get(position).Title
                    + " - ( " + mSignatureRegistrationNum + " / "
                    + mSignatureRegistrationNumMax + " )");

                TextView subtitle = (TextView) convertView
                    .findViewById(R.id.signature_list_subtitle);
                subtitle.setText(mSignatureListItem.get(position).subTitle);
            } else {
                TextView title = (TextView) convertView
                    .findViewById(R.id.signature_list_title);
                title.setText(mSignatureListItem.get(position).Title);

                TextView subtitle = (TextView) convertView
                    .findViewById(R.id.signature_list_subtitle);
                subtitle.setText(mSignatureListItem.get(position).subTitle);
            }
            return convertView;
        }
    }

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

    @Override
```

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

        if (mSpenSignatureVerification != null) {
            mSpenSignatureVerificationManager
                        .destroySignatureVerification(mSpenSignatureVerification);
            mSpenSignatureVerificationManager.close();
        }

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

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

For more information, see PenSample5_7_SignatureRegistration.java in PenSample5_7_SignatureRegistration.

```
public class PenSample5_7_SignatureVerification extends Activity {

    public ArrayList<ListItem> mSignatureListItem;

    private Context mContext = null;
    private SpenNoteDoc mSpenNoteDoc;
    private SpenPageDoc mSpenPageDoc;
    public SpenSurfaceView mSpenSurfaceView;

    public ListAdapter mSignatureAdapter;
    public ListView mSignatureList;

    int mVerificationLevel =
        SpenSignatureVerification.VERIFICATION_LEVEL_MEDIUM;

    private SpenSignatureVerificationManager mSpenSignatureVerificationManager;
    private SpenSignatureVerification mSpenSignatureVerification;

    private final int LIST_VERIFICATION = 0;
    private final int LIST_VERIFICATION_LEVEL = 1;
    private final int LIST_RETRY = 2;

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

        // Initialize Pen.
        boolean isSpenFeatureEnabled = false;
```

```
        Spen spenPackage = new Spen();
        try {
            spenPackage.initialize(this);

            isSpenFeatureEnabled =
                            spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
        } catch (SsdkUnsupportedException e) {
            if( SDKUtils.processUnsupportedException(this, e) == true) {
                return;
            }
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Pen.",
                Toast.LENGTH_SHORT).show();
            e1.printStackTrace();
            finish();
        }

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

        // Get the dimensions of the 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();
        }
        // 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.
        mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);

        Toast.makeText(mContext, "Draw your signature to verify.",
            Toast.LENGTH_SHORT).show();

        // Set the list
        mSignatureListItem = new ArrayList<ListItem>();
        mSignatureListItem.add(new ListItem("[Verification]",
```

```
            "Verify the signature"));
        mSignatureListItem.add(new ListItem("[Verification Level]",
            "Select verification level"));
        mSignatureListItem.add(new ListItem("[Retry]",
            "Clear the screen to redraw signature"));

        mSignatureAdapter = new ListAdapter(this);

        mSignatureList = (ListView) findViewById(R.id.signature_list);
        mSignatureList.setAdapter(mSignatureAdapter);

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

        // Set Verification
        mSpenSignatureVerificationManager =
            new SpenSignatureVerificationManager(mContext);

        List<SpenSignatureVerificationInfo> signatureVerificationList =
            mSpenSignatureVerificationManager.getInfoList();
        try {
            if (signatureVerificationList.size() > 0) {
                for (SpenSignatureVerificationInfo info : signatureVerificationList) {
                    if (info.name.equalsIgnoreCase("SpenSignature")) {
                        mSpenSignatureVerification = mSpenSignatureVerificationManager
                            .createSignatureVerification(info);
                        break;
                    }
                }
            } else {
                finish();
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            Toast.makeText(mContext,
               "SpenSignatureVerificationManager class not found.",
                Toast.LENGTH_SHORT).show();
            return;
        } catch (InstantiationException e) {
            e.printStackTrace();
            Toast.makeText(mContext,
               "Failed to access the SpenSignatureVerificationManager constructor.",
                Toast.LENGTH_SHORT).show();
            return;
        } catch (IllegalAccessException e) {
            e.printStackTrace();
            Toast.makeText(mContext,
             "Failed to access the SpenSignatureVerificationManager field or method.",
                Toast.LENGTH_SHORT).show();
            return;
        } catch (SpenCreationFailureException e) {
            e.printStackTrace();
            Toast.makeText(mContext, "This device does not support Recognition.",
```

```
                Toast.LENGTH_SHORT).show();
            finish();
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(mContext,
               "mSpenSignatureVerificationManager engine not loaded.",
                Toast.LENGTH_SHORT).show();
            return;
        }

        try {
            mSpenSignatureVerification
                .setResultListener(new ResultListener() {
                    @Override
                    public void onResult(List<SpenObjectStroke> input,
                        boolean result) {
                        // Test whether the signature has been successfully
                        // verified or not.
                        if (result) {
                            Toast.makeText(mContext, "Success!",
                                Toast.LENGTH_SHORT).show();
                        } else {
                            Toast.makeText(mContext, "Failure!",
                                Toast.LENGTH_SHORT).show();
                        }
                        mSpenPageDoc.removeAllObject();
                        mSpenSurfaceView.update();
                    }
                });
        } catch (IllegalStateException e) {
            e.printStackTrace();
            Toast.makeText(mContext, "SpenSignatureVerification is not loaded.",
                Toast.LENGTH_SHORT).show();
            return;
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(mContext, "SpenSignatureVerification is not loaded.",
                Toast.LENGTH_SHORT).show();
            return;
        }

        mSignatureList.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                int position, long id) {
                if (position == LIST_VERIFICATION) {
                        ArrayList<SpenObjectBase> strokeList =
                              SpenPageDoc.getObjectList(SpenObjectBase.TYPE_STROKE);
                        if (strokeList.size() > 0) {
                        // Add the object on the view to the list.
                        ArrayList<SpenObjectStroke> list =
                                                    new ArrayList<SpenObjectStroke>();
                        for (int i = 0; i < strokeList.size(); i++) {
                            list.add((SpenObjectStroke) strokeList.get(i));
                        }

                        // Send a request message for comparing registered
                        //signatures.
                        try {
```

```
                        mSpenSignatureVerification.request(list);
                    } catch (IllegalStateException e) {
                        e.printStackTrace();
                        Toast.makeText(mContext,
                         "SpenSignatureVerification is not loaded.",
                            Toast.LENGTH_SHORT).show();
                        return;
                    } catch (Exception e) {
                        e.printStackTrace();
                        Toast.makeText(mContext,
                         "SpenSignatureVerification is not loaded.",
                            Toast.LENGTH_SHORT).show();
                        return;
                    }
                }

            } else if (position == LIST_VERIFICATION_LEVEL) {

                // Set the signature verification level.
                AlertDialog.Builder ab = new AlertDialog.Builder(
                    PenSample5_7_SignatureVerification.this);

                mVerificationLevel =
                    mSpenSignatureVerification.getVerificationLevel();

                String[] strLevel = { "Low", "Medium", "High" };

                ab.setTitle("Select verification level")
                    .setSingleChoiceItems(strLevel, mVerificationLevel,
                        new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(
                                DialogInterface dialog, int which) {
                                mVerificationLevel = which;
                            }
                        })
                    .setPositiveButton("Confirm",
                        new DialogInterface.OnClickListener() {
                            @Override
                            public void
                                onClick(DialogInterface dialog,
                                    int whichButton) {
                                mSpenSignatureVerification
                                    .setVerificationLevel(mVerificationLevel);
                                mSignatureAdapter
                                    .notifyDataSetChanged();
                            }
                        }).setNegativeButton("Cancel", null).show();
            } else if (position == LIST_RETRY) {
                // Delete the object for new input.
                mSpenPageDoc.removeAllObject();
                mSpenSurfaceView.update();
                Toast.makeText(mContext, "Draw your signature to verify.",
                    Toast.LENGTH_SHORT).show();
            }
        }
    });

}
```

```
// Items for ListView
class ListItem {
    ListItem(String iTitle, String isubTitle) {
        Title = iTitle;
        subTitle = isubTitle;
    }

    String Title;
    String subTitle;
}

// Adapter class for list Item
class ListAdapter extends BaseAdapter {
    LayoutInflater Inflater;

    public ListAdapter(Context context) {
        Inflater = (LayoutInflater) context
            .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    }

    @Override
    public int getCount() {
        return mSignatureListItem.size();
    }

    @Override
    public Object getItem(int position) {
        return mSignatureListItem.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView,
        ViewGroup parent) {
        if (convertView == null) {
            convertView = Inflater.inflate(
                R.layout.signature_list_item, parent, false);
        }

        if (position == LIST_VERIFICATION) {
            TextView title = (TextView) convertView
                .findViewById(R.id.signature_list_title);

            if (mVerificationLevel
                == SpenSignatureVerification.VERIFICATION_LEVEL_LOW) {
                title.setText(mSignatureListItem.get(position).Title
                    + "  ( Level = Low )");
            } else if (mVerificationLevel
                == SpenSignatureVerification.VERIFICATION_LEVEL_MEDIUM) {
                title.setText(mSignatureListItem.get(position).Title
                    + "  ( Level = Medium )");
            } else if (mVerificationLevel
                == SpenSignatureVerification.VERIFICATION_LEVEL_HIGH) {
                title.setText(mSignatureListItem.get(position).Title
```

```
                        + "  ( Level = High )");
                }

                TextView subtitle = (TextView) convertView
                        .findViewById(R.id.signature_list_subtitle);
                subtitle
                    .setText(mSignatureListItem.get(position).subTitle);
            } else {
                TextView title = (TextView) convertView
                        .findViewById(R.id.signature_list_title);
                title.setText(mSignatureListItem.get(position).Title);
                TextView subtitle = (TextView) convertView
                        .findViewById(R.id.signature_list_subtitle);
                subtitle
                    .setText(mSignatureListItem.get(position).subTitle);
            }
            return convertView;
        }
    }

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

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

        if (mSpenSignatureVerification != null) {
            mSpenSignatureVerificationManager
                        .destroySignatureVerification(mSpenSignatureVerification);
            mSpenSignatureVerificationManager.close();
        }

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

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

For more information, see PenSample5_7_SignatureVerification.java in PenSample5_7_SignatureVerification.

4.5.7.1 Loading Signature Verification Plug-ins

To load a signature verification plug-in:

1. Create an SpenSignatureVerificationManager instance.

Call `SpenSignatureVerificationManager.getInfoList()` to get the list of signature verification information objects for the available signature verification plug-ins.

Select an appropriate signature verification plug-in from the list and call `SpenSignatureVerificationManager.createSignatureVerification()` with the appropriate signature verification information object to load that signature verification plug-in.

If Pen fails to load a plug-in, an exception is thrown with a reason for the error.

```
mSpenSignatureVerificationManager =
            new SpenSignatureVerificationManager(mContext);

List<SpenSignatureVerificationInfo> signatureVerificationList =
            mSpenSignatureVerificationManager.getInfoList();
try {
            if (signatureVerificationList.size() > 0) {
                for (SpenSignatureVerificationInfo info : signatureVerificationList) {
                    if (info.name.equalsIgnoreCase("SpenSignature")) {
                            mSpenSignatureVerification =
mSpenSignatureVerificationManager
                                .createSignatureVerification(info);
                        break;
                    }
                }
            } else {
                finish();
            }
} catch (ClassNotFoundException e) {
} catch (InstantiationException e) {
} catch (IllegalAccessException e) {
} catch (SpenCreationFailureException e) {
} catch (Exception e) {
}
```

4.5.7.2 Checking How Many Signatures Are Required for Verification

The pre-loaded signature verification plug-in needs at least three (3) samples of a signature to reliably recognizable it. However, the minimum number of signatures depends on the signature verification plug-in.

If ‘Check signature’ is selected, in the onItemClick() method:

1. Call `SpenSignatureVerification.getRegisteredCount()` to get the number of registered signatures.

Call `SpenSignatureVerification.getMinimumRequiredCount()` to get the minimum number of signatures required by the recognition plug-in.

If the two values are equal, the signatures are normally registered; otherwise, show a message that no registered signature exists.

```
public void onItemClick(AdapterView<?> parent, View view,
```

```
                int position, long id) {
        int registeredCount = mSpenSignatureVerification.getRegisteredCount();
        int minimumRequiredCount = mSpenSignatureVerification.getMinimumRequiredCount();
        if (position == LIST_CHECK_SIGNATURE) {
                    // Check whether the signature is registered.
                    if (registeredCount == minimumRequiredCount)
                        Toast.makeText(mContext, "Registered signatures exist.",
                            Toast.LENGTH_SHORT).show();
                    else
                        Toast.makeText(mContext,
                       "Registered Signature is less than minimum required count.",
                            Toast.LENGTH_SHORT).show();
```

If 'Registration' is selected, call `startActivity()` in the `onItemClick()` method to execute the PenSample5_7_SignatureRegistration activity.

```
} else if (position == LIST_REGISRTATION) {
            // Move to the Registration menu.
            Intent intent = new Intent(PenSample5_7_Signature.this,
                        PenSample5_7_SignatureRegistration.class);
            startActivity(intent); // Create RegistrationActivity
```

If 'Verification' is selected, check if the minimum specified number of signatures are registered in the `onItemClick()` method. Call `startActivity()` to execute the PenSample5_7_ SignatureVerification activity.

```
} else if (position == LIST_VERIFICATION) {
            // Move to the Verification menu if any signature is
            // registered.
            if (registeredCount == minimumRequiredCount) {
                    Intent intent = new Intent(PenSample5_7_Signature.this,
                            PenSample5_7_SignatureVerification.class);
                    startActivity(intent);
            } else
                    Toast.makeText(mContext,
                        "Registered Signature is less than minimum required count.",
                            Toast.LENGTH_SHORT).show();
```

If 'Delete Signature' is selected, check if any signatures are registered in the `onItemClick()` method. Call `SpenSignatureVerification.unregisterAll()` to delete the signature.

```
} else if (position == LIST_DELETE_SIGNATURE) {
                    // Delete the signature registered.
             if (registeredCount == 0) {
                 Toast.makeText(mContext, "Signature is not registered.",
                            Toast.LENGTH_SHORT).show();
             } else {
```

```
                mSpenSignatureVerification.unregisterAll();
                if (mSpenSignatureVerification.getRegisteredCount() == 0)
                    Toast.makeText(mContext, "Registered signature is deleted.",
                                Toast.LENGTH_SHORT).show();
            }
}
mSignatureAdapter.notifyDataSetChanged();
```

4.5.7.3 Registering a Signature

To register a signature:

1. If 'Registration' is selected and if a stroke object has been added to SpenSurfaceView, call `mSpenPageDoc.getObjectList()` in the `onItemClick()` method to get the list of objects.

Call `mSpenSignatureVerification.register()` and pass the list of objects to register the signature.

When Pen fails to register signatures, an exception is thrown with a reason.

```
public void onItemClick(AdapterView<?> parent, View view,
                int position, long id) {
    if (position == LIST_REGISRTATION) {
         ArrayList<SpenObjectBase> strokeList =
                        mSpenPageDoc.getObjectList(SpenObjectBase.TYPE_STROKE);
         if (strokeList.size() > 0) {
              // Add the object on the view to the list.
              ArrayList<SpenObjectStroke> list = new ArrayList<SpenObjectStroke>();
              for (int i = 0; i < strokeList.size(); i++) {
                   list.add((SpenObjectStroke) strokeList.get(i));
              }


              // Register the object list as a signature.
              try {
                   mSpenSignatureVerification.register(list);
              } catch (IllegalStateException e) {
              } catch (IllegalArgumentException e) {
              } catch (Exception e) {
              }
```

Call `SpenSignatureVerification.getRegisteredCount()` to get the number of registered signatures.

If it is equal to the minimum number of required signatures, call `finish()` to complete the signature registration activity and go to the top-level menu.

After registering a signature, call `SpenPageDoc.removeAllObject()` to delete the stroke object for the next registration.

Call `SpenSurfaceView.update()` to refresh the viewport.

```
int registeredCount = mSpenSignatureVerification.getRegisteredCount();
mResult = registeredCount;
if (mResult == mSignatureRegistrationNumMax) {
            // Move to the higher menu if enough signatures are
```

```
            // registered to match the MinimunRequiredCount
            Toast.makeText(mContext,
                                "Signature registration is completed",
                                Toast.LENGTH_SHORT).show();
            finish();
} else if (mResult > 0) {
            mSignatureRegistrationNum = mResult;
            Toast.makeText(mContext, "Signature has been stored.",
                                Toast.LENGTH_SHORT).show();
} else { // Signature registration error
            Toast.makeText(mContext,
                                "This signature is invalid for registration\n" +
                                "Please try again!", Toast.LENGTH_SHORT).show();
}
// After registering the signatures, delete the
// objects for the next registration.
mSpenPageDoc.removeAllObject();
mSpenSurfaceView.update();
}
```

If 'Retry' is selected, call `SpenPageDoc.removeAllObject()` to delete the stroke object to allow users to add new signature strokes.

Call `SpenSurfaceView.update()` to refresh the viewport.

```
} else if (position == LIST_RETRY) {
            // Delete the object for new input.
            mSpenPageDoc.removeAllObject();
            mSpenSurfaceView.update();
            Toast.makeText(mContext, "Draw your signature to register.",
                        Toast.LENGTH_SHORT).show();
}
}
```

4.5.7.4 Verifying a Signature

To verify a signature:

1. If 'Verification' is selected and if a stroke has been added to SpenSurfaceView, call `mSpenPageDoc.getObjectList()` in the `onItemClick()` method to get the list of objects.

Call `mSpenSignatureVerification.request()` and pass the list of objects to request signature verification.

Call `SpenSignatureVerification.setResultListener()` to register the ResultListener to receive the verification results. The `ResultListener.onResult()` callback method receives the verification results.

```
public void onItemClick(AdapterView<?> parent, View view,
                                int position, long id) {
          if (position == LIST_VERIFICATION) {
               ArrayList<SpenObjectBase> strokeList =
                         mSpenPageDoc.getObjectList(SpenObjectBase.TYPE_STROKE);
```

```
                if (strokeList.size() > 0) {
                     // Add the object on the view to the list.
                     ArrayList<SpenObjectStroke> list =
                                                  new ArrayList<SpenObjectStroke>();
                     for (int i = 0; i < strokeList.size(); i++) {
                          list.add((SpenObjectStroke) strokeList.get(i));
                     }


                     // Send a request message for comparing registered
                     //signatures.
                    try {
                             mSpenSignatureVerification.request(list);
                    } catch (IllegalStateException e) {
                    } catch (Exception e) {
                    }
}
```

If ‘Verification Level’ is selected, in the `onItemClick()` method:

1. Display a dialog to prompt the user to select a verification level: Low, Medium, or High.

Call `SpenSignatureVerification.setVerificationLevel()` and pass the selected verification level.

```
} else if (position == LIST_VERIFICATION_LEVEL) {
          // Set the signature verification level.
          AlertDialog.Builder ab = new AlertDialog.Builder(
                       PenSample5_7_SignatureVerification.this);

          mVerificationLevel =
                       mSpenSignatureVerification.getVerificationLevel();

          String[] strLevel = { "Low", "Medium", "High" };

          ab.setTitle("Select verification level")
                       .setSingleChoiceItems(strLevel, mVerificationLevel,
                           new DialogInterface.OnClickListener() {
                               @Override
                               public void onClick(
                                   DialogInterface dialog, int which) {
                                   mVerificationLevel = which;
                               }
                           })
                       .setPositiveButton("Confirm",
                           new DialogInterface.OnClickListener() {
                               @Override
                               public void
                                   onClick(DialogInterface dialog,
                                       int whichButton) {
                                   mSpenSignatureVerification
                                       .setVerificationLevel(mVerificationLevel);
                                   mSignatureAdapter
                                       .notifyDataSetChanged();
                               }
                           }).setNegativeButton("Cancel", null).show();
```

If 'Retry' is selected, in the `onItemClick()` method:

1. Call `SpenPageDoc.removeAllObject()` to delete the stroke object to allow users to create a new signature stroke.

Call `SpenSurfaceView.update()` to refresh the viewport.

```
} else if (position == LIST_RETRY) {
          // Delete the object for new input.
          mSpenPageDoc.removeAllObject();
          mSpenSurfaceView.update();
          Toast.makeText(mContext, "Draw your signature to verify.",
                        Toast.LENGTH_SHORT).show();
}
}
```

4.5.7.5 Handling Signature Recognition Events

To handle a signature recognition event:

1. Based on the value returned by the signature verification plug-in, display the signature verification results on the viewport. In the sample, “Success” or “Failure”.

Call `SpenPageDoc.removeAllObject()` to delete the stroke object.

Call `SpenSurfaceView.update()` to refresh the viewport.

```
public void onResult(List<SpenObjectStroke> input,
                        boolean result) {
          // Test whether the signature has been successfully
          // verified or not.
          if (result) {
                    Toast.makeText(mContext, "Success!",
                    Toast.LENGTH_SHORT).show();
          } else {
                    Toast.makeText(mContext, "Failure!",
                    Toast.LENGTH_SHORT).show();
          }
          mSpenPageDoc.removeAllObject();
          mSpenSurfaceView.update();
}
```

4.5.7.6 Preventing Memory Leaks

To prevent memory leaks:

1. Call `SpenSurfaceView.close()` and `SpenNoteDoc.close()` to close the SpenSurfaceView and SpenNoteDoc instances.

Call SpenSignatureVerificationManager.destroySignatureVerification() and SpenSignatureVerificationManager.close() to unload the signature verification plug-in.

```
protected void onDestroy() {
        super.onDestroy();
        if (mSpenSignatureVerification != null) {
             mSpenSignatureVerificationManager
                        .destroySignatureVerification(mSpenSignatureVerification);
             mSpenSignatureVerificationManager.close();
        }

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

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

4.5.8. Using Stroke Frame

Pen introduces Stroke Frame to allow you to draw a shape and use it as a frame for photos. You can use a free form shape or a recognized shape as the frame. To use this feature in your application, use the SpenSurfaceView.takeStrokeFrame(), SpenSurfaceView.retakeStrokeFrame() and SpenSurfaceView .cancelStrokeFrame() methods.

The sample application implements the following features:

- Button for Stroke Frame.
- Listeners for Stroke Frame button, for control events on viewport, and for receiving Stroke Frame results.
- The sample application uses Stroke Frame objects as frame for photos.

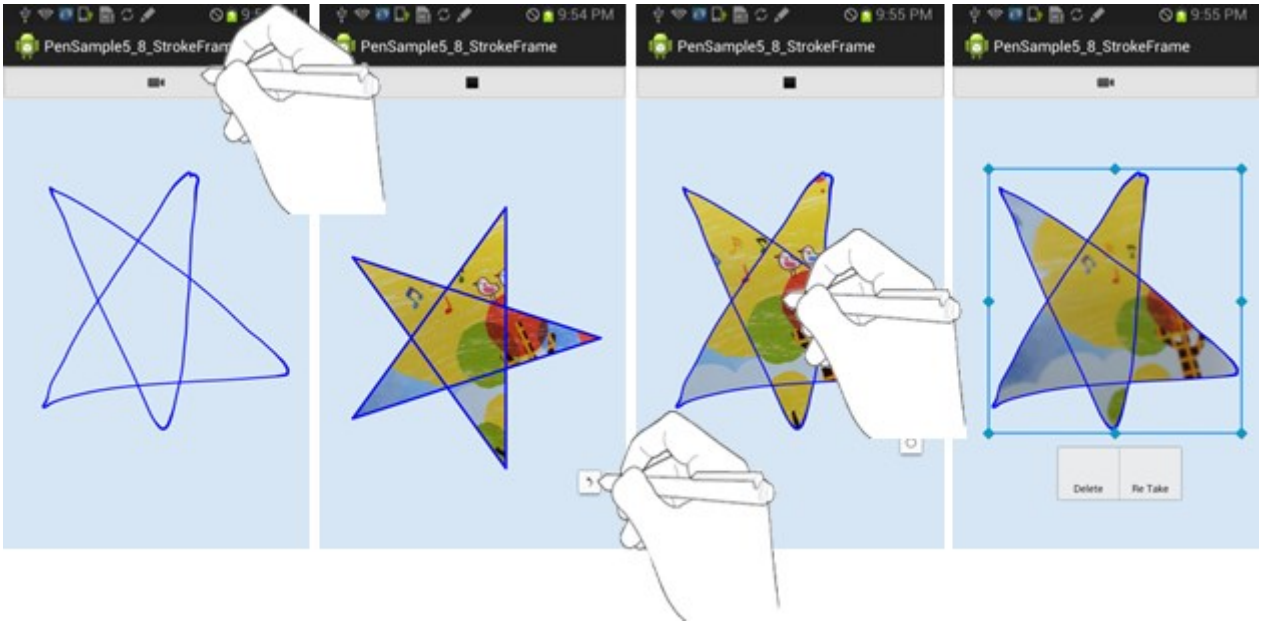


Figure 36: StrokeFrame

```
public class PenSample5_8_StrokeFrame extends Activity {

    private final int CONTEXT_MENU_DELETE = 0;
    private final int CONTEXT_MENU_RETAKE = 1;

    private Context mContext;
    private SpenNoteDoc mSpenNoteDoc;
    private SpenPageDoc mSpenPageDoc;
    private SpenSurfaceView mSpenSurfaceView;
    RelativeLayout mSpenViewLayout;

    private ImageView mStrokeFrameBtn;

    private SpenObjectContainer mStrokeFrameContainer;

    boolean mStrokeFrameStarted = false;

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

        // Initialize Pen.
        boolean isSpenFeatureEnabled = false;
        Spen spenPackage = new Spen();
        try {
            spenPackage.initialize(this);
            isSpenFeatureEnabled = spenPackage.isFeatureEnabled(Spen.DEVICE_PEN);
        } catch (SsdkUnsupportedException e) {
            if( SDKUtils.processUnsupportedException(this, e) == true) {
                return;
```

```
            }
        } catch (Exception e1) {
            Toast.makeText(mContext, "Cannot initialize Pen.",
                           Toast.LENGTH_SHORT).show();
            e1.printStackTrace();
            finish();
        }

        mSpenViewLayout =
            (RelativeLayout) findViewById(R.id.spenViewLayout);

        // Create Pen View.
        mSpenSurfaceView = new SpenSurfaceView(mContext);
        if (mSpenSurfaceView == null) {
            Toast.makeText(mContext, "Cannot create new SpenSurfaceView.",
                           Toast.LENGTH_SHORT).show();
            finish();
        }
        mSpenViewLayout.addView(mSpenSurfaceView);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);

        // Get the dimensions of the screen of the device.
        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();
        }
        // After adding a page to the NoteDoc, get an instance and set it as a member
variable.
        mSpenPageDoc = mSpenNoteDoc.appendPage();
        mSpenPageDoc.setBackgroundColor(0xFFD6E6F5);
        mSpenPageDoc.clearHistory();
        // Set a PageDoc to View.
        mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);

        initPenSettingInfo();
        // Register listener.
        mSpenSurfaceView.setControlListener(mControlListener);

        // Define buttons.
        mStrokeFrameBtn = (ImageView) findViewById(R.id.videoBtn);
        mStrokeFrameBtn.setOnClickListener(mFrameBtnClickListener);

        if(isSpenFeatureEnabled == false) {
            mSpenSurfaceView.setToolTypeAction(SpenSurfaceView.TOOL_FINGER,
                SpenSurfaceView.ACTION_STROKE);
            Toast.makeText(mContext,
                "Device does not support S pen. \n You can draw stroke with your
```

```
finger",
                Toast.LENGTH_SHORT).show();
        }
    }

    private void initPenSettingInfo() {
        // Reset the settings for pen.
        SpenSettingPenInfo penInfo = new SpenSettingPenInfo();
        penInfo.color = Color.BLUE;
        penInfo.size = 10;
        mSpenSurfaceView.setPenSettingInfo(penInfo);
    }

    private final OnClickListener mFrameBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                // Cancel Stroke Frame if picture is being taken.
                if (mStrokeFrameStarted) {
                    mStrokeFrameBtn.setImageResource(R.drawable.selector_video);
                    mStrokeFrameStarted = false;

                    mSpenSurfaceView.cancelStrokeFrame();
                } else {
                    // Create a frame with the objects and start taking a picture.
                    ArrayList<SpenObjectBase> oList =
                        mSpenPageDoc.getObjectList(SpenPageDoc.FIND_TYPE_STROKE);

                    if (oList.size() != 0) {
                        mStrokeFrameBtn.setImageResource(R.drawable.tool_ic_stop);
                        mStrokeFrameStarted = true;

                        ArrayList<SpenObjectStroke> osList =
                            new ArrayList<SpenObjectStroke>();
                        for (SpenObjectBase o : oList) {
                            osList.add((SpenObjectStroke) o);
                        }

                        mSpenSurfaceView.update();
                        mSpenSurfaceView.takeStrokeFrame((Activity) mContext,
                            mSpenViewLayout, osList, mStrokeFrameListener);
                    } else {
                        Toast.makeText(mContext,
                            "It doesn't work.\nPlease draw the stroke.",
                            Toast.LENGTH_SHORT).show();
                    }
                }

            }
    };

    SpenControlListener mControlListener = new SpenControlListener() {

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

        @Override
```

```
        public boolean onCreated(ArrayList<SpenObjectBase> objectList,
            ArrayList<Rect> relativeRectList,
            ArrayList<SpenContextMenuItemInfo> menu,
            ArrayList<Integer> styleList,  int pressType, PointF point ) {
            if (objectList == null) {
                return false;
            }
            SpenControlBase control = mSpenSurfaceView.getControl();
            if(control != null) {
                control.setContextMenuVisible(true);
            }
            // Set a context menu.
            menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_DELETE,
                   "Delete", true));
            // Add Retake context menu item if the selected object is a container.
            if(objectList.get(0).getType() == SpenObjectBase.TYPE_CONTAINER) {
                menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_RETAKE,
                       "Re Take", true));
                mStrokeFrameContainer = (SpenObjectContainer) objectList.get(0);
            }
            return true;
        }

        @Override
        public boolean onMenuSelected(
            ArrayList<SpenObjectBase> objectList, int itemId) {
            if (objectList == null) {
                return true;
            }
            // Delete the selected object (Stroke Frame).
            if (itemId == CONTEXT_MENU_DELETE) {
                mSpenPageDoc.removeSelectedObject();
                mSpenSurfaceView.closeControl();
                mSpenSurfaceView.update();
            // Retake Stroke Frame.
            } else if(itemId == CONTEXT_MENU_RETAKE) {
                SpenControlBase control = mSpenSurfaceView.getControl();
                if(control != null) {
                    control.setContextMenuVisible(false);
                }

                mSpenSurfaceView.retakeStrokeFrame((Activity)mContext, mSpenViewLayout,
                        mStrokeFrameContainer, mStrokeFrameListener);
                mStrokeFrameBtn.setImageResource(R.drawable.tool_ic_stop);
                mStrokeFrameStarted = true;
            }
            return false;
        }

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

        @Override
        public void onRectChanged(RectF rect, SpenObjectBase object) {
        }

        @Override
        public void onRotationChanged(float angle,
```

```
            SpenObjectBase objectBase) {
        }
    };

    private SpenStrokeFrameListener mStrokeFrameListener =
            new SpenStrokeFrameListener() {
        @Override
        public void onCompleted(int frameType, SpenObjectContainer o) {
            // When the picture has been taken, select the object and show the context
menu.
            mSpenPageDoc.selectObject(o);
            mSpenSurfaceView.update();
            mStrokeFrameContainer = o;

            SpenControlBase control = mSpenSurfaceView.getControl();
            if(control != null) {
                control.setContextMenuVisible(true);
            }
            mStrokeFrameBtn.setImageResource(R.drawable.selector_video);
            mStrokeFrameStarted = false;
        }

        @Override
        public void onCanceled(int state, SpenObjectContainer o) {

        }
    };

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

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

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

For more information, see PenSample5_8_StrokeFrame.java in PenSample5_8_StrokeFrame.

4.5.8.1 Registering a Listener for the Stroke Frame Button

To handle Stroke Frame button events:

1. Create a Stroke Frame Button.
2. Create an OnClickListener instance, mFrameBtnCLickListener in the sample, and register it by calling `setOnClickListener()` on the button.

In the `onClick()` method, if a picture is being taken, cancel the Stroke Frame by calling `SpenSurfaceView.cancelStrokeFrame()`. Otherwise, call `SpenSurfaceView.takeStrokeFrame()` to start taking a picture and pass the stroke drawn on the viewport and your SpenStrokeFrameListener instance as input parameters.

```
private final OnClickListener mFrameBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                // Cancel Stroke Frame if a picture is being taken.
                if (mStrokeFrameStarted) {
                    mStrokeFrameBtn.setImageResource(R.drawable.selector_video);
                    mStrokeFrameStarted = false;

                    mSpenSurfaceView.cancelStrokeFrame();
                } else {
                    // Create Stroke Frame with the objects and start taking a
picture.
                    ArrayList<SpenObjectBase> oList =
                        mSpenPageDoc.getObjectList(SpenPageDoc.FIND_TYPE_STROKE);

                    if (oList.size() != 0) {
                        mStrokeFrameBtn.setImageResource(R.drawable.tool_ic_stop);
                        mStrokeFrameStarted = true;

                        ArrayList<SpenObjectStroke> osList =
                            new ArrayList<SpenObjectStroke>();
                        for (SpenObjectBase o : oList) {
                            osList.add((SpenObjectStroke) o);
                        }

                        mSpenSurfaceView.update();
                        mSpenSurfaceView.takeStrokeFrame((Activity) mContext,
                            mSpenViewLayout, osList, mStrokeFrameListener);
                    } else {
                        Toast.makeText(mContext,
                            "It doesn't work.\nPlease draw the stroke.",
                            Toast.LENGTH_SHORT).show();
                    }
                }
            }
};
```

4.5.8.2 Registering a Control Event Listener

To handle the control events on the viewport:

1. Create an SpenControlListener instance for control events on the viewport and register it by calling `SpenSurfaceView.setControlListener()`.

In the `onCreated()` method:

- Set a 'Delete' context menu item for deleting a selected object.
- If the selected object (stroke) is a container, set a 'Re take' context menu item. When a stroke is converted to Stroke Frame, it becomes a container object (SpenObjectBase.TYPE_CONTAINER).

In the `onMenuSelected()` method, if the 'Delete' context menu item is selected:

- Call `SpenPageDoc.removeSelectedObject()` to delete the selected object.
- Call `SpenSurfaceView.closeControl()` to close the control.
- Call `SpenSurfaceView.update()` to update the viewport.

In the `onMenuSelected()` method, if the 'Re take' context menu is selected:

- Call `SpenSurfaceView.retakeStrokeFrame()` to restart taking a picture.

```
SpenControlListener mControlListener = new SpenControlListener() {

                .........

        @Override
        public boolean onCreated(ArrayList<SpenObjectBase> objectList,
            ArrayList<Rect> relativeRectList,
            ArrayList<SpenContextMenuItemInfo> menu,
            ArrayList<Integer> styleList,  int pressType, PointF point ) {
            if (objectList == null) {
                return false;
            }
            SpenControlBase control = mSpenSurfaceView.getControl();
            if(control != null) {
                control.setContextMenuVisible(true);
            }
            // Add 'Delete' context menu item.
            menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_DELETE,
                   "Delete", true));
            // Add 'Re take' context menu item if the selected object is a container.
            if(objectList.get(0).getType() == SpenObjectBase.TYPE_CONTAINER) {
                menu.add(new SpenContextMenuItemInfo(CONTEXT_MENU_RETAKE,
                       "Re Take", true));
                mStrokeFrameContainer = (SpenObjectContainer) objectList.get(0);
            }
            return true;
        }

        @Override
        public boolean onMenuSelected(
            ArrayList<SpenObjectBase> objectList, int itemId) {
            if (objectList == null) {
```

```
                return true;
            }
            // Delete the selected object (Stroke Frame).
            if (itemId == CONTEXT_MENU_DELETE) {
                mSpenPageDoc.removeSelectedObject();
                mSpenSurfaceView.closeControl();
                mSpenSurfaceView.update();
            // Retake Stroke Frame.
            } else if(itemId == CONTEXT_MENU_RETAKE) {
                SpenControlBase control = mSpenSurfaceView.getControl();
                if(control != null) {
                    control.setContextMenuVisible(false);
                }

                mSpenSurfaceView.retakeStrokeFrame((Activity)mContext,
                      mSpenViewLayout,
                        mStrokeFrameContainer, mStrokeFrameListener);
                mStrokeFrameBtn.setImageResource(R.drawable.tool_ic_stop);
                mStrokeFrameStarted = true;
            }
            return false;
        }

          .........

};
```

4.5.8.3 Registering a Stroke Frame Results Listener

To receive Stroke Frame results:

1. Create an SpenStrokeFrameListener instance by passing it as an input parameter when calling the `SpenSurfaceView.takeStrokeFrame()` or `SpenSurfaceView.retakeStrokeFrame()` methods.

In the `onCompleted()` method, which is called when taking a picture is completed:

- Call `SpenPageDoc.selectObject()` and pass SpenObjectContainer as an input parameter to select the Stroke Frame.
- Call `setContextMenuVisible()` and pass the boolean value true to show the context menu.

```
private SpenStrokeFrameListener mStrokeFrameListener =
            new SpenStrokeFrameListener() {
        @Override
        public void onCompleted(int frameType, SpenObjectContainer o) {
            // When taking the picture is completed, select the object and show the
context menu.
            mSpenPageDoc.selectObject(o);
            mSpenSurfaceView.update();
            mStrokeFrameContainer = o;
```

```
            SpenControlBase control = mSpenSurfaceView.getControl();
            if(control != null) {
                control.setContextMenuVisible(true);
            }
            mStrokeFrameBtn.setImageResource(R.drawable.selector_video);
            mStrokeFrameStarted = false;
        }
```

4.5.9. Using Shape Recognition Plug-ins

You can use Pen for shape recognition and pre-loaded recognition engines.

The sample application implements the following features:

- Creates an SpenSurfaceView instance.
- Calls `SpenShapeRecognitionManager.createRecognition()` to load a shape recognition plug-in.
- Adds stroke objects to SpenSurfaceView. When the Selection Tool button is pressed, and the stroke object to be recognized is selected, the sample application calls `SpenShapeRecognition.request()` to request for shape recognition.
- Replaces the selected stroke object with the SpenObjectStroke recognized as a shape component when the shape recognition output is calculated.

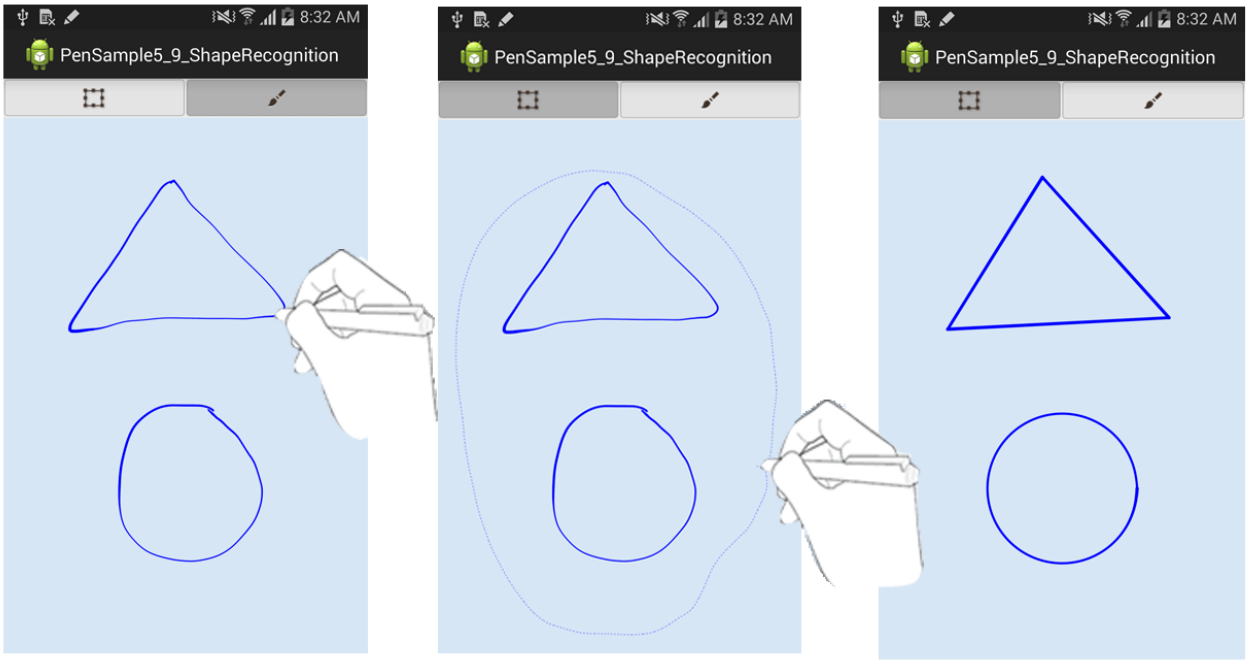


Figure 37: Shape recognition

```
public class PenSample5_9_ShapeRecognition extends Activity {

    private Context mContext;
    private SpenNoteDoc mSpenNoteDoc;
```

```
    private SpenPageDoc mSpenPageDoc;
    private SpenSurfaceView mSpenSurfaceView;
    RelativeLayout mSpenViewLayout;

    private SpenShapeRecognition mShapeRecognition = null;
    private SpenShapeRecognitionManager mSpenShapeRecognitionManager = null;
    private boolean mIsProcessingRecognition = false;

    private ImageView mSelectionBtn;
    private ImageView mPenBtn;

    private Rect mScreenRect;
    private int mToolType = SpenSurfaceView.TOOL_SPEN;

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

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

        mSpenViewLayout = (RelativeLayout) findViewById(R.id.spenViewLayout);

        // Create SpenSurfaceView
        mSpenSurfaceView = new SpenSurfaceView(mContext);
        if (mSpenSurfaceView == null) {
            Toast.makeText(mContext, "Cannot create new SpenSurfaceView.",
                    Toast.LENGTH_SHORT).show();
            finish();
        }
        mSpenViewLayout.addView(mSpenSurfaceView);

        // Get the dimension of the device screen.
        Display display = getWindowManager().getDefaultDisplay();
        mScreenRect = new Rect();
        display.getRectSize(mScreenRect);
        // Create SpenNoteDoc
        try {
            mSpenNoteDoc = new SpenNoteDoc(mContext,
                    mScreenRect.width(), mScreenRect.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
        mSpenSurfaceView.setPageDoc(mSpenPageDoc, true);

        initPenSettingInfo();
        // Register the listener
        mSpenSurfaceView.setControlListener(mControlListener);

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

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

        selectButton(mPenBtn);

        setShapeRecognition();

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

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

    private void setShapeRecognition() {
        // Set ShapeRecognition
        mSpenShapeRecognitionManager = new SpenShapeRecognitionManager(mContext);

        List<SpenRecognitionInfo> shapeRecognitionList =
mSpenShapeRecognitionManager.getInfoList(
                SpenObjectBase.TYPE_STROKE, SpenObjectBase.TYPE_CONTAINER);

        try {
            if (shapeRecognitionList.size() > 0) {
                for (SpenRecognitionInfo info : shapeRecognitionList) {
                    if (info.name.equalsIgnoreCase("SpenShape")) {
                        mShapeRecognition = mSpenShapeRecognitionManager
```

```
                                .createRecognition(info);
                        break;
                    }
                }
            } else {
                finish();
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            Toast.makeText(mContext, "SpenShapeRecognitionManager class not found.",
                    Toast.LENGTH_SHORT).show();
            return;
        } catch (InstantiationException e) {
            e.printStackTrace();
            Toast.makeText(mContext, "Failed to access the
SpenShapeRecognitionManager constructor.",
                    Toast.LENGTH_SHORT).show();
            return;
        } catch (IllegalAccessException e) {
            e.printStackTrace();
            Toast.makeText(mContext, "Failed to access the
SpenShapeRecognitionManager field or method.",
                    Toast.LENGTH_SHORT).show();
            return;
        } catch (SpenCreationFailureException e) {
            // Exit application if the device does not support Recognition feature.
            e.printStackTrace();
            AlertDialog.Builder ad = new AlertDialog.Builder(this);
            ad.setIcon(this.getResources().getDrawable(
                    android.R.drawable.ic_dialog_alert));
            ad.setTitle(this.getResources().getString(R.string.app_name))
                    .setMessage(
                            "This device does not support Recognition.")
                    .setPositiveButton("OK",
                            new DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(DialogInterface dialog,
                                        int which) {
                                    // Close the dialog.
                                    dialog.dismiss();
                                    finish();
                                }
                            }).show();
            ad = null;
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(mContext, "SpenShapeRecognitionManager engine not
loaded.",
                    Toast.LENGTH_SHORT).show();
            return;
        }

        try {
            mShapeRecognition.setResultListener(new ResultListener() {
                @Override
                public void onResult(List<SpenObjectBase> input,
                        List<SpenObjectBase> output) {
                    // Remove the selected objects and append the recognized objects
to pageDoc.
```

```
                    for (SpenObjectBase obj : input) {
                        mSpenPageDoc.removeObject(obj);
                    }

                    for (SpenObjectBase obj : output) {
                        mSpenPageDoc.appendObject(obj);
                    }
                    mIsProcessingRecognition = false;
                    mSpenSurfaceView.closeControl();
                    mSpenSurfaceView.update();
                }
            });
        } catch (IllegalStateException e) {
            e.printStackTrace();
            Toast.makeText(mContext, "SpenShapeRecognition is not loaded.",
                    Toast.LENGTH_SHORT).show();
            return;
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(mContext, "SpenShapeRecognition is not loaded.",
                    Toast.LENGTH_SHORT).show();
            return;
        }
    }

    private final OnClickListener mSelectionBtnClickListener =
            new OnClickListener() {
                @Override
                public void onClick(View v) {
                    selectButton(mSelectionBtn);
            mSpenSurfaceView.setToolTypeAction(mToolType,
SpenSurfaceView.ACTION_SELECTION);
                }
            };

    private final OnClickListener mPenBtnClickListener =
            new OnClickListener() {
                @Override
                public void onClick(View v) {
                    selectButton(mPenBtn);
            mSpenSurfaceView.setToolTypeAction(mToolType,
SpenSurfaceView.ACTION_STROKE);
                }
            };

    private final SpenControlListener mControlListener = new SpenControlListener() {

        @Override
        public boolean onCreated(ArrayList<SpenObjectBase> selectedList,
                ArrayList<Rect> arg1,
                ArrayList<SpenContextMenuItemInfo> arg2,
                ArrayList<Integer> arg3, int arg4, PointF arg5) {
            if (selectedList.size() > 0 && !mIsProcessingRecognition) {
                // List the selected strokes and send the list as a request.
                ArrayList<SpenObjectBase> inputList = new
ArrayList<SpenObjectBase>();
                for (int i = 0; i < selectedList.size(); i++) {
                    if (selectedList.get(i).getType() == SpenObjectBase.TYPE_STROKE)
{
```

```
                        inputList.add(selectedList.get(i));
                    }
                }

                if (inputList.size() <= 0) {
                    return false;
                }
                mIsProcessingRecognition = true;
                try {
                    mShapeRecognition.request(inputList);
                } catch (IllegalStateException e) {
                    e.printStackTrace();
                    Toast.makeText(mContext, "SpenShapeRecognition is not loaded.",
                            Toast.LENGTH_SHORT).show();
                    return false;
                } catch (Exception e) {
                    e.printStackTrace();
                    Toast.makeText(mContext, "SpenShapeRecognition engine not
loaded.",
                            Toast.LENGTH_SHORT).show();
                    return false;
                }
                return true;
            }
            return false;
        }

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

        @Override
        public boolean onMenuSelected(ArrayList<SpenObjectBase> arg0,
                int arg1) {
            return false;
        }

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

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

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

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

        v.setSelected(true);
    }
```

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

        if (mShapeRecognition != null) {
            mSpenShapeRecognitionManager.destroyRecognition(mShapeRecognition);
            mSpenShapeRecognitionManager.close();
        }

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

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

For more information, see PenSample5_9_ShapeRecognition.java in PenSample5_9_ShapeRecognition.

4.5.9.1 Loading Shape Recognition Plug-ins

To load shape recognition plug-ins:

1. Create an SpenShapeRecognitionManager instance.
2. Call `SpenShapeRecognitionManager.getInfoList()` to get the list of shape recognition information objects from the available recognition plug-ins with the specified input and output types. The sample uses Stroke and Container.

Select an appropriate shape recognition plug-in from the list and call `SpenShapeRecognitionManager.createRecognition()` with the associated shape recognition information object to create a shape recognition instance.

```
        // Set ShapeRecognition
        mSpenShapeRecognitionManager = new SpenShapeRecognitionManager(mContext);

        List<SpenRecognitionInfo> shapeRecognitionList =
mSpenShapeRecognitionManager.getInfoList(
                SpenObjectBase.TYPE_STROKE, SpenObjectBase.TYPE_CONTAINER);

        try {
            if (shapeRecognitionList.size() > 0) {
                for (SpenRecognitionInfo info : shapeRecognitionList) {
                    if (info.name.equalsIgnoreCase("SpenShape")) {
                        mShapeRecognition = mSpenShapeRecognitionManager
                                .createRecognition(info);
```

```
                        break;
                    }
                }
            } else {
                finish();
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            Toast.makeText(mContext, "SpenShapeRecognitionManager class not found.",
                    Toast.LENGTH_SHORT).show();
            return;
        } catch (InstantiationException e) {
            e.printStackTrace();
            Toast.makeText(mContext, "Failed to access the SpenShapeRecognitionManager
constructor.",
                    Toast.LENGTH_SHORT).show();
            return;
        } catch (IllegalAccessException e) {
            e.printStackTrace();
            Toast.makeText(mContext, "Failed to access the SpenShapeRecognitionManager
field or method.",
                    Toast.LENGTH_SHORT).show();
            return;
        } catch (SpenCreationFailureException e) {
            // Exit application if the device does not support Recognition feature.
            e.printStackTrace();
            AlertDialog.Builder ad = new AlertDialog.Builder(this);
            ad.setIcon(this.getResources().getDrawable(
                    android.R.drawable.ic_dialog_alert));
            ad.setTitle(this.getResources().getString(R.string.app_name))
                    .setMessage(
                            "This device does not support Recognition.")
                    .setPositiveButton("OK",
                            new DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(DialogInterface dialog,
                                        int which) {
                                    // Close the dialog.
                                    dialog.dismiss();
                                    finish();
                                }
                            }).show();
            ad = null;
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(mContext, "SpenShapeRecognitionManager engine not loaded.",
                    Toast.LENGTH_SHORT).show();
            return;
        }
```

4.5.9.2 Handling Insert Stroke Button Events

To handle Insert Stroke button events:

1. Create an Insert Stroke button.

2. Create an OnClickListener instance named mPenBtnClickListener for the Insert Stroke button and register it by calling `setOnClickListener()` of the button.

In the `onClick()` method of the Insert Stroke button, indicate that the button is selected and set `mToolType` to ACTION_STROKE.

```
    private final OnClickListener mPenBtnClickListener =
            new OnClickListener() {
                @Override
                public void onClick(View v) {
                    selectButton(mPenBtn);
            mSpenSurfaceView.setToolTypeAction(mToolType,
SpenSurfaceView.ACTION_STROKE);
                }
            };
```

4.5.9.3 HandlingSelection Button Events

To handle Selection button events:

1. Create a Selection button.

2. Create an OnClickListener instance named mSelectionBtnClickListener for the Selection button and register it by calling `setOnClickListener()` of the button.

In the `onClick()` method of the Selection button, set`mToolType`to ACTION_SELECTION and indicate that the button is selected.

```
    private final OnClickListener mSelectionBtnClickListener =
        new OnClickListener() {
            @Override
            public void onClick(View v) {
                selectButton(mSelectionBtn);
            mSpenSurfaceView.setToolTypeAction(mToolType,
SpenSurfaceView.ACTION_SELECTION);
            }
        };
```

4.5.9.4 Selecting Objects for Shape Recognition

To select an object for shape recognition:

1. Create a control event listener for stroke object selection in SpenSurfaceView and register it by calling `SpenSurfaceView.setControlListener()`.

In the `onCreated()` method that is called when a control appears on the View, call `SpenShapeRecognition.request()` to request shape recognition and pass the list of selected stroke objects.

Set `isProcessingRecognition` to true to avoid duplicate requests while the shape is being recognized.

```
    private final SpenControlListener mControlListener = new SpenControlListener() {

        @Override
        public boolean onCreated(ArrayList<SpenObjectBase> selectedList,
                ArrayList<Rect> arg1,
                ArrayList<SpenContextMenuItemInfo> arg2,
                ArrayList<Integer> arg3, int arg4, PointF arg5) {
            if (selectedList.size() > 0 && !mIsProcessingRecognition) {
                // List the selected strokes and send the list as a request.
                ArrayList<SpenObjectBase> inputList = new ArrayList<SpenObjectBase>();
                for (int i = 0; i < selectedList.size(); i++) {
                    if (selectedList.get(i).getType() == SpenObjectBase.TYPE_STROKE) {
                        inputList.add(selectedList.get(i));
                    }
                }

                if (inputList.size() <= 0) {
                    return false;
                }
                mIsProcessingRecognition = true;
                try {
                    mShapeRecognition.request(inputList);
                } catch (IllegalStateException e) {
                    e.printStackTrace();
                    Toast.makeText(mContext, "SpenShapeRecognition is not loaded.",
                            Toast.LENGTH_SHORT).show();
                    return false;
                } catch (Exception e) {
                    e.printStackTrace();
                    Toast.makeText(mContext, "SpenShapeRecognition engine not
loaded.",
                            Toast.LENGTH_SHORT).show();
                    return false;
                }
                return true;
            }
            return false;
        }
        ......
```

4.5.9.5 Handling Shape Recognition Events

To handle shape recognition events:

1. Create a ResultListener instance for shape recognition events and register it by calling `SpenShapeRecognition.setResultListener()`.

In the `onResult()` method, do the following:

- Call `SpenPageDoc.removeObject()` to delete the selected stroke objects.
- Call `SpenPageDoc.appendObject()` to add the SpenObjectStroke objects that are recognized as shape to the current page.
- Call `SpenSurfaceView.update()` to refresh the viewport.
- Set `isProcessingRecognition` to false to prevent duplicate recognition requests.

- Call `SpenSurfaceView.closeControl()` to close the control.

```
        try {
            mShapeRecognition.setResultListener(new ResultListener() {
                @Override
                public void onResult(List<SpenObjectBase> input,
                        List<SpenObjectBase> output) {

                    // Remove the selected objects and append the recognized objects
to pageDoc.
                    for (SpenObjectBase obj : input) {
                        mSpenPageDoc.removeObject(obj);
                    }

                    for (SpenObjectBase obj : output) {
                        mSpenPageDoc.appendObject(obj);
                    }
                    mIsProcessingRecognition = false;
                    mSpenSurfaceView.closeControl();
                    mSpenSurfaceView.update();
                }
            });
        } catch (IllegalStateException e) {
            e.printStackTrace();
            Toast.makeText(mContext, "SpenShapeRecognition is not loaded.",
                    Toast.LENGTH_SHORT).show();
            return;
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText(mContext, "SpenShapeRecognition is not loaded.",
                    Toast.LENGTH_SHORT).show();
            return;
        }
```

4.5.9.6 Preventing Memory Leaks

To prevent memory leaks:

1. Call `SpenShapeRecognitionManager.destroyRecognition()` and `SpenShapeRecognitionManager.close()` to unload the shape recognition plug-in.
2. Call `SpenSurfaceView.closeControl()` to close the control.
3. Call `SpenSurfaceView.close()` and `SpenNoteDoc.close()` to close the SpenSurfaceView and SpenNoteDoc instances, respectively.

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

        if (mShapeRecognition != null) {
            mSpenShapeRecognitionManager.destroyRecognition(mShapeRecognition);
            mSpenShapeRecognitionManager.close();
        }
```

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

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

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/