Pass

Programming Guide

Version 1.1.4

Table of Contents

1. Overview

Pass allows you to use fingerprint recognition in your application to identify users whose fingerprints have been registered in the device.

You can use Pass to:

- Provide better application security.
- Increase the convenience of the user identification process.

1.1. Architecture

The following figure shows the Pass architecture.

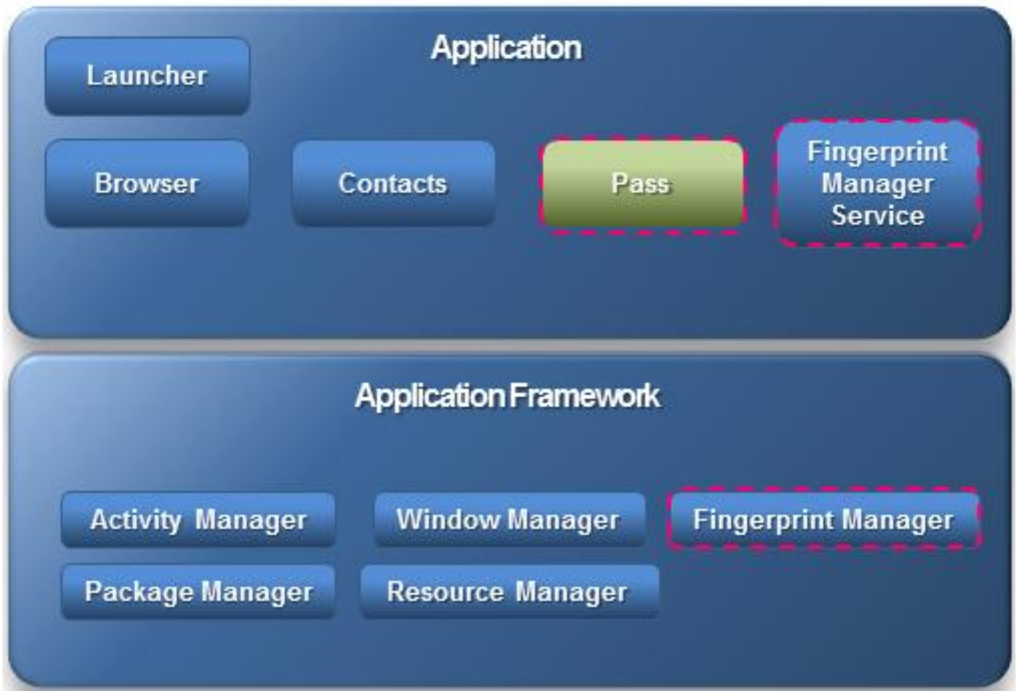

Figure 1: Pass architecture

The architecture consists of:

- **Applications:** One or more applications that use Pass.
- **Pass:** Components for initializing the Pass package.
- **Fingerprint Manager Service:** Service component for fingerprint recognition, registration, and deletion.
- **Fingerprint Manager:** Proxy component for the Fingerprint Manager Service.

1.2. Classes and Interfaces

The Pass classes and interfaces that you can use in your application include:

- **Spass:** Initializes the Pass package.
- **SpassFingerprint:** Manages fingerprint recognition.
- **IdentifyListener:** Listens for fingerprint recognition events.
- **RegisterListener:** Listens for fingerprint registration events.

1.3. Supported Platforms

Android 4.2 (Jelly Bean API 17) or above supports Pass.

1.4. Supported Features

Pass supports the following features:

- Recognizing fingerprints
- Cancelling recognition requests
- Checking whether a registered fingerprint exists on the device
- Registering fingerprints through the Enroll screens of the Setting menu
- Getting the fingerprint index upon recognition success
- Getting a list of names and indexes of registered fingerprints
- Getting a list of names or unique ID of registered fingerprints with their indexes
- Specifying an index for fingerprint recognition
- Adding a title to the user interface
- Adding a logo to the user interface
- Setting the transparency of background elements
- Setting dialog behavior when background elements are touched

1.5. Components

- Components
    - pass-v1.1.3.jar
- Imported packages:
    - com.samsung.android.sdk.pass

1.6. Installing the Package for Eclipse

To install Pass for Eclipse:

1. Add the pass-v1.1.3.jar file to the libs folder in Eclipse.

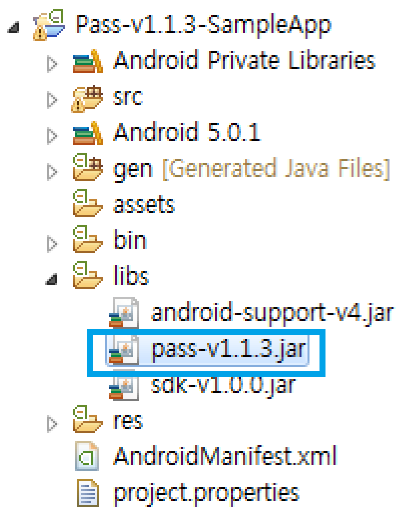

Figure 2: libs folder in Eclipse

The following permission has to be specified in the AndroidManifest.xml file to initialize Pass.

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

If you don't add the permission,

o Android 4.4.2 (KitKat) and above: SecurityException is thrown and your application won't work.

o Prior to Android 4.4.2 (KitKat): No exception and the application will work properly.

2. Hello Pass

Hello Pass is a simple program that:

1. Initializes the Spass class.
2. Requests for fingerprint recognition.
3. Receives fingerprint recognition events from the registered listener.

```
public class HelloPass extends Activity {

       private SpassFingerprint mSpassFingerprint;
       private Context mContext;

       private SpassFingerprint.IdentifyListener listener =
       new SpassFingerprint.IdentifyListener() {

           @Override
           public void onFinished(int eventStatus) {
               // It is called when fingerprint identification is finished.
               if (eventStatus == SpassFingerprint.STATUS_AUTHENTIFICATION_SUCCESS) {
                   // Identify operation succeeded with fingerprint

               } else if (eventStatus == SpassFingerprint.
STATUS_AUTHENTIFICATION_PASSWORD_SUCCESS) {
                   // Identify operation succeeded with alternative password
                               }
                 else {
                  // Identify operation failed with given eventStatus.
                  // STATUS_TIMEOUT_FAILED
                  // STATUS_USER_CANCELLED
                  // STATUS_AUTHENTIFICATION_FAILED
                  // STATUS_QUALITY_FAILED
   }
           }

           @Override
           public void onReady() {
               // It is called when fingerprint identification is ready after
               // startIdentify() is called.
           }

           @Override
           public void onStarted() {
               // It is called when the user touches the fingerprint sensor after
               // startIdentify() is called.
           }
```

```
    };

@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mContext = this;
        mSpass = new Spass();

        try {
            mSpass.initialize(SampleActivity.this);
        } catch (SsdkUnsupportedException e) {
            // Error handling
        } catch (UnsupportedOperationException e){
            // Error handling
        }
        isFeatureEnabled = mSpass.isFeatureEnabled(Spass.DEVICE_FINGERPRINT);

        if(isFeatureEnabled){
            mSpassFingerprint = new SpassFingerprint(SampleActivity.this);
        } else {
            log("Fingerprint Service is not supported in the device.");
        }
}
```

3. Using the Spass Class

The Spass class provides the following methods:

- `initialize()` initializes Pass. You need to initialize the Pass package before you can use it. If the device does not support Pass, SsdkUnsupportedException is thrown.
- `getVersionCode()` gets the Pass version number as an integer.
- `getVersionName()` gets the Pass version name as a string.
- `isFeatureEnabled()` checks if a Pass package feature is available on the device.

```
Spass spass = new Spass();
try {
    spass.initialize(mContext);
} catch (SsdkUnsupportedException e) {
    // Error handling
}

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

3.1. Using the initialize() Method

The `Spass.initialize()` method:

- Initializes the Pass package.
- Checks if the device is a Samsung device.
- Checks if the device supports the Pass package.
- Checks if the Pass package libraries are installed on the device.

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

Initializing should be implemented just once before accessing SpassFingerprint class. If the Pass package fails to initialize, 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()`.

3.3. Checking the Availability of Pass Package Features

You can check if a Pass package feature is supported on the device with the `isFeatureEnabled()` method. The feature types are defined in the `Spass` 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. Feature types include the following:

- DEVICE_FINGERPRINT: Indicates if the device supports the default identification requests and the user interface.
- DEVICE_FINGERPRINT_FINGER_INDEX : Indicates if the API allows you to set a specified fingerprint using its index for fingerprint recognition
- DEVICE_FINGERPRINT_CUSTOMIZED_DIALOG : Indicates the API for defining a custom user interface
- DEVICE_FINGERPRINT_UNIQUE_ID : Indicates the API for getting index and fingerprint unique IDs

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

4. Using the Pass Package

The SpassFingerprint class provides the following methods for fingerprint recognition:

- `startIdentify()`: Requests the identification operation to recognize a fingerprint without a user interface
- `startIdentifyWithDialog()`: Requests the identification operation to recognize a fingerprint with a user interface
- `hasRegisteredFinger()`: Checks whether there are any registered fingerprints on the device
- `cancelIdentify()`: Cancels the identification operation
- `registerFinger()`: Registers a fingerprint on the device through the Enroll screen of the Setting menu
- `getIdentifiedFingerprintIndex():Obtain the fingerprint index of the identified fingerprint when recognition is a success.`
- `getRegisteredFingerprintName(): Obtain a list of  registered fingerprint indexes and names in the Settings menu.`
- `setIntendedFingerprintIndex(): Specifies a fingerprint index for identification before requesting fingerprint recognition`
- setDialogTitle() : Adds a title in the user interface before requesting fingerprint recognition.
- setDialogIcon() : Adds a logo in the bottom left portion of the UI before requesting fingerprint recognition.
- setDialogBgTransparency() : Sets the transparency of background elements before requesting fingerprint recognition.

  setCanceledOnTouchOutside() : Sets a value that determines whether the user interface is dismissed or not when touching background elements before requesting fingerprint recognition.

4.1. Checking for Registered Fingerprints on the Device

To check whether a registered fingerprint for the current user exists on the device, call `hasRegisteredFinger()`.

The method returns true if a registered fingerprint exists on the device and returns false if no registered fingerprint is found.

```
    boolean mHasRegisteredFinger = mSpassFingerprint.hasRegisteredFinger();
```

4.2. Requesting Fingerprint Recognition

To request the identification operation to recognize a fingerprint without a user interface (UI), call `startIdentify()`. Before calling this method, initialize an IdentifyListener, which provides the following events:

- `onReady()`: Called when the fingerprint sensor is in the ready state
- `onStarted()`: Called when the user touches the fingerprint sensor and starts to swipe their finger
- `onFinished()`: Called when the identify operation is completed

To request the identify operation with a UI, call `startIdentifyWithDialog()`.

If your application uses the identify operation with a UI, you have the additional option of using a password instead of a fingerprint for identification. To show the **Password** button on the UI, set `enablePassword` to true. To hide the button and use only fingerprint identification, set `enablePassword` to false.

Both `startIdentify` () and `startIdentifyWithDialog` () methods permits only 5 attempts for recognizing a fingerprint. After 5 failed attempts,fingerprint recognition requests are blocked.

- In the case of `startIdentify` () :
  - If there is another request, `SpassInvalidStateException` is thrown and a notification for enabling fingerprint recognition is displayed in the notification bar

  .
- In the case of `startIdentifyWithDialog` () with password :
  - After 5 failed attempts or another request is made, the screen is changed to password recognition and the password recognition event is delivered to the app.

- In the case of `startIdentifyWithDialog` () without password :
  - After 5 failed attempts, the screen is shut down and a notification for enabling fingerprint recognition is displayed in the notification bar.
  - If another request is made, the password recognition screen is displayed first for enabling fingerprint recognition and doesn't deliver any event to app regarding password recognition. Afterwards, the screen is changed backed to the fingerprint recognition as per the original request.

Password verification requests are unlimited after 5 failed fingerprint attempts, however, a 30 second timer will be introduced per request if an `SpassInvalidStateException occurs.`

When the identify operation is requested, the fingerprint sensor enters the Ready state. When the `startIdentify()` or `startIdentifyWithDialog()` operations are completed, the `onFinished()` method of IdentifyListener is called and passes the event status that allows you to verify the result of the identify operation:

- If there is no activity for 20 seconds after the fingerprint sensor enters the Ready state, the request is canceled, and the `onFinished()` method is called with the event status `STATUS_TIMEOUT_FAILED`.
- If the backup password matches, the event status is `STATUS_AUTHENTIFICATION_PASSWORD_SUCCESS`.
- If the fingerprint recognition fails, the event status defines the reason for the failure.

  STATUS_USER_CANCELLED,

  STATUS_QUALITY_FAILED,

  STATUS_SENSOR_FAILED,

STATUS_USER_CANCELLED_BY_TOUCH_OUTSIDE,

STATUS_AUTHENTIFICATION_FAILED

```
mSpassFingerprint.startIdentifyWithDialog(SampleActivity1.this, listener, false);

private SpassFingerprint.IdentifyListener listener =
   new SpassFingerprint.IdentifyListener() {
       @Override
       public void onFinished(int eventStatus) {
           // It is called when fingerprint identification is finished.
       }

       @Override
       public void onReady() {
           // It is called when fingerprint identification is ready after
           // startIdentify() is called.
       }

       @Override
       public void onStarted() {
           // It is called when the user touches the fingerprint sensor after
           // startIdentify() is called.
       }
};
```

4.3. Cancelling Fingerprint Recognition

To cancel the fingerprint recognition request that started the identify operation through `startIdentify()` or `startIdentifyWithDialog()`, call `cancelIdentify()`.

```
mSpassFingerprint.cancelIdentify();
```

4.4. Registering Fingerprints

If there are no registered fingerprints on the device, you can directly access the Settings menu to register fingerprints by calling `registerFinger()`. Before calling this method, initialize a RegisterListener. To return to the previous screen after the fingerprint registration completes, call `onFinished()` of RegisterListener.

```
mSpassFingerprint.registerFinger(this, mRegisterListener);


   private SpassFingerprint.RegisterListener mRegisterListener = new
SpassFingerprint.RegisterListener() {


      @Override
      public void onFinished() {
            log("RegisterListener.onFinished()");

        }
    };
```

4.5. Getting the Fingerprint Index upon Recognition Success

After fingerprint recognition is finished, you can get the fingerprint index of the identified fingerprint. This API must be called inside IdentifyListener.onFinished() in order for it to return the fingerprint index. Otherwise, it returns -1 to indicate invalid value. If the users authenticate using a password, the API returns 0.

```
mSpassFingerprint.getIdentifiedFingerprintIndex();
```

```
private SpassFingerprint.IdentifyListener listener =
    new SpassFingerprint.IdentifyListener() {
        @Override
        public void onFinished(int eventStatus) {
            int FingerprintIndex = mSpassFingerprint.getRecognizedFingerprintIndex();
            // It is called when fingerprint identification is finished.
        }
```

4.6. Getting the Registered Fingerprint Index and Name

Return a list of indexes and names of registered fingerprints in the Settings menu.

```
mSpassFingerprint.getRegisteredFingerprintName ();
```

4.7. Getting the Registered Fingerprint Index and unique ID

Return a list of indexes and unique IDs of registered fingerprints in the Settings menu if isFeatureEnabled(Spass.DEVICE_FINGERPRINT_UNIQUE_ID) return true.

```
if (mSpass.isFeatureEnabled(Spass.DEVICE_FINGERPRINT_UNIQUE_ID)){
  mSpassFingerprint. getRegisteredFingerprintUniqueID ()
}
```

4.8. Specifying an Index for Fingerprint Recognition

You can specify a fingerprint index for identification before requesting fingerprint recognition by calling the setIntendedFingerprintIndex() method of the SpassFingerprint class if isFeatureEnabled(Spass.DEVICE_FINGERPRINT_FINGER_INDEX) return true.

If the parameter to this method is set as null or no index, then the function finds a match against all registered fingerprints. If the parameter to this method is a valid index, then the function finds a match against the designated fingerprint.

```
mSpassFingerprint.setIntendedFingerprintIndex(ArrayList<Integer>);
```

```
ArrayList<Integer> designatedFingers = new ArrayList<Integer>();
                                designatedFingers.add(2);
                                designatedFingers.add(3);

if (mSpass.isFeatureEnabled(Spass.DEVICE_FINGERPRINT_FINGER_INDEX)){
   mSpassFingerprint. setIntendedFingerprintIndex (designatedFingers);
}
mSpassFingerprint.startIdentifyWithDialog(SampleActivity.this,listener, true);

private SpassFingerprint.IdentifyListener listener =
    new SpassFingerprint.IdentifyListener() {
    ...
```

4.9. Adding a title to the UI

This feature allows you to add the customized title on top of the UI before requesting fingerprint recognition.

If isFeatureEnabled(Spass. DEVICE_FINGERPRINT_CUSTOMIZED_DIALOG) returns true, then this function can be activated.

Calling the setDialogTitle() method from the SpassFingerprint class allows you to set a string as a custom title. If the string is null, then the title would not be displayed in the UI.

```
mSpassFingerprint.setDialogTitle(String titleText, int titleColor);
```

```
if(mSpass.isFeatureEnabled(Spass.DEVICE_FINGERPRINT_CUSTOMIZED_DIALOG)){
   mSpassFingerprint.setDialogTitle("Customized Dialog", 0xff0000);
}
mSpassFingerprint.startIdentifyWithDialog(SampleActivity.this,listener, true);
```

```
private SpassFingerprint.IdentifyListener listener =
   new SpassFingerprint.IdentifyListener() {
   ...
```

4.10. Adding a Logo to the UI

This feature allows you to add a customized Logo on the bottom left corner of the UI before requesting fingerprint recognition. You can use the setDialogIcon() method of the SpassFingerprint class, isFeatureEnabled (Spass. DEVICE_FINGERPRINT_CUSTOMIZED_DIALOG) returns true.

The parameter of the function accepts a file name which is stored in /res/Drawable/… of the device. If the parameter is set to null or the image cannot be found, then the method displays the default dialog without the icon.

```
mSpassFingerprint.setDialogIcon(String iconName);
```

```
if(mSpass.isFeatureEnabled(Spass.DEVICE_FINGERPRINT_CUSTOMIZED_DIALOG)){
   mSpassFingerprint.setDialogIcon("ic_launcher");
}
mSpassFingerprint.startIdentifyWithDialog(SampleActivity.this,listener, true);

private SpassFingerprint.IdentifyListener listener =
   new SpassFingerprint.IdentifyListener() {
   ...
```

4.11. Setting the Transparency in Background elements

This feature allows you to set transparency of background elements when the user interface is active before requesting fingerprint recognition. Since this is a custom UI feature, isFeatureEnabled(Spass. DEVICE_FINGERPRINT_CUSTOMIZED_DIALOG) must return true so that you can use this feature.

If there is no value assigned to the setDialogBgTransparency() method of the SPassFingerprint class, then the method assumes default transparency.

```
mSpassFingerprint.setDialogBgTransparency(int transparency);
```

```
if(mSpass.isFeatureEnabled(Spass.DEVICE_FINGERPRINT_CUSTOMIZED_DIALOG)){
   mSpassFingerprint.setDialogBgTransparency(0);
}
mSpassFingerprint.startIdentifyWithDialog(SampleActivity.this,listener, true);

private SpassFingerprint.IdentifyListener listener =
    new SpassFingerprint.IdentifyListener() {
    ...
```

4.12. Setting Dialog Behavior when Background Elements are Touched

This feature allows you to set whether the UI is dismissed or not when touching background elements. The setCanceledOnTouchOutside() method from the SPassFingerprint class accepts a boolean value and it must be set before requesting fingerprint recognition in order for it to work. This also requires isFeatureEnabled(Spass.DEVICE_FINGERPRINT_CUSTOMIZED_DIALOG) to return true for activation.

If the parameter is true, then the UI is dismissed and you can receive STATUS_USER_CANCELLED_BY_TOUCH_OUTSIDE as event status in IdentifyListener.onFinished() when touching background elements. If the parameter is false, the UI stays on screen regardless of interactions with background elements.

```
setCanceledOnTouchOutside(boolean cancel);
```

```
if(mSpass.isFeatureEnabled(Spass.DEVICE_FINGERPRINT_CUSTOMIZED_DIALOG)){
   mSpassFingerprint.setCanceledOnTouchOutside(true);
}
mSpassFingerprint.startIdentifyWithDialog(SampleActivity.this,listener, true);

private SpassFingerprint.IdentifyListener listener =
    new SpassFingerprint.IdentifyListener() {
    ...
```

Copyright