Pass

Programming Guide

Version 1.0

Revision History

Version	Date	Description
1.0		

Table of Contents

1. Overview

Pass allows you to use fingerprint recognition in your application to identify the user 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.

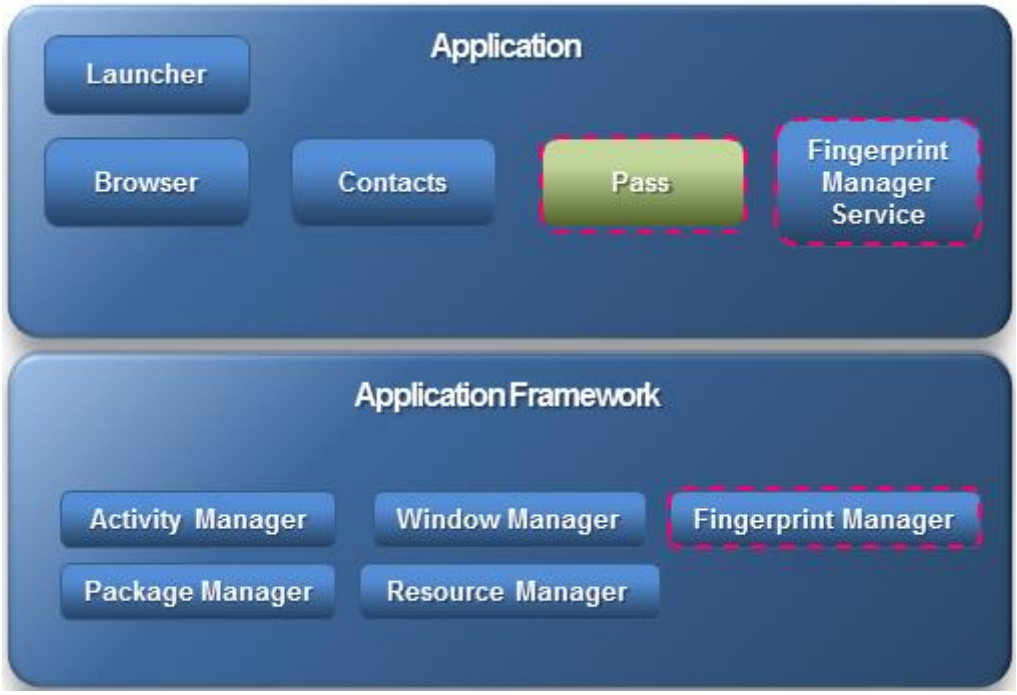

Figure 1: Pass architecture

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 support 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

1.5. Components

- Components
    - pass-v1.0.0.jar
- Imported packages:
    - com.samsung.android.sdk.pass

1.6. Installing the Package for Eclipse

To install Pass for Eclipse:

1. Add the pass-v1.0.0.jar file to the libs folder in Eclipse.

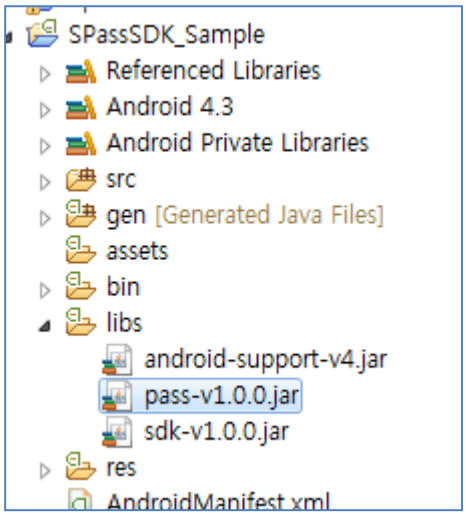

Figure 2: libs folder in Eclipse

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;
            mSpassFingerprint = new SpassFingerprint(HelloPass.this);

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

            mSpassFingerprint.startIdentify(listener);
        }
    }
```

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
```

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.

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

4. Using the Pass Package

The SpassFingerprint class provides the following methods for fingerprint recognition:

- `startIdentify()`: Requests the identify operation to recognize a fingerprint without a user interface
- `startIdentifyWithDialog()`: Requests the identify operation to recognize a fingerprint with a user interface
- `hasRegisteredFinger()`: Checks whether there are any registered fingerprints on the device
- `cancelIdentify()`: Cancels the identify operation
- `registerFinger()`: Registers a fingerprint on the device through the Enroll screen of the Setting menu

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 identify operation to recognize a fingerprint without a user interface (UI), call `startIdentify()`. Before calling this method, register 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.

Unlike `startIdentify()`, the `startIdentifyWithDialog()` method permits 5 attempts for recognizing a fingerprint and 3 attempts for entering a password.

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 alternative password matches, the event status is `STATUS_AUTHENTIFICATION_PASSWORD_SUCCESS`.
- If the fingerprint recognition fails, the event status defines the reason for the failure.

```
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 to the Setting menu to register fingerprints by calling `registerFinger()`. Before calling this method, register the 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()");

         }
   };
```

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/