ROAM

POWERING MCOMMERCE | An Ingenico Company

Android Software Development Kit

Online Payments

280 Summer Street, Lobby Level, Boston, MA 02210
Phone: +1.888.589.5885 | www.roamdata.com

Table of Contents

Android Software Development Kit

This documentation provides knowledge in developing Stripe mobile payment inside any Android app. If you need help or have any questions after reading this documentation, we recommend you to check out our answers for common questions or contact other developers in #stripe on freenode.

Stripe has created a Java library for Android, allowing you to easily submit payments from an Android app. Our library eliminates the need to send card data directly to your server. Instead, it sends the card data directly to our servers, where we can convert them to tokens.

Your app will receive the token back, and will then be able to send the token to an endpoint on your server, where it can be used to process a payment, establish recurring billing, or merely saved for later use.

Stripe supports Android back to version 4 (Ice Cream Sandwich), and the library has no external dependencies.

Installation

There is a difference in installing the Stripe Android library depending on whether you use Android Studio or Eclipse.

Android Studio

Add the following code to your app's build.gradle file, inside the dependencies section:

```
compile 'com.stripe:stripe-android:+'
```

Eclipse

1. Download the stripe-android libraries.
2. Be sure you have installed the Android SDK with a minimum of API Level 17 and android-support-v4.
3. Import the stripe folder into Eclipse.
4. In your project settings, add the 'stripe' project under the "Libraries" section of the "Android" category.

Collecting Credit Card Information

At some point in the flow of your app, you will want to obtain payment details from the user. There are 2 ways to do this:

- Use Android Pay to access your customer's stored card information

- Build your own credit card form

We recommend you to write your app to offer support for both.

Using Android Pay

Through Android Pay, you can access payment information stored in your customer's Google accounts. Following are the instructions to integrate your app with Android Pay.

> **Note:** When this documentation was released, Android Pay was still in Beta Version.

Setting Up Your App

First, you will need to obtain credentials and a client ID for your app, as explained in the Android Pay API Tutorial. You will also need to set up the latest version of Google Play services.

Collecting Payment Information Through Android Pay

To use Android Pay in your app, first enable the Android Pay API by adding the following code to the <application> tag of your AndroidManifest.xml:

```
<application
  ...
  <meta-data
    android:name="com.google.android.gms.wallet.api.enabled"
    android:value="true" />
</application>
```

In the activity's layout, your application will also need a SupportWalletFragment (the placeholder for the Android Pay purchase button). To create this, add the following code to your program:

```
<!-- You will need to add the wallet namespace to your enclosing Layout -->
xmlns:wallet="http://schemas.android.com/apk/res-auto"

<fragment
  android:id="@+id/wallet_fragment"
  android:name="com.google.android.gms.wallet.fragment.SupportWalletFragment"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  wallet:environment="test"
  wallet:fragmentMode="buyButton"/>
```

After placing the fragment, you need to:

5. Grab a reference to that fragment
6. Create a MaskWalletRequest
7. Initialize the fragment

In the MaskWalletRequest, you are able to specify the amount to charge and what additional information you would like to collect (e.g., the shipping address). This is also where you can specify that you are using Stripe as the processor. Doing so will allow the application to request a Stripe token directly from the wallet.

Before starting the Android Pay flow, use the isReadyToPay() method to check whether the user has the Android Pay app installed and is ready to pay through it. Make sure you have already mastered Google's documentation for information on their UI and branding requirements.

The following code is the Android Pay flow:

```
public class PaymentActivity extends FragmentActivity implements
GoogleApiClient.ConnectionCallbacks,
    GoogleApiClient.OnConnectionFailedListener {

  // You will need to use your live API key even while testing
  public static final String PUBLISHABLE_KEY = "pk_live_XXX";

  // Unique identifiers for asynchronous requests:
  private static final int LOAD_MASKED_WALLET_REQUEST_CODE = 1000;
  private static final int LOAD_FULL_WALLET_REQUEST_CODE = 1001;
  private SupportWalletFragment walletFragment;

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Wallet.Payments.isReadyToPay(googleApiClient).setResultCallback(
      new ResultCallback<BooleanResult>() {
        @Override
        public void onResult(@NonNull BooleanResult booleanResult) {
          if (booleanResult.getStatus().isSuccess()) {
            if (booleanResult.getValue()) {
              showAndroidPay();
            } else {
              // Hide Android Pay buttons, show a message that Android Pay
              // cannot be used yet, and display a traditional checkout button
            }
          } else {
            // Error making isReadyToPay call
            Log.e(TAG, "isReadyToPay:" + booleanResult.getStatus());
          }
        }
      });
    ...
  }

  public void showAndroidPay() {
    setContentView(R.layout.payment_activity);
    walletFragment = (SupportWalletFragment) getSupportFragmentManager()
      .findFragmentById(R.id.wallet_fragment);

    MaskedWalletRequest maskedWalletRequest = MaskedWalletRequest.newBuilder()
      // Request credit card tokenization with Stripe:
      .setPaymentMethodTokenizationParameters(
        PaymentMethodTokenizationParameters.newBuilder()
          .setPaymentMethodTokenizationType(PaymentMethodTokenizationType.PAYMENT_GATEWAY)
          .addParameter("gateway", "stripe")
```

```
            .addParameter("stripe:publishableKey", PUBLISHABLE_KEY)
            .addParameter("stripe:version", com.stripe.Stripe.VERSION)
            .build())
        .setShippingAddressRequired(true)
        .setEstimatedTotalPrice("20.00")
        .setCurrencyCode("USD")
        .build();

    WalletFragmentInitParams initParams = WalletFragmentInitParams.newBuilder()
      .setMaskedWalletRequest(maskedWalletRequest)
      .setMaskedWalletRequestCode(LOAD_MASKED_WALLET_REQUEST_CODE)
      .build();

    walletFragment.initialize(initParams);
    ...
  }

  public void onStart() { ... }
  public void onStop()  { ... }

  @Override
  protected void onActivityResult(int requestCode, int resultCode, Intent data) { ... }
  @Override public void onConnectionFailed(ConnectionResult connectionResult) {}
  @Override public void onConnected(Bundle bundle) {}
  @Override public void onConnectionSuspended(int i) {}
}
```

> **Note:** The price set within the Android app is written as a decimal and is for the Android app only. The token received back will be sent to your server, and the charge request will be made of the Stripe API from there. The actual amount to be charged is requested at that point, and is set as an integer.

The last step of the Android Pay setup process for the app is to connect to the Google Wallet API. This connection handles the case when a user presses the Android Pay purchase button and the payment is processed. To do so, add the following code:

```
public class PaymentActivity extends FragmentActivity {
  ...
  private GoogleApiClient googleApiClient;

  public void onCreate(Bundle savedInstanceState) {
    ...
    googleApiClient = new GoogleApiClient.Builder(this)
      .addConnectionCallbacks(this)
      .addOnConnectionFailedListener(this)
      .addApi(Wallet.API, new Wallet.WalletOptions.Builder()
        .setEnvironment(WalletConstants.ENVIRONMENT_TEST)
        .setTheme(WalletConstants.THEME_LIGHT)
        .build())
      .build();
  }

  public void onStart() {
    super.onStart();
    googleApiClient.connect();
  }
```

```
  public void onStop() {
    super.onStop();
    googleApiClient.disconnect();
  }
}
```

Once your customer confirms their purchase, the application then needs to create a FullWalletRequest. This will allow the application to request access to the customer's FullWallet, which is where the payment information comes from. To do this, implement onActivityResult:

```
public class PaymentActivity extends FragmentActivity {
  ...
  @Override
  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == LOAD_MASKED_WALLET_REQUEST_CODE) {
      if (resultCode == Activity.RESULT_OK) {
        MaskedWallet maskedWallet = data.getParcelableExtra(WalletConstants.EXTRA_MASKED_WALLET);
        FullWalletRequest fullWalletRequest = FullWalletRequest.newBuilder()
          .setCart(Cart.newBuilder()
            .setCurrencyCode("USD")
            .setTotalPrice("20.00")
            .addLineItem(LineItem.newBuilder()
              .setCurrencyCode("USD")
              .setQuantity("1")
              .setDescription("Premium Llama Food")
              .setTotalPrice("20.00")
              .setUnitPrice("20.00")
              .build())
            .build())
          .setGoogleTransactionId(maskedWallet.getGoogleTransactionId())
          .build();
        Wallet.Payments.loadFullWallet(googleApiClient, fullWalletRequest, LOAD_FULL_WALLET_REQUEST_CODE);
      }
    } else if (requestCode == LOAD_FULL_WALLET_REQUEST_CODE) {
      ...
    }
  }
}
```

> **Note:** The above example has only one item, but if your customer is purchasing multiple items, you can add multiple items by calling addLineItem additional times.

Creating Tokens from Android Pay

Once your customer allows access to their wallet for payment, the application will be given back a Stripe token. You will then send this token to your server for use through the API.

```
public class PaymentActivity extends FragmentActivity {
  ...
  // Keep track of your current environment.
  // Change to WalletConstants.ENVIRONMENT_PRODUCTION when ready to go live.
  public static final int mEnvironment = WalletConstants.ENVIRONMENT_TEST;

  @Override
```

```
  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == LOAD_MASKED_WALLET_REQUEST_CODE) {
      ...
    } else if (requestCode == LOAD_FULL_WALLET_REQUEST_CODE) {
      if (resultCode == Activity.RESULT_OK) {
        FullWallet fullWallet =
data.getParcelableExtra(WalletConstants.EXTRA_FULL_WALLET);
        String tokenJSON = fullWallet.getPaymentMethodToken().getToken();
        // A token will only be returned in production mode,
        // i.e. WalletConstants.ENVIRONMENT_PRODUCTION
        if (mEnvironment == WalletConstants.ENVIRONMENT_PRODUCTION) {
          com.stripe.model.Token token = com.stripe.model.Token.GSON.fromJson(
            tokenJSON, com.stripe.model.Token.class);
          // TODO: send token to your server
        }
      }
    } else {
      super.onActivityResult(requestCode, resultCode, data);
    }
  }
}
```

Testing and Deploying with Android Pay

To test your Android Pay flow, use your live Stripe API key, not your test key, in conjunction with the Android Pay test environment, specified by WalletConstants.ENVIRONMENT_TEST.

In test mode, fullWallet.getPaymentMethodToken().getToken() will return the string "TEST_GATEWAY_TOKEN" in place of a JSON string representing a token.

If you want to test your application on a physical device, make sure the device supports NFC. You will also need to add a support credit card to your Android Pay account.

When you are ready, you can get production access to Android Pay by submitting your APK to Google for review.

Building Your Own Form

If you plan to build your own form, make sure you will at least be able to collect your customer's card numbers and expiration dates. We recommend you to also collect the CVC to prevent fraud. The user's name and billing address are optional and would benefit you in terms of fraud protection.

Once you have collected a customer's information, you will need to exchange the information for a Stripe token.

Creating and Validating Cards from a Customer Form

The first step is to import the Stripe classes before using them:

```
import com.stripe.android.*;
```

There are two main classes: Card and Stripe. The Card class contains many useful helpers for validating card input on the client-side before a charge is created.

To construct a Card instance with your customer's payment information, add the following code to your program:

```
Card card = new Card(
  cardNumber,
  cardExpMonth,
  cardExpYear,
  cardCVC
);
card.validateNumber();
card.validateCVC();
```

The Card instance contains helpers to validate that:

- the card number passes the Luhn check
- the expiration date is in the future
- the CVC looks valid

You will probably want to validate these three things at once, so we have included a validateCard function that does so:

```
Card card = new Card("4242-4242-4242-4242", 12, 2017, "123");
if ( !card.validateCard() ) {
  // Show errors
}
```

Creating Tokens from a Customer Form

The next step is to pass off that sensitive payment information securely to Stripe, where you will exchange it for a token.

You can create tokens using the Stripe instance method createToken, passing in a Card instance and completion callbacks. An asynchronous network request will be executed, and the appropriate callback invoked when it completes.

```
Card card = new Card("4242424242424242", 12, 2017, "123");
Stripe stripe = new Stripe("pk_test_6pRNASCoBOKtIshFeQd4XMUh");

stripe.createToken(
  card,
  new TokenCallback() {
    public void onSuccess(Token token) {
      // Send token to your server
```

```
      }
      public void onError(Exception error) {
        // Show localized error message
        Toast.makeText(getContext(),
          error.getLocalizedString(getContext()),
          Toast.LENGTH_LONG
        ).show();
      }
    }
  );
```

We have placed your test publishable API key as the first argument to Stripe. You will need to swap it out with your live publishable key in production. You can see all your keys after logging into your Stripe dashboard.

Using Tokens

Using the payment token requires an API call from your server using your secret API key. For security purposes, you should never embed your secret API key in your app.

To do so, you need to set up an endpoint on your server that can receive an HTTP POST call for the token. In the onActivityResult method or the onSuccess callback, you will need to POST the supplied token to your server. Make sure any communication with your server is SSL secured to prevent eavesdropping.