Getting Started Guide

PayPal Here™ Android SDK

PROPRIETARY AND CONFIDENTIAL

Document edition:
v1.0, 07/31/13

PayPal™

Contents

Overview

The PayPal Here Android SDK enables your app to create transactions (invoices) and accept payments. The PayPal Here app is an example of a mobile app used by merchants.

For **sample code**, see Payments with the Sample App.

Your app can accept payments using:

- **A credit card reader (by swiping a card).** Your app can enable a merchant to take a credit card payment with card readers ("peripherals"). PayPal provides a card reader that plugs into the audio jack of a mobile device. The card reader provides "card present data" to your app.
- **Manual credit card entry.** When a merchant manually enters data from a card (and the card is not swiped), the data is "key-in" data, also called "card not present data." Fees for key-in are **higher** than fees for swipes; see the PayPal Here FAQs.
- **PayPal payments (check-ins).** A merchant's location is displayed in a prospective customer's Local directory in their PayPal mobile app. After a customer checks in (opens a tab) by indicating that they are at a merchant's location, the merchant can charge the customer's PayPal account.

Note: The functionality described in this document is subject to change without notice.

The SDK enables many use cases for:

- Partners who sell an app, e.g. a point-of-sale app, to merchants
- Merchant partners who are building an app for their own use

A partner who sells apps to merchants generally must enable a merchant to create an account with that partner. For example, assume that "Dixon's Apps" is a partner. If Dixon's Apps sells an app to a merchant named "Smith's Shoes," Dixon's Apps must enable Smith's Shoes to have an account with Dixon's Apps. If Smith's Shoes already has a PayPal Here account, Smith's Shoes can link that PayPal Here account to their Dixon's Apps account.

If Smith's Shoes has a PayPal account but no PayPal Here account, Smith's Shoes can create a PayPal Here account and link it to their Dixon's Apps account. Similarly, if Smith's Shoes does not have any type of PayPal account, Smith's Shoes can create a PayPal Here account and link it to their Dixon's Apps account.

The first time a Smith's Shoes user signs into your app, you would authenticate the Smith's Shoes user to your account system. Then you would send the user to PayPal for Log In with PayPal (PayPal Access) authentication. Finally, you would link the two accounts when the user is returned your URL. For more information, see Authentication with the SDK.

Reviewing the SDK Code and Samples

To begin using the SDK, please contact your relationship manager to get added to the GitHub repository. Then clone the repository for your own use. For sample code, see the following section of this document: Payments with the Sample App.

As you browse the javadoc delivered with the SDK, note that the major SDK classes (`TransactionManager`, `PeripheralsManager`, etc.) are at the top level of the `com.paypal.merchant.sdk` package. The `PayPalHereSDK` class is used to interact with those major classes. Other important classes, such as the `Merchant` class, are found primarily in the `com.paypal.merchant.sdk.domain` package.

The SDK includes a **sample app**, called *PayPalHereSampleApp*, and a sample server. The source files of the sample app are commented. A sample merchant, called "teashop," is hard-coded into the sample files.

To start developing an app:

- Review this Getting Started Guide and review the javadoc delivered with the SDK.
- See Payments with the Sample App.
- Review the code comments in the source files of the sample app and sample server.

Set up the Permissions

Among the permissions needed by your app (from the Android operating system) is the location permission. The merchant's location is used in SDK flows. During the installation of your app, the user should grant your app permission to access the location of the mobile device.

Other permissions are needed as well. The following is the list of permissions to include your *AndroidManifest.xml* file:

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

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

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

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

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

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

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

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

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

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

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

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

Initialization and Authentication

This section introduces the `PayPalHereSDK` class. **Please review the javadoc (which is delivered with the SDK) for information about all of the SDK classes.** Also see Example Use Case: Credit Card Reader.

The `PayPalHereSDK` class is the main class for interacting with the following:

- Other major SDK classes

- Hardware devices, such as card readers and chip & devices

Initializing the SDK

You can initialize a `PayPalHereSDK` object by specifying the environment (`PayPalHereSDK.LIVE` or `PayPalHereSDK.SANDBOX`):

```
PalPalHereSDK.init(myApplication, PayPalHereSDK.Sandbox);
```

Authenticating a Merchant

To authenticate a merchant, prompt the merchant to log in to your app and perform the OAuth process with Log In with PayPal (PayPal Access). Retrieve a token, a refresh URL, and a token expiration value. Typically, you would prompt the merchant to log in to your app every time the merchant opens your app. For more information, see Payments with the Sample App.

See the `Credentials` class in the javadoc (which is delivered with the SDK) for how to create and initialize a `Credentials` object. Based on the access token, create a new `Credentials` object and start the setup of an active merchant. The `Credentials` object's `ExpiredAccessTokenHandler` method notifies your app when an access token is not valid, e.g. due to token expiration, so you can obtain a new token.

The `MerchantManager` class is successfully initialized when a valid `Credentials` object is set in the `PayPalHereSDK` using the `setCredentials` method. The `MerchantManager` class utilizes an active merchant who is changed only when that merchant logs out of your app and another merchant logs in. Uses of the `MerchantManager` class include setting up and managing the merchant's current location, as well as handling check-ins.

You should set up an active merchant every time a merchant enters your app, and provide meaningful feedback to the merchant if errors are encountered.

Payments, Shopping Carts, and Refunds

This section introduces the `TransactionManager` class and introduces refunds. For an example of taking payments and using a shopping cart, see Payments with the Sample App.

Note that the SDK enables an app to store multiple carts at once, e.g. for multiple customers. After you use the `beginPayment` method (of the `TransactionManager` class), your app can save the shopping cart, and when your app is ready to resume using the cart, you can call the `setShoppingCart` method (of the `TransactionManager` class). Then your app can add items to the cart and complete the transaction with the `finalizePayment` method (of the `TransactionManager` class).

Taking a Payment

The `TransactionManager` class is used to take payments and process refunds. The class is stateful and attempts to take a payment with the shopping cart it is holding. When a payment is made, a transaction record is returned.

A `TransactionManager` object works with the `PeripheralsManager` to emit events that are intended for display to the merchant user. Events are emitted to any `TransactionListener` object that is registered with the `TransactionManager` object. Thus, callbacks need to be defined and implemented. For example, when taking a payment, your app needs to handle certain events, such as `NEED_SIG`. A callback from the SDK to your app indicates that the SDK requires an action to be completed by the merchant. Your app's user interface should tell the user the required action.

Payments can be made for a "fixed amount" by specifying the amount to be charged, the mode of payment (e.g., credit card, using a card reader), and a callback handler for the status. The following example takes a payment for 42.50 in the Merchant's default currency:

```
transactionManager.beginPayment(new BigDecimal("42.50"));

transactionManager.finalizePayment(PaymentType.CARD_READER,
mResponseHandler);
```

Alternatively, in a **shopping cart**, payments are made for a list of items. For example:

```
ShoppingCart cart = transactionManager.beginPayment();

cart.addItem(apples, 6);

cart.addItem(juice, 1);

cart.addItem(orange, 6);

transactionManager.setShoppingCart(cart);

transactionManager.finalizePayment(PaymentType.CARD_READER,
mResponseHandler);
```

There are additional methods in `TransactionManager` for card-not-present payments, payments from checked-in users, and more; see the `TransactionManager` in the javadoc delivered with the SDK.

As shown in the *CreditCardPeripheralActivity.java* file in the SDK sample app, the response handler (`mResponseHandler`, above) implements `onSuccess` and `onError` methods, as follows:

```
PaymentResponseCompletionHandler mPaymentResponseHandler = new
PaymentResponseCompletionHandler() {

        // If the transaction went through successfully…
```

```
        public void onSuccess(PaymentResponse response) {

            // Displaying a success message in the UI.

            updateUIForPurchaseSuccess(response);

            // We can now enable the refund option as well.

            mRefundButton.setVisibility(View.VISIBLE);

            // We can now take another payment.

            mAnotherTransButton.setVisibility(View.VISIBLE);

        }

        // If an error occurred while completing the transaction…

        @Override

        public void onError(PaymentError e) {

            // Display the error message in the UI.

            updateUIForPurchaseError(e);

            // In case of an error, the transaction-related data (such as the
shopping cart, etc.) gets removed. Hence we might need to reinstate the data,
to continue with the same transaction.

            reInstateShoppingCart();

            showButtons(true);

        }

    };
```

Making a Refund

For a refund, use the `doRefund` method of the `TransactionManager` class. When you use the `doRefund` method, specify the original transaction record, the amount of the refund, and a response handler. (When the original payment was made, an original transaction record was returned.) A new transaction record will be returned. The following is an example from the SDK sample app:

```
    private void doRefund() {

        // Check for a previous transaction (for which a refund is needed).

        // If there is one, make sure the transaction record and the amount
are recorded.
```

```
        if (mAmount == null || mTransactionRecord == null) {

            CommonUtils.createToastMessage(CreditCardPeripheralActivity.this,
"Invalid refund operation!");

            return;

        }

        displayPaymentState("Performing refund...");

        // Call the Refund API in the SDK, to perform the refund.

        PayPalHereSDK.getTransactionManager().doRefund(mTransactionRecord,
mAmount, mDefaultResponseHandler);

}
```

The sample implements a response handler, for the response status:

```
DefaultResponseHandler mDefaultResponseHandler = new DefaultResponseHandler()
{

        // If the refund went through successfully…

        @Override

        public void onSuccess() {

            displayPaymentState("Refund successful.");

            // Disable the refund button when the refund is complete.

            mRefundButton.setVisibility(View.GONE);

        }

        // If there are issues during the refund operation…

        @Override

        public void onError(com.paypal.merchant.sdk.domain.PPError e) {

            displayPaymentState("Refund failed.");

        }

    };
```

Working with the PayPal Here API

The SDK contains libraries for working with the PayPal Here API. Information is available in the *Developer Guide for the PayPal Here API*.

The PayPal Here API is a RESTful interface with the following characteristics:

- The data-interchange format is JSON.
- Each request must contain an authentication token; see Authentication with the SDK.
- The base URI is the following:
  `https://www.paypal.com/webapps/hereapi/merchant/v1`

Prerequisites for Testing Apps

- Contact with your relationship manager, for the latest SDK information.
- Android Gingerbread (a minimum Android SDK version of 10).
- The PayPal Here credit-card reader, if your app will utilize the reader.

Prerequisites for Production Apps

- Create a Log In with PayPal (PayPal Access) app:
  - Obtain an App ID (from developer.paypal.com) and secret; the App ID is also called a Client ID
- For app authentication in production, a backend server is needed to store your secret
- A merchant who uses your app must have either a PayPal Business or PayPal Premier account

Example Use Case: Credit Card Reader

After an app initializes the SDK, the app can enable a merchant to take a payment with a credit card reader. The app could use the following steps:

1. Access the `TransactionManager` interface and register a `PaymentEventListener` object to be notified of status messages related to the payment. The `TransactionManager` interface is stateful and thus saves data between calls.
2. Access the `PeripheralsManager` and register a `PeripheralsListener` object to be notified of status messages related to the card reader device.
3. Create a transaction, or "summary of charges," to present to the customer.
4. Initiate a transaction by calling `TransactionManager.beginPayment`. If you specify an amount, the payment is a fixed payment. If you do not specify an amount, the transaction is based on a shopping cart.
5. If necessary, prompt the user to connect a card reader. Use the `PeripheralsListener` object to tell the user when a card can be swiped. Also notify the user when transaction processing begins, and if necessary, prompt for a signature.
6. Notify the user when the transaction has been processed.

Authentication with the SDK

In the PayPal Here API, the OAuth 2.0 standard is used. Your app's API calls must include a Log In with PayPal (PayPal Access) authentication token to indicate the merchant on whose behalf the calls are made. Thus, you need a test user, one of which is the test merchant delivered with the sample app.

For sample code, see Payments with the Sample App.

Creating a Log In with PayPal App

To use the sample server and app delivered with the SDK, you need an App ID (client ID) and secret. You can obtain these by creating an app on the PayPal Developer site. When you create the app, specify the following capability: Log In with PayPal.

Contact your relationship manager for a special API scope that must be added to your App ID. You also must pass a scope identifier on each request for an access (bearer) token; the scope for PayPal Here is the following URI, which is an identifier, not a link:

```
https://uri.paypal.com/services/paypalhere
```

When you are ready to get your own Log In with PayPal App ID (same as client ID) and secret, see the *Developer Guide for the PayPal Here API* and see LIPP Integration Details. Note that for LIPP, you must provide a Return URL, and the user is not sent directly back to your mobile app after a login.

Storing the App Secret

You should **not** store your app secret on a mobile device due to concerns that it could be jail-broken or otherwise compromised. (If your app secret is compromised, barriers are raised in your ability to provide updates to users.) Thus, you need a back-end server to store your app secret.

Although you can use Log In with PayPal (LIPP) as your sole point of authentication, you probably have an existing account system. As shown in the sample server delivered with the SDK, you would:

- Authenticate your users to your account system,
- Send them to PayPal, and then
- Link up the accounts when users are returned your URL.

Overview of Your Back-End Server

The purposes of your backend server are:

- Provide a way to link your existing account system with merchants' PayPal accounts

- Avoid the need to store an app secret on a mobile device, due to security concerns

The sample server delivered with the SDK is easily deployed on the Heroku site, and is designed to be easily modified to run elsewhere. The sample server runs on Node.js, an environment and library for running server-side javascript programs. You can download Node.js and begin using the files in the *\sample_server\heroku_server* folder.

As shown in the sample server delivered with the SDK, your back-end server will send users to a *paypal.com* location for authentication. When a user is redirected back to a URL of your back-end server (see Endpoints Supported by the Sample Server), a token is included, which in turn is exchanged for an access token. A refresh token also is included, to help avoid re-authentication.

Calls to the PayPal Here API should be made from the mobile device. The access token is kept on the mobile device (encrypted using a random value—a "ticket"—created when a user logs in for authentication for the first time).

In the *\sample_server\heroku_server\config.js* file, the `exports.CENTRAL_SERVER_MODE` value is set to true (which is the default for the sample server delivered with the SDK). (Since the value is true, the server can refresh access tokens for mobile clients.) The encrypted access token is stored on the mobile device. However, an API call to refresh a token is proxied through the back-end server.

Endpoints Supported by the Sample Server

The sample server delivered with the SDK implements four REST service endpoints:

1. **/login** -- A dummy version of user authentication. Returns a ticket that can be used in place of a password to reassure you that the person you're getting future requests from is the same person that entered their password in your app.
2. **/goPayPal** -- Validates the ticket and returns a URL which your app can open in start the Log In with PayPal (LIPP) flow. This method specifies the OAuth scopes you're interested in, which must include the PayPal Here scope, defined as *https://uri.paypal.com/services/paypalhere*. (Note that if authentication already was completed, then the oAuth access token is returned.)
3. **/goApp** – The endpoint to which PayPal returns the user (the same redirect URL that was specified when the LIPP app was created), after the user completes authentication. This **/goApp** endpoint inspects the result of authentication and redirects back to your app.

   The server encrypts the access token received from PayPal using the client ticket, so if someone hijacks your app's URL handler, the data isn't usable; the data is not the data sent by the LIPP flow. The server returns a URL to your app that allows the app to refresh its access token when necessary. This URL is the **/refresh** URL (see below), and includes the refresh token issued by PayPal, encrypted with an account-specific server secret. The refresh token is never stored on the server, and is not stored in a directly usable form on the client either. This minimizes the value of

centralized data on your server, and allows you to cut off refresh tokens in cases of compromised tokens.

4. **/refresh/username/token** - Decrypts the refresh token and calls the token service to get a new access token, given the refresh token.

When you are ready to set up a server with your App ID, set up your return URL in LIPP to point to your server. Assuming you want to test on a device, this URL generally needs to work on that device and on your simulator, meaning you need a live DNS entry on the internet.

Payments with the Sample App

You can step through the code in the SDK sample app, which is in the *PayPalHereSampleApp* folder. Also see the following:

- Initialization, Credentials, and Payments
- Example Use Case: Credit Card Reader
- Setting up the Sample Server
- Setting up the Sample App

This section contains code examples (from the SDK sample app) for the following:

1. Initializing the SDK, authenticating a merchant, setting a merchant's location, and starting a transaction
2. Taking a payment using:
    - A credit card reader
    - Manually-entered data from a credit card
    - PayPal payments (check-ins)
3. Displaying a transaction ID to a merchant

Running the code below assumes that the right permissions have been enabled for an app.

Step 1: Initialize the SDK

File name in the SDK sample app: LoginScreenActivity.java

```
// Initialize the SDK with the application context and server environment
name.

// This init is necessary because the SDK needs the app context to init a few
underlying objects.

// The two options available are "Live" and "Sandbox"

        PayPalHereSDK.init(getApplicationContext(), "Sandbox");
```

Step 2: Authenticate a merchant and pass the credentials to the SDK

File name in the SDK sample app: OAuthLoginActivity.java

The sample app makes calls to the back-end server, and the steps depend on whether there is an existing token that has not expired. **In both cases**, the merchant enters the username and password **for the sample app** (which is different from the PayPal username and password), and the sample app send these values to the back-end server. The back-end server provides the ticket ID and merchant information (username, business name, address, etc.) to the sample app.

If there is an existing access token that has not expired: The back-end server uses the ticket ID and the merchant username to provide the existing access token (for subsequent API calls).

If there is no existing token, or if the token has expired: The back-end server provides a LIPP URL. This LIPP URL is displayed to the merchant and the merchant enters their **PayPal** username and password. Upon successful log in, the following values are returned to the sample app: the access token (for subsequent API calls), the token expiration date, and the token refresh URL; see Receiving the Encrypted Access Token.

After the sample app has retrieved the access token, the app creates a `Credentials` object and sends that object to a `PayPalHereSDK` object. This does the following: an `init` of the merchant with the SDK and a merchant check-in, for check-in based transactions:

```
    private void setMerchantAndCheckIn(String accessToken) {

        // Create a credentials object, based on the decrypted access token.

        // Also implement a callback listener in case the access token is
expired.

        Credentials credentials = new Credentials(accessToken,

                new Credentials.ExpiredAccessTokenHandler() {

                    @Override

                    public void accessTokenExpired(String oldAccessToken) {

                        // TODO: refresh the token

                    }

                });

        // Display a message that indicates the merchant is being set up.

        showInitializingMerchantDialog();

        Log.d("Access Token", accessToken);
```

```
        // Init the SDK with the current merchant credentials.

        PayPalHereSDK.setCredentials(credentials, new MerchantManager

                .MerchantManagerResponseHandler()
```

Step 3: Set the merchant's location

File name in the SDK sample app: OAuthLoginActivity.java

In the sample app, the merchant's location is set automatically when the merchant logs in. Although your app can separately provide an option to "check in" a merchant, the sample code checks in the merchant right after the merchant successfully logs in.

```
         // Check in the merchant based on geo-location provided by the
device.

         PayPalHereSDK.getMerchantManager().checkinMerchant("Joe's Pizza",

                              new
MerchantManager.MerchantManagerResponseHandler() {

                                  @Override

                                  public void onSuccess(Merchant merchant) {

                                      Log.d("PayPal", "Merchant check in
successful.");

                                      Log.d("PayPal",
merchant.getCheckedInMerchant().getLocationId());

                                  }

                                  @Override

                                  public void onError(java.lang.Error error) {

                                      Log.e("PayPal", "Merchant check in
Unsuccessful.");

                                      Log.e("PayPal", error.getMessage());

                                  }

           });
```

Step 4: Begin an itemized (shopping cart) transaction

File name in the SDK sample app: ItemizedActivity.java

The following sample includes the `beginPayment` method (of the `TransactionManager` class), which starts a transaction. In subsequent steps, your app can add items to a shopping cart, and then complete a transaction by calling the `finalizePayment` method (of the `TransactionManager` class).

```
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        // Setting the layout for this activity.

        setContentView(R.layout.activity_itemized_tab_type);

        // For the itemized payment, call the beginPayment() method to

        // indicate to the SDK an itemized payment.

        // NOTE: Once the beginPayment method is invoked, the customer is
allowed to swipe in their card at any point.

        // This card data would be held by the SDK and would be used during
the payment.

        // This feature is mainly aimed to offer flexibility as to when the
card could be swiped.

        // When a card swipe is detected, a SecureCreditCard object is also
returned back by the SDK, which an app can choose to store.

        // Once the itemized payment is initialized in the SDK, get the
shopping cart object created by the SDK and add items to the same.

        mShoppingCart = PayPalHereSDK.getTransactionManager().beginPayment();
```

Step 5: Adding more items to the shopping cart

File name in the SDK sample app: ItemizedActivity.java

```
        CartItem ci = mStoreItems.get(name);

        // Add the item into the shopping cart, with the quantity 1.

        // The quantity is updated as we keep adding or removing items.

        // To remove an item, use: mShoppingCart.addItem(ci, new Long(-1));

        mShoppingCart.addItem(ci, new Long(1));
```

Step 4-5 Alternative: Fixed amount transaction

File name in the SDK sample app: FixedPriceActivity.java

Payments can be made for a fixed amount by specifying the amount to be charged, the mode of payment (e.g., card reader), and a callback handler that will be notified of the status.

```
// Inform the SDK of the fixed price transaction and provide the amount.

// This would create a shopping cart with one item having the given amount.

// Once the beginPayment method is invoked, the customer is allowed to swipe
in their card at any point.

// This card data would be held by the SDK and would be used during the
payment.

// This feature is mainly aimed to offer flexibility as to when the card
could be swiped.

// When a card swipe is detected, a SecureCreditCard object is also returned
back by the SDK,

// which the apps can choose to store.

ShoppingCart sc =
PayPalHereSDK.getTransactionManager().beginPayment(mFixedPriceVal);
```

Step 6: Take a payment using a credit card reader

File name in the SDK sample app: CreditCardPeripheralActivity.java

a. After you call `PayPalHereSDK.getTransactionManager().beginPayment`, the merchant can swipe a card. The sample app listens for a card swipe by implementing a `PeripheralsListener` object:

```
/**

 * This activity is meant to complete the transaction either by connecting to
a PayPalHere card reader or the PayPalHere Bond device to take in the credit
card information.

 * We would need to implement the TransactionListener to listen to
payment/transaction-related actions such as payment complete, error in
transaction etc.

 * We would also need to implement the PeripheralsListener to listen to card
swipes within this activity.

 */

public class CreditCardPeripheralActivity extends MyActivity implements
TransactionListener,

        PeripheralsListener {
```

```
    private static final String LOG = "PayPalHere.CreditCardPeripheral";

    private static final int SIGNATURE_ACTIVITY_REQ_CODE = 654;
```

b. Receive events from a `PeripheralsManager` using methods in the `PeripheralsListener` object:

```
/**

* This method is invoked by the SDK when it detects a reader being connected
to the device.

* @param readerType; indicates whether this is a magnetic stripe reading
device or a more advanced device.

* @param transport; indicates how the reader is connected to this device
(audio jack, etc.).

*/

public void onPaymentReaderConnected(Constants.ReaderTypes readerType,
Constants.ReaderConnectionTypes transport) {

    displayPaymentState("Reader is connected! Please swipe your card.");

    }

/**

* This method is invoked by the SDK when it detects a reader is disconnected.

     * @param readerType

*/

    public void onPaymentReaderDisconnected(Constants.ReaderTypes readerType)
{

        displayPaymentState("Reader is disconnected!");

    }



    /**

     * This method is invoked by the SDK when it detects a card swipe.

     *

     * @param paymentCard

     */
```

```
    public void onCardReadSuccess(SecureCreditCard paymentCard) {

        // The SDK passes the card data back to the app. The data could be
stored by the app, as in the case of multiple card swipes.

        displayPaymentState("Card read successful. Click Purchase to complete
the transaction.");

        // adding the swiped card data into a list.

        mCreditCardList.add(paymentCard);

        // Update the UI showing the list of swiped cards.

        mCreditCardAdapter.notifyDataSetChanged();

        // Make this list visible on the UI screen.

        mCreditCardLayout.setVisibility(View.VISIBLE);

    }

    /**

     * This method is invoked by the SDK in case of failure while trying to
read the card data.

     *

     * @param reason

     */

    public void onCardReadFailed(Constants.FailureCodes reason) {

        displayPaymentState("Bad swipe.  Try again!");

    }

    /**

     * This method is invoked by the SDK to indicate certain card-reader-
related events.

     *

     * @param e

     */

    @Override

    public void onPeripheralEvent(PeripheralEvent e) {

        PeripheralEventType type = e.getEventType();
```

```
        if (type == PeripheralEventType.BAD_SWIPE)
            displayPaymentState("Bad Swipe!  Try Again.");
        else if (type == PeripheralEventType.CARD_BLOCKED)
            displayPaymentState("Card blocked!  Please use another card.");
        else if (type == PeripheralEventType.CARD_INVALID)
            displayPaymentState("Invalid Card!  Please use another.");
    }
    /**
     * This method is meant to display the given text on the UI screen.
     *
     * @param state
     */
    private void displayPaymentState(String state) {
        Log.d(LOG, "state: " + state);
        TextView tv = (TextView) findViewById(R.id.purchase_status);
        tv.setText(state);
    }
    /**
     * This method is meant to check whether any card reading device is
connected to the device.
     *
     * @return
     */
    private void checkForPeripheralDevices() {
        // Get the list of devices/readers that the device is connected to.
        ArrayList<ReaderTypes> readerList =
PayPalHereSDK.getPeripheralsManager().availableReaders();
        if (readerList == null)
            displayPaymentState("Please connect a card reader!.");
```

```
        else if
(readerList.contains(Constants.ReaderTypes.MagneticCardReader))

            displayPaymentState("Reader is connected! Please swipe your
card.");

    }
```

c. If your app receives card data from the swipe, you can call the `finalizePayment` method of the `TransactionManager` class. The SDK automatically decides whether to make a fixed price transaction or an itemized transaction, based on the `BeginPayment` type:

```
PayPalHereSDK.getTransactionManager().finalizePayment(PaymentType.CARD_READER
, mPaymentResponseHandler);
```

d. The payment response handler (`mPaymentResponseHandler`) returns the status of the payment, e.g. whether there was an error (in which case an error message is returned) or success (in which case a transaction record is returned):

```
    PaymentResponseCompletionHandler mPaymentResponseHandler = new
PaymentResponseCompletionHandler() {

        // If the transaction went through successfully.

        public void onSuccess(PaymentResponse response) {

            // Displaying a success message on the UI.

            updateUIForPurchaseSuccess(response);

            // We can now enable the refund option as well.

            mRefundButton.setVisibility(View.VISIBLE);

            // We can now take another payment.

            mAnotherTransButton.setVisibility(View.VISIBLE);

        }

        // If an error occurred while completing the transaction.

        @Override

        public void onError(PaymentError e) {

            // Display the error message onto the UI.

            updateUIForPurchaseError(e);
```

```
            // In case of an error, the transaction-related data (such as the
shopping cart, etc.) gets removed. Hence,

            // we might need to re-instate the data back to continue with the
same transaction.

            reInstateShoppingCart();

            showButtons(true);

        }

    };
```

Step 6 Alternative: Take a payment using manual credit card entry

File name in the SDK sample app: CreditCardManualActivity.java

```
    /**

     * Method to take a payment from the keyed in credit card info.

     */

    private void takePayment() {

        // Create a ManualEntryCardData object by providing the credit card.

        ManualEntryCardData manualEntryCardData = new
ManualEntryCardData(mCCInfo,

                mExpDate.getText().toString(), mCCVInfo);

        // Set the Card Holder's name.

        manualEntryCardData.setCardHoldersName("John Doe");

        // Hide the purchase button

        mPurchaseButton.setVisibility(View.GONE);

        displayPaymentState("Taking Payment...");

        // Call the SDK to take a payment.

        // **NOTE**: The transaction state i.e., the shopping cart, the
transaction extras,

        // previously read credit cards, etc., would be kept intact only
between the begin - finalize payments. If the

        // payment goes through successfully or if it returns back with a
failure,
```

```
        // all the above-mentioned objects are removed, and the app would
need to call beginPayment again to

        // re-init, set the shopping cart back (which they would be holding
onto) and try again.


PayPalHereSDK.getTransactionManager().finalizePayment(manualEntryCardData,
mPaymentResponseHandler);

    }
```

The response handler is similar to the response handler for taking a payment with a credit card reader:

```
    /**

     * Implementing a PaymentResponseHandler to handle the response status of
a transaction.

     */

    PaymentResponseCompletionHandler mPaymentResponseHandler = new
PaymentResponseCompletionHandler() {

        public void onSuccess(PaymentResponse response) {

            updateUIForPurchaseSuccess(response);

            mAnotherTransButton.setVisibility(View.VISIBLE);

        }

        @Override

        public void onError(PaymentError e) {

            updateUIForPurchaseError(e);

            // In case of an error, the transaction-related data (such as the
shopping cart, etc.) gets removed. Hence,

            // we might need to re-instate the data back to continue with the
same transaction.

            reInstateShoppingCart();

            mPurchaseButton.setVisibility(View.VISIBLE);

        }

    };
```

Step 6 Alternative: Take a PayPal payment

File name in the SDK sample app: PayPalMerchantCheckinActivity.java

In the sample app, this step assumes that at least one customer has checked into the merchant's location. The merchant retrieves a list of customers and selects the customer who will make a payment.

a. Get list of checked-in customers:

```
    /**
     * This method retrieves the list of checked-in customers by invoking an
API in the SDK.
     */
    private void getCheckedInClientList() {
        // Invoke the API to retrieve the list of checked-in customers.
        // Provide a handler in order to receive the response.
        PayPalHereSDK.getMerchantManager().getCheckedInClientsList(new
CheckedInClientsListUpdateHandler() {
            // if the list was obtained successfully, display on the UI
screen.
            @Override
            public void onSuccess(List<CheckedInClient> arg0) {
                displayClientList(arg0);
                hideProgressDialog();
            }
            @Override
            public void onError(Error arg0) {

CommonUtils.createToastMessage(PayPalMerchantCheckinActivity.this, "Error
while retrieving the list.");
                hideProgressDialog();
            }
        });
    }
```

b. Select a customer:

```
    /**
     * This method is called to display the list of checked-in clients on the
UI.
     *
     * @param clientList
     */
    private void displayClientList(final List<CheckedInClient> clientList) {
        mGridViewAdapter = new ClientGridViewAdapter(this, clientList);
        mClientGridView.setAdapter(mGridViewAdapter);
        mClientGridView.setOnItemClickListener(new OnItemClickListener() {
            // If a customer is selected, get the "CheckedInClient" object
and use the same to take the payment.
            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1, int
position, long arg3) {
                CheckedInClient client = clientList.get(position);

CommonUtils.createToastMessage(PayPalMerchantCheckinActivity.this,
                        "You've selected: " + client.getClientsName());
                takePaymentWithCheckedInClient(client);
            }
        });
    }
```

c. Take a PayPal payment:

```
    /**
     * This method is invoked to take a payment with the checked-in customer.
     *
```

```
     * @param client

     */

    private void takePaymentWithCheckedInClient(CheckedInClient client) {

        displayPaymentState("Taking payment... ");

        // Invoke the API with the checked-in customer to take the payment.

        // **NOTE**: The transaction state, i.e. the shopping cart, the
transaction extras,

        // previously read credit cards, etc., would be kept intact only
between the begin - finalize payments. If the

        // payment goes through successfully or if it returns back with a
failure,

        // all the above mentioned objects are removed and the app would need
to call beginPayment once again to

        // re-init, set the shopping cart back (which they would be holding
onto) and try again.

        PayPalHereSDK.getTransactionManager().finalizePayment(client,
mPaymentResponseHandler);

    }
```

Step 7 Display the transaction ID to the merchant

File name in the SDK sample app: CreditCardPeripheralActivity.java

```
    /**

     * This method updates the UI screen in case of a successful transaction.

     * We display a transaction ID on the UI.

     * @param response

     */

    private void updateUIForPurchaseSuccess(PaymentResponse response) {

        // Setting these values for refund-related operations.

        mTransactionRecord = response.getTransactionRecord();

        mAmount =
response.getTransactionRecord().getShoppingCart().getGrandTotal();
```

```
        displayPaymentState("Payment completed successfully!  TransactionId:
" + response.getTransactionRecord()

                .getTransactionId());

    }
```

Setting up the Sample Server

Since the SDK sample server runs on Node.js, download Node.js. Set up the sample server on the Heroku site. For more information, see Overview of Your Back-End Server. The sample server includes a test merchant.

Please review the code comments in the following files:

- *\sample_server\heroku_server\config.js*
- *\sample_server\heroku_server\server.js*
- *\sample_server\heroku_server\lib\oauth.js*

Setting up the Sample App

The first time you run the sample app (simulating a merchant user), you log into the sample server with a merchant username of `teashop` and a password of `11111111`. You are then redirected to paypal.com URL to log in with your PayPal username. This process links teashop's PayPal account to the sample app's App ID.

After the link is done, you (teashop) are returned to the URL of the sample server, which completes authentication and then redirects to the sample app; see Endpoints Supported by the Sample Server. For additional information, see the code comments in *\sample_server\heroku_server\server.js*. The sample app now can use the SDK to directly call the PayPal Here API.

See *\sample_server\heroku_server\users.db*; this is the nStore database file for storing a merchant's username and password, and eventually for storing the access and refresh tokens (nStore is a database for node.js).

Receiving the Access Token

The first time you run the sample app (located in *\PayPalHereSampleApp*), the app receives an encrypted access token and a refresh URL (that is, a URL to the sample server that can be used to refresh the token).

See the following section for more information: Step 2: Authenticate a merchant and pass the credentials to the SDK.

Note: The *\heroku_server\sample_server\server.js* file has a default, test merchant named "teashop," whose address is a confirmed address in teashop's PayPal account.

The sample app (*PayPalHereSampleApp*) has sample code (in *OAuthLoginActivity.java*) for initiating the SDK with the credentials of a merchant:

```
        // Init the SDK with the current merchant credentials.
        PayPalHereSDK.setCredentials(credentials, new MerchantManager
                .MerchantManagerResponseHandler() {
            @SuppressLint("NewApi")
            @Override
            public void onSuccess(Merchant merchant) {
                mMerchantInitDialog.dismiss();
                // Check to see if the merchant is already checked in.
                // In this sample app, we achieve this by storing the checked
in merchant location ID as a shared preference.
                if (!isMerchantLocationSaved()) {
                    // Check-in the merchant based on the geo-location
provided by the device.
                    PayPalHereSDK.getMerchantManager().checkinMerchant("Joe's
Pizza",
                            new
MerchantManager.MerchantManagerResponseHandler() {
                                @Override
                                public void onSuccess(Merchant merchant) {
                                    Log.d("PayPal", "Merchant check-in
successful.");
                                    Log.d("PayPal",
merchant.getCheckedInMerchant().getLocationId());
                                }
                                @Override
                                public void onError(java.lang.Error error) {
                                    Log.e("PayPal", "Merchant check-in
Unsuccessful.");
                                    Log.e("PayPal", error.getMessage());
```

```
                                }

                            });

                }

                // Hide the progress dialog.

                hideProgressDialog();

                // Ask the merchant to fill in the address information, which
would be set in the SDK and would be used for merchant payments.

                openAddressDialog();

            }


            @Override

            public void onError(Error arg0) {

                Log.e(LOG,

                        "Merchant init failed : "

                                + arg0.getMessage());

            }

        });

    }
```

After you have initiated the SDK with the credentials of a merchant, you can retrieve a merchant's data for an order (transaction) with the `getActiveMerchant` method of the `MerchantManager` class. For more information, see the `MerchantManager` class in the javadoc delivered with the SDK.

Running the Sample App

After you complete the steps in Setting up the Sample App, you can step through the code in the sample app while you accept a test payment.

To use the app to accept a swiped credit card, run the app on an actual mobile device that has the PayPal card reader.

If a signature is required, the current SDK release requires capture of the signature before submission of a payment transaction; it is planned that a future release will not require this initial capture of a signature.