



***Mobile Payments Library
Developer Guide and
Reference –
Android OS Edition***

PayPal Mobile Payments Developer Guide and Reference – Android OS Edition

Document Number 10116.en_US-201610

© 2011 - 2016 PayPal, Inc. All rights reserved. PayPal is a registered trademark of PayPal, Inc. The PayPal logo is a trademark of PayPal, Inc. Other trademarks and brands are the property of their respective owners. The information in this document belongs to PayPal, Inc. It may not be used, reproduced or disclosed without the written approval of PayPal, Inc.

Copyright © PayPal. All rights reserved. PayPal S.à r.l. et Cie, S.C.A., Société en Commandite par Actions.

Registered office: 22-24 Boulevard Royal, L-2449, Luxembourg, R.C.S. Luxembourg B 118 349

Consumer advisory: The PayPal™ payment service is regarded as a stored value facility under Singapore law. As such, it does not require the approval of the Monetary Authority of Singapore. You are advised to read the terms and conditions carefully.

Notice of non-liability:

PayPal, Inc. is providing the information in this document to you "AS-IS" with all faults. PayPal, Inc. makes no warranties of any kind (whether express, implied or statutory) with respect to the information contained herein. PayPal, Inc. assumes no liability for damages (whether direct or indirect), caused by errors or omissions, or resulting from the use of this document or the information contained in this document or resulting from the application or use of the product or service described herein. PayPal, Inc. reserves the right to make changes to any information herein without further notice.

Contents

Preface	5
Purpose.....	5
Scope.....	5
OS and Hardware Support	5
Revision History	5
Where to Go for More Information	7
 1. PayPal Mobile Payments Library	 8
Mobile Payments Library API Reference.....	8
Declaring the Library and Permissions in AndroidManifest.xml	8
Adding the Library Jar File and Importing Classes	9
Required Methods in the Mobile Payments Library.....	10
Optional Methods in the Mobile Payments Library	16
Activity Results for the Mobile Payments Library	16
After the Payment	18
Instant Payment Notification	18
Transaction Details.....	18
Refunds	18
Simple, Parallel, and Chained Payments	19
Preapprovals.....	21
How Preapprovals Work.....	21
About Preapproval Keys	21
About Preapproval Pins	22
Sample Call	22
Custom Objects in the Mobile Payments Library.....	23
Enumerated Values in the Mobile Payments Library	28
Localization Support in the Mobile Payments Library	29
 2. The Checkout Experience with the Mobile Payments Library.....	 31
Checkout Experience #1 – Goods or Services with Shipping	31
Checkout Experience #2 – Goods or Services without Shipping	32
Checkout Experience #3 – Donations	33
Checkout Experience #4 – Personal Send Money Payments.....	34
Checkout Experience #5 – Create Pin	35
Checkout Experience #6 – Preapproval	36
Basic Preapproval Checkout.....	36
Creating Preapproval PINs During Preapproval Checkout	37

3. Submitting Your Application to PayPal.....	38
A. Currencies Supported by PayPal.....	39
B. Countries and Regions Supported by PayPal	41

Preface

The PayPal Mobile Payments library provides secure, extensible, and scalable PayPal payment functionality to the Android platform.

Important: The Mobile Payments Library is based on the PayPal Adaptive Payments API. As of October 6, 2016, Adaptive Payments is now a limited release product. It is restricted to select partners for approved use cases and should not be used for new integrations without guidance from PayPal.

Purpose

The PayPal Mobile Payments Library provides an easy way for you to integrate payments into your Android applications. You can download the library from the [PayPal GitHub repository](#) and include it in your application. With the library, you need only a few lines of code to integrate the payments library with your application.

When a buyer makes a payment, the library controls the checkout experience – logging in, reviewing, and completing the payment. After buyers complete their payments, the library returns the buyer to your application.

Scope

This document describes how to integrate the PayPal Mobile Payments Library with your application. You must create and provide your build to PayPal so PayPal can review your application before it is approved to accept payments via the library. The approval process is described later in the document.

OS and Hardware Support

The PayPal Mobile Payments Library supports Android OS 1.5 and higher. All Android devices are supported.

Revision History

The following table lists revisions made to the *Mobile Payments Library Developer Guide and Reference*.

Version	Date Published	Description
1.5.6	October 2016	Added a note to the preface about the Adaptive Payments API, which is now a limited-release API.
1.5.6	Aug 2016	Various documentation updates.

1.5.5	July 2011	Initialization is recallable and completes within 3 sec down from 20-30 sec. “Keep Me Logged” in functionality reinstated. Merchant has the ability cancel transaction from dynamic amount calculation.
1.5	February 2011	Improved Tablet Support
1.1.1	January 2011	Eliminated “Keep Me Logged In” functionality
1.1	December 2010	Added information about preapproval; dropped support for the enumeration value BUTTON_118x24
1.0	September 2010	Adaptive Payments Support
0.7	May 2010	First publication

Where to Go for More Information

- [Adaptive Payments integration information](#)
- [Sandbox user guide](#)
- [Merchant setup and administration guide](#)
- [PayPal Developer portal](#)

1. PayPal Mobile Payments Library

This section provides details about the Mobile Payments Library API, and it provides instructions and examples for integrating the library with your Android application.

Mobile Payments Library API Reference

The flow of the library is:

1. Your application initializes the library.
2. The library creates a **Pay with PayPal** Button and returns it to you so you can place it on the screen.
3. *(Optional)* Your application enables Dynamic Amount Calculation (see step 4 below) to recalculate the payment amount, tax, currency, and shipping values when buyers change the shipping address for the payment.
4. When buyers select the **Pay with PayPal** button, the library takes them through the PayPal Checkout experience.
5. *(Optional)* If you enabled dynamic amount calculation in step 1 above:
 - a. When a buyer chooses an address for the payment, the library returns a call back to your application with the address information.
 - b. Your application recalculates the payment and other amounts, based on the address.
 - c. The library returns the buyer to the checkout experience, which uses the updated payment amount, tax, currency, and shipping values.
6. After buyers complete their payments, the library returns a callback to your application with the status of the payment and the pay key. Note: at this time, the library is still in control of the UI and has not returned control to your application.
7. After the library flow is complete, an activity result is posted for your application.

Declaring the Library and Permissions in AndroidManifest.xml

Since the Mobile Payments Library is an Activity, you must declare it in the `AndroidManifest.xml` file of your application. You must also declare Internet and Phone State permissions that the Library requires. Below is an example `AndroidManifest.xml`. Sections specifically relevant to the Library are in **bold**:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.paypal.MobilePayments.Pizza"
    android:versionCode="1"
    android:versionName="1.0">
```

```

<application
    android:icon="@drawable/icon"
    android:label="@string/app_name">
        <activity
            android:name=".PizzaMain"android:
            label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER"
                />
            </intent-filter>
        </activity>
        <activity
            android:name="com.paypal.android.MEP.PayPalActivity"
            android:theme="@android:style/Theme.Translucent.NoTitleBar"
            android:configChanges="keyboardHidden|orientation/>
        </application>
    <uses-sdk android:minSdkVersion="3" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission
        android:name="android.permission.READ_PHONE_STATE"
    />
</manifest>

```

Adding the Library Jar File and Importing Classes

1. Right click on your project and select “Properties”
2. Select “Java Build Path”
3. Select the “Libraries” tab
4. Select the “Add Jars...” button
5. Choose the “PayPal_MPL.jar” file from your folder structure and click “OK”

Also, import the appropriate classes into your application classes. The following classes must be imported:

```

import com.paypal.android.MEP.CheckoutButton;
import com.paypal.android.MEP.PayPal;
import com.paypal.android.MEP.PayPalReceiverDetails;

import com.paypal.android.MEP.PayPalPayment;
or
import com.paypal.android.MEP.PayPalAdvancedPayment;

```

Required Methods in the Mobile Payments Library

initWithAppID Method

The `initWithAppID` method creates and returns the `PayPal` object. You must pass in the context and the unique application ID (`appID`) that PayPal has provided. You can choose whether to use the live or sandbox server, or use non-networked (Demo) mode (see below).

```
static public PayPal initWithAppID(Context context, String appID,
int server)
```

An example of initializing the Library with this method is:

```
PayPal ppObj = PayPal.initWithAppID(this.getBaseContext(),
"APP- 80W284485P519543T", PayPal.ENV_SANDBOX);
```

Parameter	Description
<code>context:</code>	<i>(Required)</i> The context.
<code>appID:</code>	<i>(Required)</i> PayPal Application ID from X.com. <ul style="list-style-type: none">• This will be different for each server. Thus, the <code>appID</code> will be different for Stage and Sandbox. Any <code>appID</code> value can be used when testing on „none“ since the library does not contact the server when set to this.
<code>server:</code>	<i>(Required)</i> Sets the PayPal server to Live, Sandbox, or None. Allowable values are: <ul style="list-style-type: none">• <code>ENV_LIVE</code> – Use the PayPal production servers. (does not support simulators)• <code>ENV_SANDBOX</code> – Use the PayPal testing servers.• <code>ENV_NONE</code> – Do not use any PayPal servers. Operate in demonstration mode, instead. Demonstration mode lets you view various payment flows without requiring production or test accounts on PayPal servers. Network calls within the library are simulated by using demonstration data held within the library.

NOTE: The `initWithAppID` method should only be invoked once. After initialization, reference the `PayPal` object using the `getInstance` method instead. Calling the `initWithAppID` method more than once will throw an `IllegalStateException`.

NOTE: If you do not set the optional parameter `forEnvironment`, the library defaults to use the PayPal production the servers. When testing your application, PayPal recommends that you initialize the library to use the PayPal test servers, instead.

NOTE: The Mobile Payments Library binds specific devices to specific application IDs, for enhanced security. For each of your application IDs, you must use a different sandbox

account for each of your devices. If you try to login with a different account on a device after binding, you will get the following error: “This app is attached to another PayPal account. To remove it, the account holder must visit PayPal.com and select Mobile Applications from the profile.”

To switch a device or simulator to use a different sandbox account, go to the PayPal Sandbox website on your computer, login with the account that was used on the device, select Profile > Mobile Applications, and then unbind the device from the application ID.

getCheckoutButton Method

You must get the **Pay with PayPal** payment button from the Mobile Payments Library. Use this method, which returns a `CheckoutButton` (a subclass of `LinearLayout`), which you can place in your application.

```
public CheckoutButton getCheckoutButton(Context context, int style,
int textStyle);
```

Example code of getting the Payment button from the Library is:

```
CheckoutButton launchPayPalButton =
ppObj.getCheckoutButton(this, PayPal.BUTTON_278x43,
CheckoutButton.TEXT_PAY);

RelativeLayout.LayoutParams params = new
RelativeLayout.LayoutParams (LayoutParams.WRAP_CONTENT,
LayoutParams.WRAP_CONTENT);

params.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);

params.bottomMargin = 10;

launchPayPalButton.setLayoutParams(params);
launchPayPalButton.setOnClickListener(this);

((RelativeLayout) findViewById(R.id.RelativeLayout01)).addView(lau
nchPayPalButton);
```

Parameter	Description
style:	<p>(Required) Size and appearance of the Pay with PayPal button</p> <p>Allowable values are:</p> <ul style="list-style-type: none"> • PayPal.BUTTON_152x33 • PayPal.BUTTON_194x37 • PayPal.BUTTON_278x43 • PayPal.BUTTON_294x45 <p>For images of the different button types, see Enumerated Values in the Mobile Payments Library.</p>
context:	(Required) The context
textType:	<p>(Required) The type of button to be used. The type will determine the text that is to be used on the button. This has no bearing on the payment and only affects the button itself. Allowable values are:</p> <ul style="list-style-type: none"> • CheckoutButton.TEXT_PAY • CheckoutButton.TEXT_DONATE

Start the Library Activity

The Library uses the native Android Activity mechanism to start the checkout flow, and to communicate completion back to you. In addition to the `onActivityResult` callback, you can implement `PayPalResultDelegate` to be informed immediately upon successful completion of a payment.

To start the PayPal payment, you must start the Library activity, using the Android method `startActivityForResult`. Do this when buyers touch the **Pay with PayPal** button (which you placed on your page with the `getCheckoutButton` method)

You must first create the PayPal intent and give it the Payment object. There are two types of payment objects. `PayPalPayment` handles *simple* payments, which support single receivers of payments with one transaction and a few details. `PayPalAdvancedPayment` handles *parallel* and *chained* payments, which support multiple receivers of payment with one transaction and with additional details, such as invoice data.

In the following example, the buyer checks out with a simple payment for a single recipient:

```
PayPalPayment newPayment = new
PayPalPayment();
newPayment.setSubtotal(10.f);
newPayment.setCurrency("USD");
newPayment.setRecipient("my@email.com");
newPayment.setMerchantName("My Company");

Intent paypalIntent = PayPal.getInstance().checkout(newPayment, this);
this.startActivityForResult(paypalIntent, 1);
```

In the following example, the buyer checks out with a parallel or chained payment for multiple recipients:

```
PayPalReceiverDetails receiver0,  
receiver1; receiver0 = new  
PayPalReceiverDetails();  
...  
//setup receiver details  
...  
PayPalAdvancedPayment advPayment = new  
PayPalAdvancedPayment(); advPayment.setCurrency("USD");  
advPayment.getReceivers().add(receiver0);  
advPayment.getReceivers().add(receiver1);  
  
Intent paypalIntent = PayPal.getInstance().checkout(advPayment, this);  
this.startActivityForResult(paypalIntent, 1);
```

For more information on receivers and how to use them, see [Custom Objects in the Mobile Payments Library](#).

Parameter	Description
intent:	<i>(Required)</i> A <code>PayPalPayment</code> object that contains information about the payment. Create this intent with the following code: <pre>Intent checkoutIntent = new Intent(this, PayPalActivity.class);</pre> For the properties of this object type, see “ PayPalPayment ” on page 24.
requestCode:	<i>(Required)</i> Use any integer for the request code. When you get the <code>onActivityResult()</code> callback from the Library Activity, you can use this request code to know that the results are coming from the Library activity.

Note: On older devices and/or firmware, starting the activity may take significant time to complete, resulting in an “Activity is not responding” popup if the user attempts to enter input during this time. It is recommended to create a loading thread with an animation of some sort if you witness this regularly.

Dynamic Amount Calculation

The Mobile Payments Library allows you to modify the payment based on the buyer’s shipping address. For instance, you might want to recalculate the tax amount based on the buyer’s location.

To enable this, use the optional method `enableDynamicAmountCalculation()` (see “[Optional Methods](#)” below). You must provide the logic that creates the new payment values based on the buyer’s address. The library includes a `PaymentAdjuster` class for this.

To use this feature, one of your classes (*not an Activity*) must implement “`PaymentAdjuster`”, as well as implement “`Serializable`”. For simplicity, we recommend creating a new class that does this. This class must include the following methods:

```
public MEPAmounts adjustAmount(MEPAddress address, String
currency, String amount, String tax, String shipping);
```

Parameter	Description
address	The buyer's address that should be used when calculating adjusted tax and shipping amounts.
currency	The currency of the payment.
amount	The current subtotal amount.
tax	The current tax amount.
shipping	The current shipping amount.

Your method must return a new `MEPAmounts` object (see [Custom Objects](#)). This object contains the new currency and amounts.

```
public Vector<MEPReceiverAmounts>
adjustAmountsAdvanced(MEPAddress address, String currency,
Vector<MEPReceiverAmounts> receivers);
```

Parameter	Description
address	The buyer's address that should be used when calculating adjusted tax and shipping amounts.
currency	The currency of the payment.
receivers	A collection of current receivers and the amounts associated with each receiver.

Your method must return a new `Vector<MEPReceiverAmounts>` to update the library with adjusted amounts for each receiver. (See [Custom Objects in the Mobile Payments Library](#))

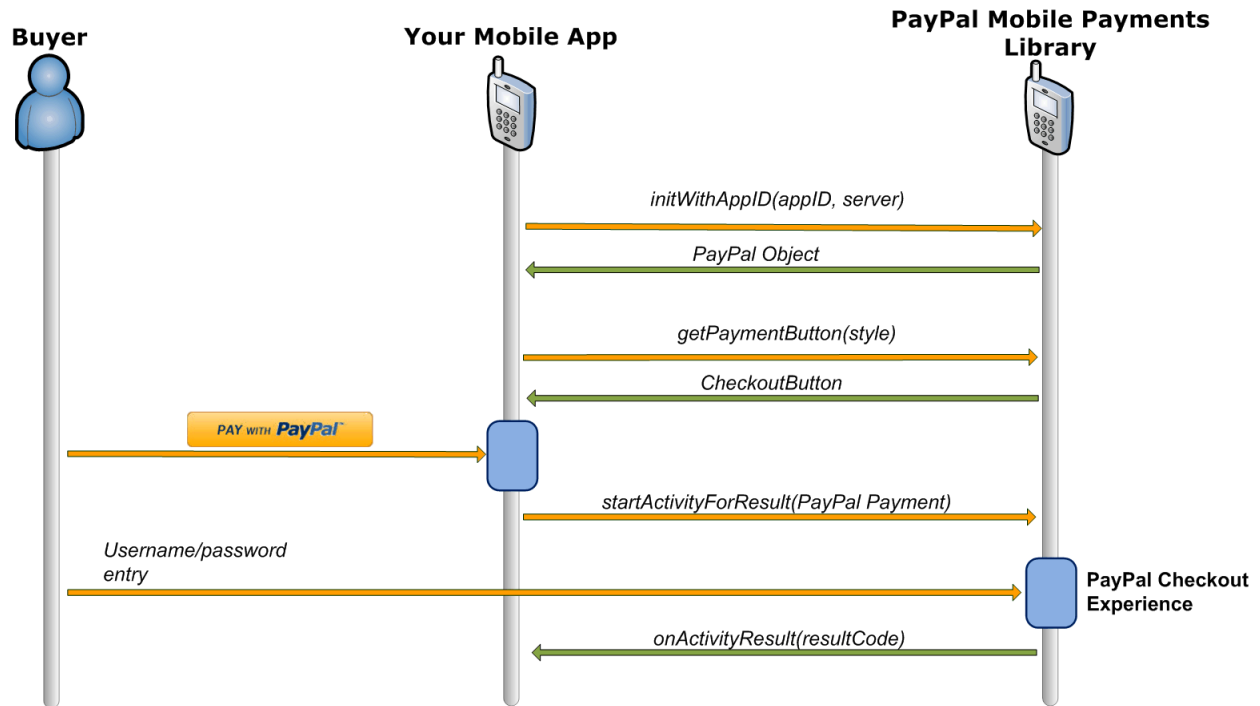
Applications may cancel a payment during the `PaymentAdjust` callback by returning 'null'. Returning null will cancel the entire payment. A dialog will be presented to the user indicating that the adjustment failed and the payment is being cancelled. Applications may also call `PayPal.setAdjustPaymentError(String message)` to establish the message that will be presented. Otherwise a standard message will be used. The error message should be established prior to failing the payment adjust callback.

When you create the `PayPal` Activity, you must pass through the `PaymentAdjuster` class to another form of the checkout method. For example, if you created an “AdjustAmounts” class that implements `PaymentAdjuster` and contains the `adjustAmount` method, your code could be:

```
AdjustAmounts adjustClass = new AdjustAmounts();
Intent paypalIntent = PayPal.getInstance().checkout([your
payment object], this, adjustClass);
this.startActivityForResult(paypalIntent, 1);
```

Method Sequence

The following diagram illustrates the sequence of methods required to implement the checkout experience.



Optional Methods in the Mobile Payments Library

getInstance Method

This method returns the singleton PayPal object.

```
PayPal paypal = PayPal.getInstance();
```

setLanguage Method

```
paypal.setLanguage(String emailOrPhone);
```

Enable / Disable Shipping Method

This method lets buyers include display of shipping addresses in the Library. With shipping enabled, buyers can choose an address from the list available in their PayPal accounts. The chosen shipping address is then used for the payment. Shipping is enabled by default.

```
paypal.setShippingEnabled(boolean isEnabled);
```

setFeesPayer Method

This method is valid only for Personal payments only. Call this method to set who pays fees by default. If you do not call this method, the receiver pays any fees by default.

```
paypal.setFeesPayer (int feesPayer);
```

setDynamicAmountCalculationEnabled Method

This method lets you recalculate the payment amount, tax, currency, and shipping values based on the shipping address chosen by a buyer. If you use this method to enable dynamic amount calculation before the checkout starts, the library will dynamically update the payment based on logic you provide (see above).

NOTE: If shipping is not enabled, this method is ignored.

```
paypal.setDynamicAmountCalculationEnabled(boolean enabled);
```

Activity Results for the Mobile Payments Library

You will receive the payment results through the `onActivityResult` method. This method must be overridden in the same class in which you called `startActivityForResult`. There are three possible results. The first two are native Android Activity results. The third result is a custom result for the Library.

In addition to these activity results, a successful payment callback is provided to note when the payment is successfully completed. These Activity results are only posted when the library Activity has terminated and control of the UI has passed back to the host Application:

Activity.RESULT_OK

This result is returned when a PayPal payment succeeds. The pay key is available in the Intent data as `PayPalActivity.EXTRA_PAY_KEY`. The code to get the pay key as a string is: `data.getStringExtra(PayPalActivity.EXTRA_PAY_KEY)`.

Activity.RESULT_CANCELED

This result is returned if the buyer cancels the payment during checkout.

```
paymentCanceled();
```

PayPalActivity.RESULT_FAILURE

This result is returned if the payment fails for any reason other than a buyer who cancels the payment during checkout. The error id and the buyer-friendly error message are available from the Intent data as `PayPalActivity.EXTRA_ERROR_ID` and `PayPalActivity.EXTRA_ERROR_MESSAGE`.

Example code on handling the response is:

```
@Override
public void onActivityResult(int requestCode, int resultCode,
Intent data) {
    switch(resultCode) {
        case Activity.RESULT_OK:
            //The payment
            succeeded String
            payKey =
            data.getStringExtra(PayPalActivity.EXTRA_PAY_KEY);
            //Tell the user their payment succeeded
            break;
        case Activity.RESULT_CANCELED:
            //The payment was canceled
            //Tell the user their payment was canceled
            break;
        case PayPalActivity.RESULT_FAILURE:
            //The payment failed -- we get the error from
            the EXTRA_ERROR_ID and EXTRA_ERROR_MESSAGE
            String errorID =
            data.getStringExtra(PayPalActivity.EXTRA_ERROR_ID);
            String errorMessage =
            data.getStringExtra(PayPalActivity.EXTRA_ERROR_MESSAGE);
            //Tell the user their payment was failed.
    }
}
```

For a list of error types, see “Enumerated Values in the Mobile Payments Library”.

The interface for providing details on when a payment is completed is defined in `com.paypal.android.MEP.PayPalResultDelegate`. This interface provides you with a way to be notified immediately when a payment has completed:

```
public interface PayPalResultDelegate{
    void onPaymentSucceeded(String payKey, String paymentStatus);
    void onPaymentFailed(String paymentStatus, String
        correlationID, String payKey, String errorID, String
        errorMessage);
    void onPaymentCanceled(String paymentStatus);
}
```

PayPal recommends that you implement this interface to be immediately informed upon the completion of a payment, rather than waiting to receive the transaction details upon the completion of the PayPal MPL Activity.

After the Payment

After the payment is completed, the Mobile Payments Library will return the `payKey`. There are also a number of other features available to you to further deal with the payment: Instant Payment Notification, Transaction Details, and Refunds.

Instant Payment Notification

Instant Payment Notification (IPN) is PayPal's message service that sends a notification when a transaction is affected. You can integrate IPN with your systems to automate and manage your back office. More details and documentation are available at:

www.paypal.com/ipn

Transaction Details

You can integrate with the PayPal `PaymentDetails` API to retrieve details on a payment based on the `payKey`. More details and documentation are available at:

<https://developer.paypal.com/docs/classic/adaptive-payments/integration-guide/APIIntro/>

Refunds

Refunds can be supported by manual refund via the PayPal account interface or via the Refund API. More details and documentation are available at:

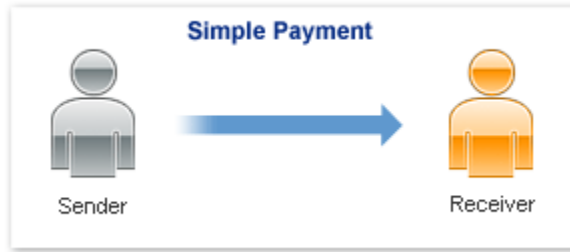
<https://developer.paypal.com/docs/classic/adaptive-payments/integration-guide/APIIntro/>

Simple, Parallel, and Chained Payments

Simple payments have a single recipient. Parallel and Chained payments have multiple recipients and differ in the how the payments are split.

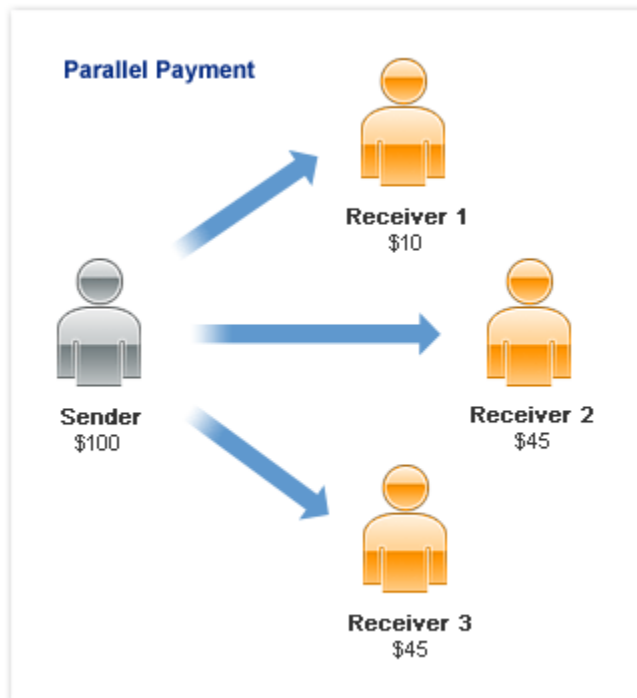
Simple Payments

Simple payments use the `PayPalPayment` object, which only supports a payment to a single recipient.



Parallel Payments

Parallel Payments allow you to make payments for any amount to any number of recipients. A parallel payment is a payment made to multiple recipients with no primary recipient. From the end-user's standpoint, a parallel payment will affect the checkout experience by showing the details for each recipient. Contrary to chained payments, the recipients of a parallel payment are not linked together in terms of amount.

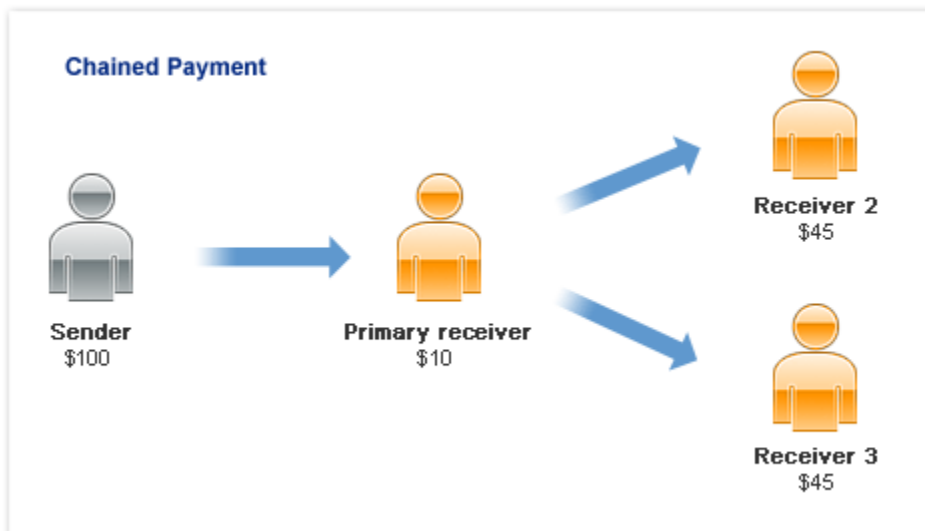


Chained Payments

A chained payment is a payment from a sender that is indirectly parallel among multiple receivers. It is an extension of a typical payment from a sender to a receiver; however, a receiver, known as the *primary receiver*, passes part of the payment to other receivers, who are called *secondary receivers*.

NOTE: Chained payments require a specific permission level on the part of the API caller and merchant. For information, refer to the section "[Adaptive Payments Service Permissions](#)" in the Adaptive Payments integration guide.

You can have at most one primary receiver and from 1 to 5 secondary receivers. Chained payments are useful in cases when the primary receiver acts as an agent for other receivers. The sender deals only with the primary receiver and does not know about the secondary receivers, including how a payment is parallel among receivers. The following example shows a sender making a payment of \$100:



In this example, the primary receiver receives \$100 from the sender's perspective; however, the primary receiver actually receives only \$10 and passes a total of \$90 to secondary receivers Receiver 2 and Receiver 3.

NOTE: The scenario above is an example only and does not take PayPal fees into account.

Preapprovals

The PayPal Mobile Payments Library lets you set up obtain authorization in advance from buyers for future payments to you without requiring buyers to authorize each payment individually. For example, you might use the library to establish preapproval agreements for subscriptions to mobile content, such as mobile streaming audio or video. Or, you might use the library to establish preapproval agreements for payments to gain access to higher levels of difficulty in mobile games.

How Preapprovals Work

There are 3 steps to setting up and using preapprovals.

1. Obtain a *pending preapproval key* from PayPal.

From your web server, send a `Preapproval` request to PayPal with the terms of your preapproval agreement.

2. Obtain authorization from the buyer for the preapproval agreement.

From your mobile application, start the Library activity by using the Android method `startActivityForResult` with the preapproval key and the merchant's name. The library launches the preapproval checkout experience and returns a *confirmed preapproval key*.

3. Take payments from the buyer under the terms of the preapproval agreement.

From your web server, send a `Pay` request to PayPal with the buyer's confirmed preapproval key.

For more information about the `Preapproval` and `Pay` requests, see the [Adaptive Payments integration guide](#).

About Preapproval Keys

Preapproval keys uniquely identify preapproval your agreements. Preapproval keys that you obtain by using the `Preapproval` API identify your pending preapproval agreements. No buyers have yet agreed to them. Pending approval keys remain valid for 3 hours before expiring without confirmation from buyers.

Call the `preapprovalWithKey` method to launch the preapproval checkout experience to confirm a buyer's agreement to a pending preapproval. If the buyer completes the preapproval checkout, the library returns a confirmed preapproval key. Maintain a record of buyers and their confirmed preapproval keys on your web server. Later on your web server, take payments from buyers by sending `Pay` requests with buyers' preapproval keys to PayPal.

About Preapproval Pins

Confirmed preapproval keys let you take payments from buyers without requiring them to log in to PayPal to authorize payments individually. Depending on your business model, you may want to obtain consent quickly from buyers before you take individual payments. Preapproval PINs are special codes that buyers enter to authorize preapproved payments individually without logging in to PayPal.

For example, you might have a mobile game that requires payment from buyers to enter a higher level of difficulty. You could take the payment, without notice, when the buyer enters the higher level. However, the buyer might dispute the payment later, despite the preapproval agreement and the automatic payment notice from PayPal. Obtain a buyer's consent before you take the entrance fee to help improve the buying experience.

Specify that you want your preapprovals to use preapproval PINs when you send `Preapproval` requests from your web server to PayPal. Set the `PreapprovalRequest.pinType` to `REQUIRED`. PayPal returns preapproval keys that require buyers to create preapproval PINs during preapproval checkout.

Later, when you take payments by using a buyer's confirmed preapproval key, prompt the buyer for the preapproval PIN. Pass the buyer's PIN to PayPal when you send the `Pay` request from your web server. PayPal recommends that you display the payment reason and payment amount when you prompt buyers for their preapproval PINs.

Sample Call

After you obtain a pending approval key, construct a `PayPalPreapproval` object that includes the key and the merchant's name. Then, use the `preapprove()` method to create an `Intent` and pass it in the Android `startActivityForResult()` method.

```
PayPalPreapproval preapproval = new
PayPalPreapproval();
preapproval.setCurrencyType("USD");
preapproval.setMerchantName("Preapproval Merchant");

Intent preapproveIntent =
PayPal.getInstance().preapprove(preapproval, this);

startActivityForResult(preapproveIntent, request);
```

NOTE: See [Activity Results for the Mobile Library](#) for callback details.

Custom Objects in the Mobile Payments Library

The Mobile Payments Library includes custom objects for passing information between the library and your application during checkout. Use these objects if you enable dynamic amount calculation by calling the `DynamicAmountUpdate` method.

MEPAddress

This object is passed to your `PaymentAdjuster` class in the `AdjustAmounts` method. Use this address to update the payment amount, tax, currency, and shipping values of the payment.

Then, the buyer continues to check out with the new amounts.

Method	Description
<code>String getStreet1()</code>	First line of the street address.
<code>String getStreet2()</code>	Second line of the street address.
<code>String getCity()</code>	Name of the city.
<code>String getState()</code>	Name of the state or province.
<code>String getPostalcode()</code>	U.S. ZIP code or other country-specific postal code.
<code>String getCountrycode()</code>	The 2-character country code.
<code>String getCountry()</code>	The name of the country.

MEPAmounts

This object is returned to the Library by the `AdjustAmounts` method of your `PaymentAdjuster` class. This object contains the values for the updated payment.

Property	Description
<code>setCurrency(String currency)</code>	<i>(Optional)</i> Currency code of the amount. Defaults to USD.
<code>setPaymentAmount(String paymentAmount)</code> or <code>setPaymentAmount(BigDecimal paymentAmount)</code>	<i>(Required)</i> Amount of the payment before tax or shipping.
<code>setTax(String tax)</code> or <code>setTax(BigDecimal tax)</code>	<i>(Optional)</i> Tax amount associated with the item.
<code>setShipping(String shipping)</code> or <code>setShipping(BigDecimal shipping)</code>	<i>(Optional)</i> Shipping amount for the item

MEPReceiverAmounts

Similarly to `MEPAmounts` and `MEPAddress`, this object is used in dynamic amount calculation for advanced payments.

Property	Description
<code>public MEPAmounts amounts;</code>	Amounts for this receiver.
<code>public String receiver;</code>	The receiver for this amount calculation.

PayPalPayment

This object is passed to the library through extra data in the Intent when the Library Activity is started. This object contains all the values for a payment

Method	Description
<code>setCurrencyType (String currency)</code>	<i>(Required)</i> - Currency code for the payment. Defaults to USD if not set.
<code>setSubtotal (BigDecimal amount)</code>	<i>(Required)</i> – Subtotal for the payment.
<code>setRecipient (String recipient)</code>	<i>(Required)</i> - The email address or phone number of the payment's recipient.
<code>setMerchantName (String name)</code>	<i>(Optional)</i> - Displayed at the top of the library screen. If not set, displays the recipient email or phone number string instead.
<code>setInvoiceData (PayPalInvoiceData)</code>	<i>(Optional)</i> – The <code>InvoiceData</code> holds information such as tax and shipping amounts as well as an optional breakdown of the items included.
<code>setCustomId (String customId)</code>	<i>(Optional)</i> – The merchant's custom ID.
<code>setIpnUrl (String IPNURL)</code>	<i>(Optional)</i> – The IPN URL.
<code>setDescription (String description)</code>	<i>(Optional)</i> - Description of the item. Defaults to “Item” if not set.
<code>setMemo (String memo)</code>	<i>(Optional)</i> – Note for the payment.

PayPalAdvancedPayment

This object is passed to the library through extra data in the Intent when the Library Activity is started. This object contains all the values for a payment

Method	Description
<code>setCurrencyType (String currency)</code>	<i>(Required)</i> - Currency code for the payment. Defaults to USD if not set.
<code>setReceivers (ArrayList<PayPalReceiverDetails> receivers)</code> or <code>getReceivers ()</code>	<i>(Required)</i> – Your payment must have at minimum one receiver. The receiver itself should be set up before adding it to the payment. See Receiver below for more details. The <code>getReceivers ()</code> call can be used to alter the current <code>ArrayList</code> .
<code>setIpnUrl (String IPNRUL)</code>	<i>(Optional)</i> – The IPN URL
<code>setMemo (String memo)</code>	<i>(Optional)</i> – Note for the payment.

PayPalPreapproval

This object is passed to the library through extra data in the Intent when the Library Activity is started. This object contains all the values for a payment

Method	Description
<code>setCurrencyType(String currency)</code>	<i>(Required)</i> - Currency code for the payment. Defaults to USD if not set.
<code>setType(int type)</code>	<i>(Optional)</i> – The type of button to display, either <code>TYPE_AGREE</code> or <code>TYPE_AGREE_AND_PAY</code> . Defaults to <code>TYPE_AGREE_AND_PAY</code> .
<code>setIpnUrl(String IPNRUL)</code>	<i>(Optional)</i> – The IPN URL
<code>setMemo(String memo)</code>	<i>(Optional)</i> – Note for the payment.

PayPalInvoiceData

This is an optional object to be set for any receiver. The `PayPalPayment` (simple payment) has only one receiver and the `InvoiceData` is added directly to the payment object itself. The `PayPalAdvancedPayment` can support multiple receivers, each of which can have an invoice data added to it.

While `InvoiceData` is an optional parameter for any given receiver, once added the `InvoiceData` must be populated with certain required parameters (see below).

Method	Description
<code>setTax(BigDecimal shipping)</code>	<i>(Required)</i> – Tax amount to be used for the payment. This can be updated after the checkout flow has been started if dynamic amount calculation is enabled. The amount can be 0.
<code>setShipping(BigDecimal shipping)</code>	<i>(Required)</i> – Shipping amount to be used for the payment. This can be updated after the checkout flow has been started if dynamic amount calculation is enabled. The amount can be 0.
<code>setInvoiceItems(ArrayList<PayPalInvoiceItem> items)</code>	<i>(Required)</i> – Sets the list of items in the invoice. See <code>InvoiceItem</code> below for more details. These items do not affect the total amount of the payment but must equal the subtotal.

PayPalInvoiceItem

These items can be added to any InvoiceData.

NOTE: The `UnitPrice` and `Quantity` must multiply together correctly to equal the `TotalPrice`. The `TotalPrices` of all `invoiceItems` of a `PayPalPayment` or a `PayPalAdvancedPayment` must equal the subtotal of the payment.

Method	Description
<code>setName(String name)</code>	<i>(Optional)</i> – The name of the item.
<code>setID(String ID)</code>	<i>(Optional)</i> – A unique ID for the item.
<code>setTotalPrice(BigDecimal price)</code>	<i>(Required)</i> – The total cost of this item (unit cost * quantity).
<code>setUnitPrice(BigDecimal price)</code>	<i>(Required)</i> – The cost of a single unit of this item.
<code>setQuantity(int qty)</code>	<i>(Required)</i> – The quantity of this item.

PayPalReceiverDetails

This object is used in the `PayPalAdvancePayment` to identify a single recipient.

Method	Description
<code>setRecipient(String recipient)</code>	<i>(Required)</i> – The email address or phone number of this recipient.
<code>setSubtotal(BigDecimal amount)</code>	<i>(Required)</i> – The subtotal of the payment to this recipient.
<code>setIsPrimary(boolean isPrimary)</code>	<i>(Optional)</i> – Whether or not the receiver is a the primary receiver of the payment. A payment with multiple recipients may specify the primary recipient of the payment to denote a chained payment. Payments with multiple recipients but no primary receivers are parallel payments. See “ Simple, Parallel, and Chained Payments ” for more details.
<code>setPaymentType(int paymentType)</code>	<i>(Optional)</i> – If this value is not set, the value passed into <code>getPaymentButton()</code> will be used instead.
<code>setPaymentSubType(int paymentSubtype)</code>	<i>(Optional)</i> – If this value is not set, the value passed into <code>getPaymentButton()</code> will be used instead.
<code>setInvoiceData(PayPalInvoiceData invoice)</code>	<i>(Optional)</i> – The <code>InvoiceData</code> holds information such as tax and shipping amounts as well as an optional breakdown of the items included. See <code>InvoiceData</code> above for more details.
<code>setCustomID(String ID)</code>	<i>(Optional)</i> – The recipient's custom ID.
<code>setMerchantName(String name)</code>	<i>(Optional)</i> - Displayed at the top of the library screen. If not set, displays the recipient email or phone number string instead.

Enumerated Values in the Mobile Payments Library

The enumerated values supported by various methods in the library are:

PAYPAL_ENVIRONMENT

- `ENV_LIVE`: Use the PayPal production servers
- `ENV_SANDBOX`: Use the PayPal testing servers
- `ENV_NONE`: Do not use any PayPal servers. Operate in demonstration mode, instead. Demonstration mode lets you view various payment flows without requiring production or test accounts on PayPal servers. Network calls within the library are simulated by using demonstration data held within the library.

NOTE: `ENV_LIVE` does not support simulators.

PAYPAL_BUTTON_TYPE

`BUTTON_152x33`



`BUTTON_194x37`



`BUTTON_278x43`



`BUTTON_294x45`



NOTE: If the `buttonTextType` parameter is set to `TEXT_DONATE` the word “Pay” in the above buttons is replaced by “Donate.” The language of the button will also change based on the language you pass into the `setLang` method or the auto detected language on the phone.

PayPalPaymentType

`PAYMENT_TYPE_HARD_GOODS`
`PAYMENT_TYPE_SERVICE`
`PAYMENT_TYPE_PERSONAL`

NOTE: For `Personal` payment types, the PayPal Checkout experience differs slightly from other payment types. Additionally for `Personal` payment types, senders in some cases can choose who pays any fees – the sender or the recipient. In India and Germany, recipients always pay any fees.

For more information, see the `setFeesPayer` method in [Optional Methods in the Mobile Payments Library](#).

PAYPAL_FAILURE

```
SYSTEM_ERROR  
RECIPIENT_ERROR  
APPLICATION_ERROR  
CONSUMER_ERROR
```

These failures are returned by the `PayPalActivity.RESULT_FAILURE` result from the PayPal Library activity.

Localization Support in the Mobile Payments Library

To change the language in the Library, call `setLanguage(String localeCode)` on the PayPal object. Pass through the desired locale code (e.g. for US English, use `en_US`). If you do not use this method, the Library defaults to the locale to which the device is set.

The Library supports the following regions:

Language (Country or Region)	Locale Code
Chinese (Hong Kong)	zh_HK
Chinese (Taiwan)	zh_TW
Dutch (Belgium)	nl_BE
Dutch (Netherlands)	nl_NL
English (Australia)	en_AU
English (Belgium)	en_BE
English (Canada)	en_CA
English (France)	en_FR
English (Germany)	en_DE
English (Great Britain)	en_GB
English (Hong Kong)	en_HK
English (India)	en_IN
English (Japan)	en_JP
English (Mexico)	en_MX
English (Netherlands)	en_NL
English (Poland)	en_PL
English (Singapore)	en_SG

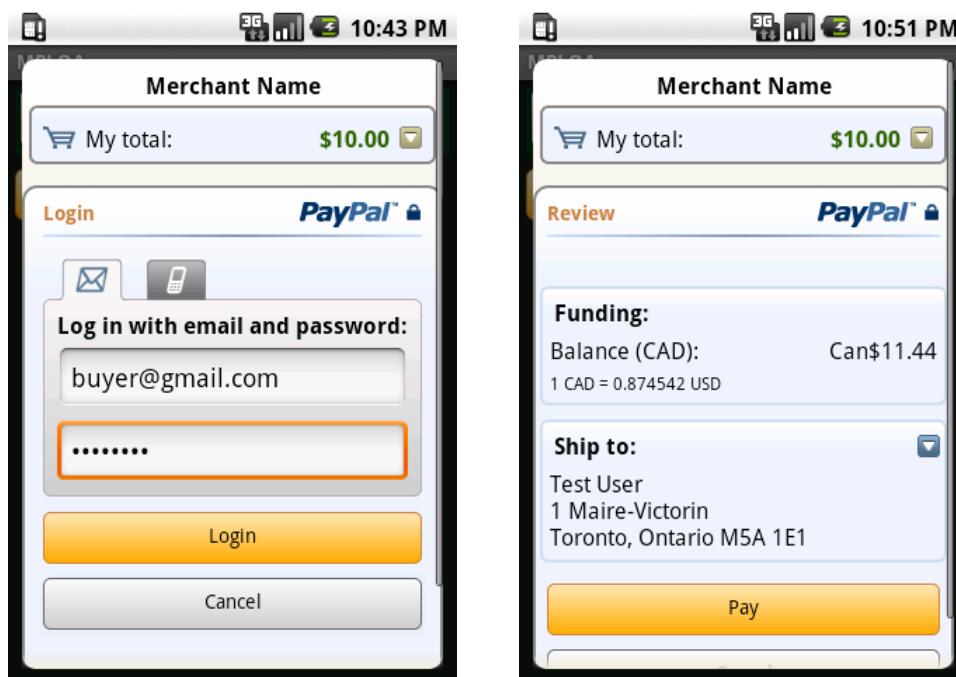
Language (Country or Region)	Locale Code
English (Spain)	en_ES
English (Switzerland)	en_CH
English (Taiwan)	en_TW
English (United States)	en_US
French (Belgium)	fr_BE
French (Canada)	fr_CA
French (France)	fr_FR
French (Switzerland)	fr_CH
German (Austria)	de_AT
German (Germany)	de_DE
German (Switzerland)	de_CH
Italian (Italy)	it_IT
Japanese (Japan)	ja_JP
Spanish (Argentina)	es_AR
Spanish (Mexico)	es_MX
Spanish (Spain)	es_ES
Polish (Poland)	pl_PL
Portuguese (Brazil)	pt_BR

2. The Checkout Experience with the Mobile Payments Library

The following screen shots illustrate several different PayPal Checkout experiences that occur after buyers click the PayPal button that your application obtains from the library by using the `getPayPalButton()` method.

Checkout Experience #1 – Goods or Services with Shipping

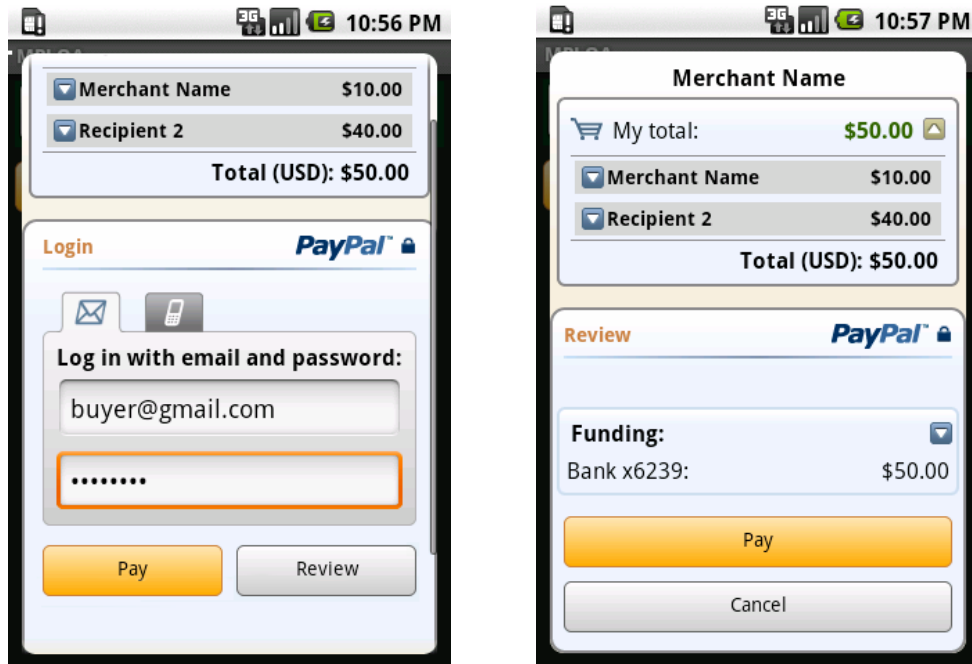
Payment type=Hard Goods or Services / Shipping=enabled



In the preceding experience, buyers enter their PayPal login credentials in the Log In To PayPal screen. Then, they can review details of the payment in the second screen and change funding source or shipping address. If satisfied, buyers click **Pay** to complete the payment.

Checkout Experience #2 – Goods or Services without Shipping

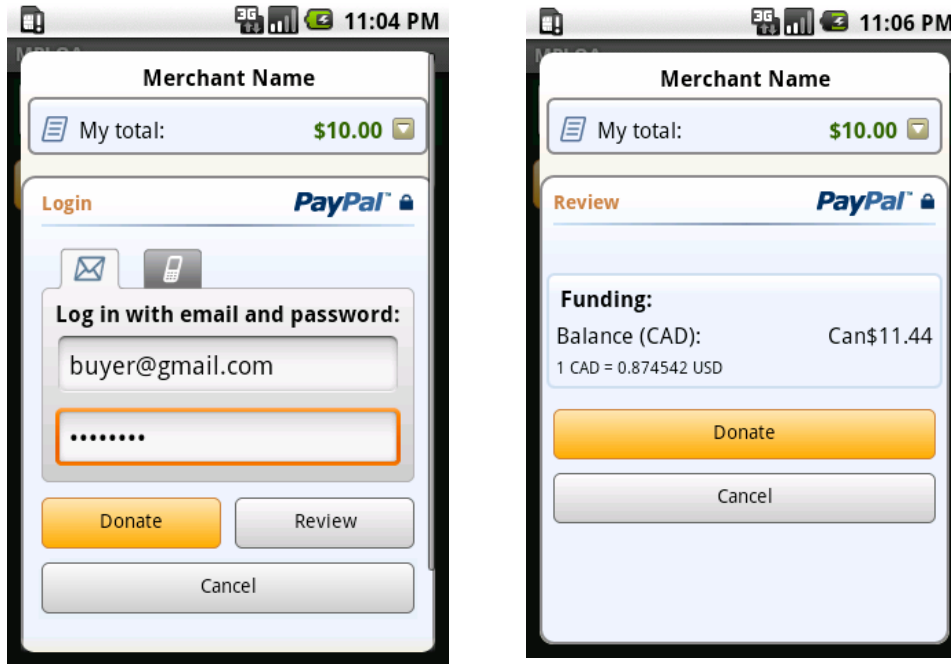
Payment type=Hard Goods or Services / Shipping=disabled



In the case, shipping is not required (i.e. manual pick up of goods or services). Shipping is disabled by a call to the `setShippingEnabled()` library method. Buyers enter their PayPal login credentials and pay directly by clicking the **Pay** button on the first screen. Buyers can review funding choices by clicking the **Review** button on the same page.

Checkout Experience #3 – Donations

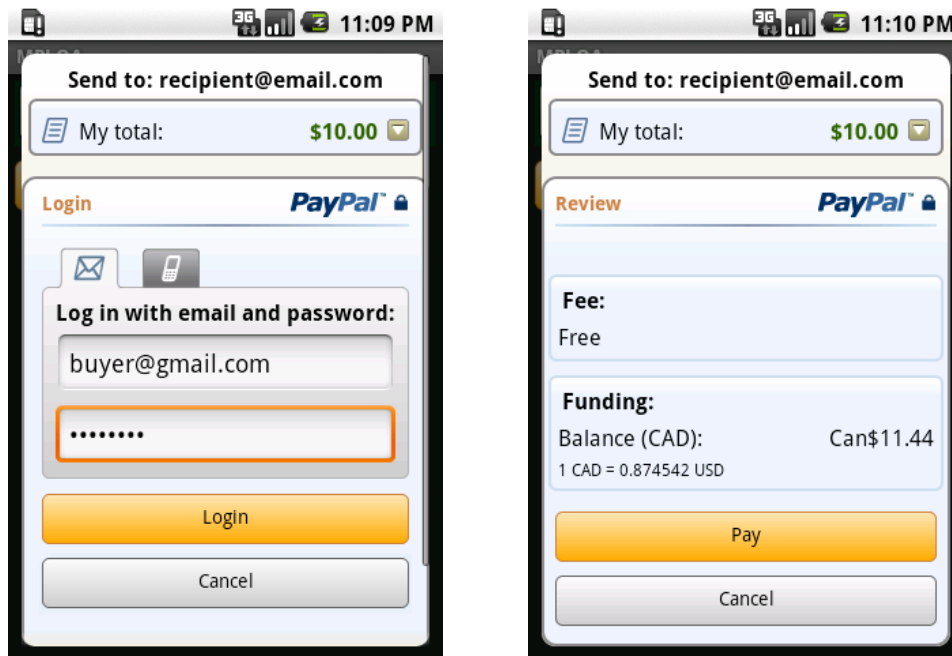
Payment type=Donations / Shipping=enabled



In the preceding experience, buyers make a donation to a charity or other cause. In this context the charity or cause wants to leverage PayPal members' addresses as mailing addresses for donation receipts. By enabling shipping in the library, buyers are presented with their primary mailing address can choose another mailing address from the ones in their PayPal accounts.

Checkout Experience #4 – Personal Send Money Payments

Payment type=Personal payments / Shipping=disabled



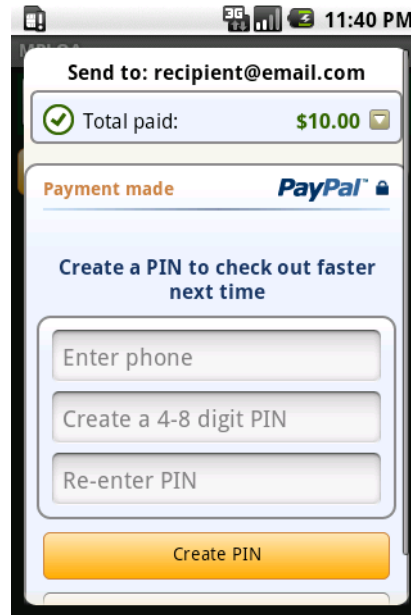
In the preceding experience, PayPal members make personal payments to other PayPal members. There are no transaction fees when it is funded by PayPal balance or by a bank account on file.

The transaction carries a fee when it is funded by payment card – debit or credit or PayPal Credit cards. In some cases, senders choose who pays any fees – sender or recipient. In India and Germany, recipients always pay any fees.

For more information on PayPal Send Money and pricing, refer to:

<https://www.paypal.com/us/webapps/mpp/send-money-online>

Checkout Experience #5 – Create Pin



In the preceding experience, a PayPal member has just completed a payment and does not currently have a PIN associated with his/her account. By following the on-screen instructions, the user can associate his/her account with a phone number and PIN for easier login in the future.

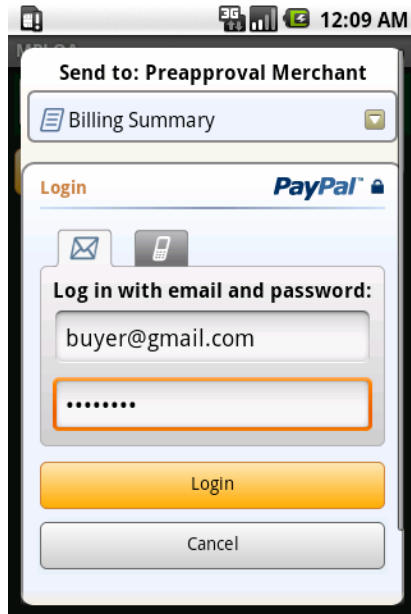
Upon successful creation of the PIN, the user will be returned to your application triggering the `paymentSuccess()` delegate callback.

Checkout Experience #6 – Preapproval

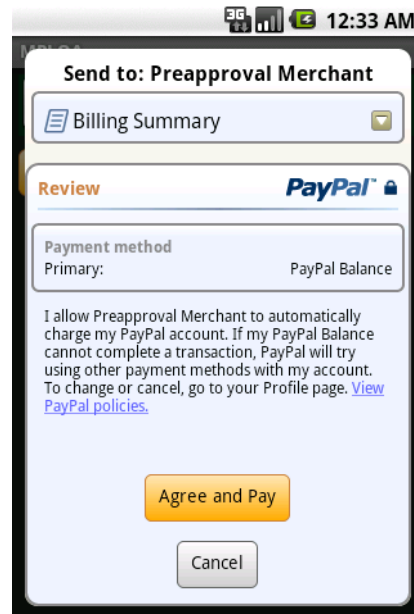
In this experience, you started the Android Activity with the preapproval intent, as discussed under [Preapprovals](#).

Basic Preapproval Checkout

Login Screen



Agree and Pay Screen



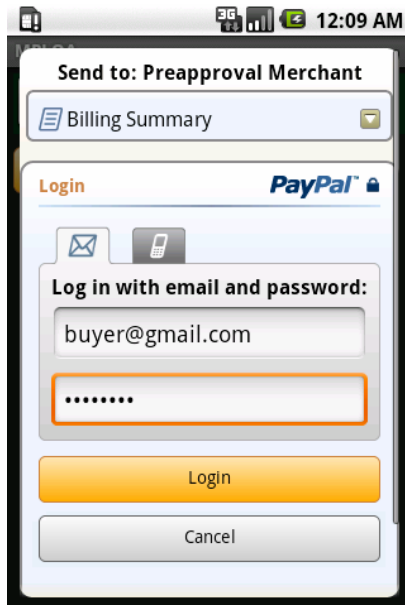
During a preapproval checkout, the buyer agrees to the terms of a preapproval agreement. The agreement authorizes you to take payments without requiring the buyer to log in to PayPal to authorize the payments individually. After the buyer completes the checkout, PayPal returns the buyer's *confirmed preapproval key* to your mobile application.

Use the buyer's confirmed preapproval key to take the preapproved payments. The library does not take the payments for you. After UI control returns to your mobile application, store the buyer's preapproval key on your web server. Then, take your first preapproved payment by sending a `Pay` request with the buyer's preapproval key from your web server to PayPal.

Creating Preapproval PINs During Preapproval Checkout

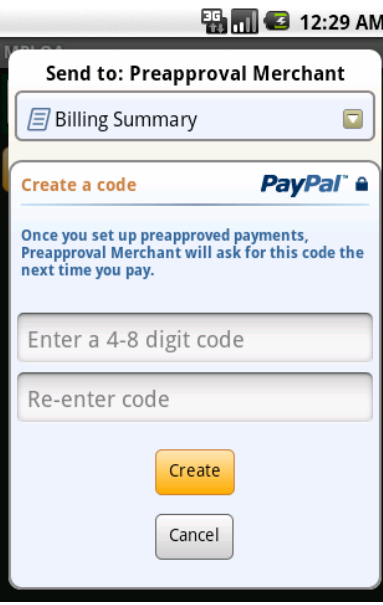
Depending on your business model, you may require buyers to create *preapproval PINs* during preapproval checkout. Preapproval PINs are special codes that buyers specify during checkout to let them consent quickly later to individual payments. If your preapproval agreements require PINs, PayPal displays the optional “Create a Code” screen during preapproval checkout.

Login Screen



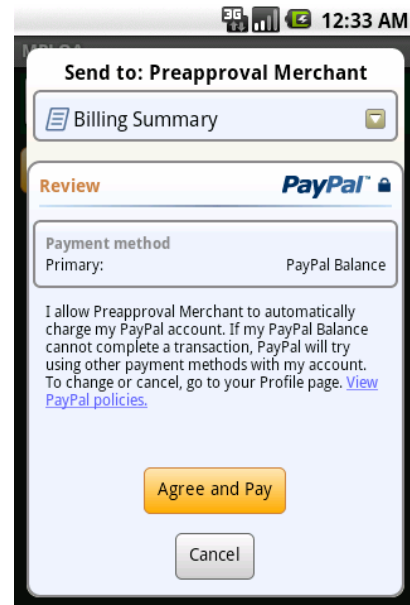
This screenshot shows the PayPal login interface. At the top, it says "Send to: Preapproval Merchant" with a "Billing Summary" link. Below this is the "Login" section with the PayPal logo. It prompts the user to "Log in with email and password:" and shows the email "buyer@gmail.com" and a masked password field. There are "Login" and "Cancel" buttons at the bottom.

Create a Code Screen



This screenshot shows the "Create a code" screen. It has the same header as the login screen. The main text says: "Once you set up preapproved payments, Preapproval Merchant will ask for this code the next time you pay." Below this are two input fields: "Enter a 4-8 digit code" and "Re-enter code". There are "Create" and "Cancel" buttons at the bottom.

Agree and Pay Screen



This screenshot shows the "Review" screen. It has the same header. It shows the "Payment method" as "Primary: PayPal Balance". Below this is a paragraph of text: "I allow Preapproval Merchant to automatically charge my PayPal account. If my PayPal Balance cannot complete a transaction, PayPal will try using other payment methods with my account. To change or cancel, go to your Profile page. [View PayPal policies.](#)" There are "Agree and Pay" and "Cancel" buttons at the bottom.

After logging in to PayPal, the buyer enters a code that only the buyer and PayPal know. Later, before you take a preapproved payment, prompt the buyer to enter the preapproval PIN. Then from your web server, include the PIN that the buyer entered with the `Pay` request that you send to PayPal. PayPal recommends that you display the payment reason and payment amount when you prompt the buyer for the preapproval PIN.

3. Submitting Your Application to PayPal

Before you submit your application to Apple and distribute your mobile application publicly, you need an authorized application ID from PayPal. PayPal tests all mobile applications before issuing application IDs. Test your mobile application thoroughly in the PayPal Sandbox by using `APP-80W284485P519543T` as your test application ID. Then, submit your test application to PayPal.

1. Log in or sign up on [PayPal's developer website](#).
2. On the Dashboard, navigate to [My Apps & Credentials](#).
3. Scroll to NVP/SOAP API apps and click [Manage NVP/SOAP API apps](#).
4. Click **New App** and complete the form.
5. Click **Submit App**.

Result:

For those using simple or parallel payments, PayPal reviews your application within 24 hours and responds by sending you your `PayPalApplicationID`. Reviewers at PayPal follow up by email with questions, should they arise, before they approve your mobile application. For those using chained payments or preapprovals, the review may take longer.

After completing this task:

Wait until PayPal sends you your application ID. Then, make sure that you update your software with the following changes before you submit your mobile application to Apple:

- **Application ID:** in your calls to `initWithApplicationId`
- **Environment:** in your calls to `initWithApplicationId`
- **Recipient:** in the `PayPalPayment` object

A. Currencies Supported by PayPal

PayPal uses 3-character ISO-4217 codes for specifying currencies in fields and variables. See the [currency codes reference page](#) for more information.

B. Countries and Regions Supported by PayPal

PayPal uses 2-character ISO-3166-1 codes for specifying countries and regions that are supported in fields and variables. See the [country codes reference page](#) for more information.