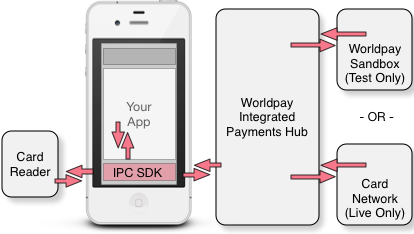
# Introduction

The Worldpay Total Integrated Payment Client Software Development Kit (IPC SDK) is a wonderful way for you to develop rich mobile applications in the native language of the mobile device. We currently have SDKs for iOS, Android, and Windows. Besides allowing you to develop native applications, there are two other compelling reasons to choose this platform:

* We do all of the device management of your card reader. No need to worry about device drivers. No Bluetooth management functions. Nothing. We take care of those and report back to you when something of interest happens.
* You do not have to undergo EMV certification testing, because we protect you from the card data. This would otherwise be a costly and time-consuming activity.

The SDKs provide you with the glue necessary to talk with a card reader (if you use one) and the code to make the remote calls to the Worldpay Integrated Payments Hub.



Note that while one of the advantages to using the IPC SDK is the management of the card reader, you can still use this library for card not present applications or any other app that wants access to our vault, tokenization, and customer management features. Think you might one day move from card not present to card present? Develop against the app today and rest comfortable that your app will be future-proofed when you are ready to make the change – same code base.

The IPC SDK installs alongside your software - adding transaction processing to your applications. It facilitates all transactional communication between the Worldpay Integrated Payments Hub (IPH) and approved hardware devices to isolate payment data and keeps it separate from your software application.

# Requirements

|  |  |
| --- | --- |
| Platform Android 4.4 and later | Supported Devices Miura M010 Shuttle |

# Overview of the Software Development Kit API

Your starting point for all of the operations available in the IPC SDK is the WorldPayAPI object. This object is a singleton accessible by the instance method on the WorldPayAPI class.

## Securely Acquiring Card Information

The IPC SDK manages all interaction with the attached card reader, and then uses the card account information to perform transactions through the Worldpay Total Integrated Payment Hub. The IPC SDK also provides a way for you to manually enter card information for those times when the card is not present for the transaction.

The IPC SDK supports the following methods to capture card information:

|  |  |
| --- | --- |
| **Card Information Capture Methods** | |
|  |  |
| Magnetic Stripe Reader | Captures the card information from the magnetic stripe reader on the attached device. |
| EMV Contact | Obtains card information from a chip card inserted in the EMV reader of the attached device. |
| Manual Entry | Presents a form where the customer can manually enter the card number, expiration date and postal code, and then calls the Integrated Payment Hub to perform the specified payment transaction. Returns the result of the payment transaction to the associated delegate. |

## Making Payments

The following table shows the transactions you can use to manage payments. The IPC SDK takes care of the network interactions with the Integrated Payment Hub.

| **Payment Transactions** | |
| --- | --- |
| Authorize | This call authorizes a transaction but does not capture the transaction for settlement. In a card-present environment, this option is most commonly used in service industry transactions where an initial amount is authorized prior to a gratuity being added. If the Authorization Only call is used, a subsequent Prior Auth Capture call must be made to capture the transaction for settlement. |
| Prior Auth Capture | This call allows a previously authorized transaction to be captured for settlement. In a card-present environment, Prior Auth Capture is commonly used when the transaction amount must be modified during capture, e.g., in the service industry when a gratuity is added to the original amount. |
| Sale | This call authorizes the transaction and, if successful, captures it. This is the most common call for card-present transactions. (For some transactions, it may be necessary to separate these steps, e.g., in the service industry example described above. In such cases, separate calls to Authorize Only and Prior Auth Capture should be used instead of Sale.) |
| Refund | The Refund method must be linked to a settled transaction. This is done by specifying the transactionId from the original Authorization or Charge as part of the request. By default, this method refunds the FULL amount of the transaction. However, you can perform a partial refund by passing a specific amount. If a refund is attempted on a transaction that has not yet settled, the Integrated Payment Hub will automatically run a Void on the transaction. The transactionType in this case will switch to Void. |
| Void | A void may be applied against any charge that has not yet settled; it effectively undoes the original authorization or charge as if it had never taken place. |
| Credit | A credit is a payment from a merchant account to a credit card or checking account that does not have to be linked to a previous transaction. To use the Credit method, the merchant must be specifically enabled for it on the Worldpay Total platform. This is not recommended due to the risk of misuse. |

## Controlling Settlement with Batches

All transactions that have been authorized and captured are associated with a batch. A batch is considered open until it is settled, which completes the captured transactions and allows funds to be transferred from the customers to the merchant. While open, transactions can still be altered (i.e. voids, adding a tip). Once a batch is settled, it is considered closed; no alterations can be made to transactions in a closed batch.

In some cases, your account may be configured to automatically close out all open transactions and start a new batch at a particular time every night. However, you can close a batch at any time using the Close method. The API also provides reporting methods that allow you to retrieve details for both open and closed batches for a particular merchant.

All batches are automatically closed at 4:00AM ET.

| **Managing Batches** | |
| --- | --- |
| Close a Batch | Closing the current open batch settles all captured transactions in the batch |
| Retrieving the Current Batch ID | Retrieve the ID for the current batch |
| Retrieving Transactions in the a Batch | Calling this method retrieves the transactions in the specified batch. It returns an array of the transactions in the open batch, along with the full details of each as returned during the original authorization. |

# Setting up Your Project in Android Studio

## Step 1 – Get the Lastest IPC SDK

Download the latest IPC SDK framework for Android from GitHub at:

<https://github.com/worldpayus/ipc_sdk_android>

## Step 2 – Framework Intergration

1. Unzip the SDK and add the .AAR file to the project directory (e.g.: /<ProjectName>/)
2. Import the .AAR file into your project by clicking File->New Module->Import .AAR/.JAR file to <ProjectName>.
3. This will create a new project based on the .AAR file. Give the project a name
4. Navigate to the build.gradle file of the application that will be utilizing the project.
5. In the dependencies block, add a line: compile project(‘:<project name’) with the name of the project created off of the .AAR file. In the example below, *gatewaysdk* is the name of the project created from the .AAR file.

**Example:** apply **plugin**: **'com.android.application'***android {  
 compileSdkVersion 23  
 buildToolsVersion* ***"23.0.2"*** *defaultConfig {  
 …..**}  
 buildTypes {*

*……  
}  
}  
  
dependencies {**compile project(****':gatewaysdk'****)  
}*

1. Rebuild the project. The application now has access to the payment and swiper SDK classes.

# Your Application

|  |
| --- |
| **Asynchronous Responses** |
| Communication over the network to the Integrated Payment Hub happens asynchronously in a background thread. When you initiate a payment transaction, control returns immediately to your application. The IPC SDK manages the background thread and waits for the response from the Integrated Payment Hub. |

Once the framework installed and the project has been configured, you can begin to code your application. You can use the features of the IPC SDK in a variety of ways to meet your payment needs.

To get you started here is a basic application flow:

* Obtain an authorization token
* Import the TransactionDialogFragment
* Get card information and process the transaction
* Handle the transaction response

## Step 1 – Create an Auth Token

Each Task & corresponding Request object require the AuthToken, MerchantKey, MerchantId, DeveloperId, and Application Information. Before any other calls can be made an auth token must be created. **It is the responsibility of the implementer to maintain a reference of and secure the auth token.**

**Example:**

AuthTokenCreateRequest request = new AuthTokenCreateRequest();  
if (mMerchantGatewayType == MerchantGatewayType.*SECURENET*) {  
 request.setSecureNetId(BuildConfig.*GATEWAY\_ID*);  
 request.setSecureNetKey(BuildConfig.*GATEWAY\_KEY*);  
} else if (mMerchantGatewayType == MerchantGatewayType.*MERCHANT\_PARTNERS*) {  
 mMainActivity.showErrorDialog(R.string.*error\_unsupported\_gateway*);  
 return;  
} else {  
 mMainActivity.showErrorDialog(R.string.*error\_unsupported\_gateway*);  
 return;  
}  
  
request.setApplicationId(BuildConfig.*APPLICATION\_ID*);  
request.setApplicationModel(Build.*MODEL*);  
request.setApplicationOs("Android");  
request.setTerminalId("");  
request.setTerminalVendor("Miura");  
request.setDeveloperId(BuildConfig.*DEVELOPER\_ID*);

## Step 2 – Begin making calls.

All web service calls have a corresponding AsyncTask. Web service calls are made by initializing the desired Task and passing in the appropriate Request object. In order to handle the response, the implementer should override the onPostExecute method which passes the appropriate Response object back to the implementer.

**Example:**

new CreateAuthTokenTask(request) {  
 @Override  
 protected void onPostExecute(AuthTokenCreateResponse authTokenCreateResponse) {  
 super.onPostExecute(authTokenCreateResponse);  
 mMainActivity.hideProgressDialog();  
 if (authTokenCreateResponse != null) {  
 if (!authTokenCreateResponse.hasError()) {  
 if (!TextUtils.*isEmpty*(authTokenCreateResponse.getAuthToken())) {  
 WorldPaySampleApp.*getInstance*()  
 .setAuthToken(authTokenCreateResponse.getAuthToken());  
 mMainActivity.showTransactionDialogFragment();  
 } else {  
 mMainActivity.showErrorDialog(R.string.*error\_authentication\_failed*);  
 }  
 } else {  
 mMainActivity  
 .showErrorDialog(authTokenCreateResponse.getExceptionMessage());  
 }  
 } else {  
 mMainActivity.showErrorDialog(R.string.*error\_no\_network*);  
 }  
 }  
};

## Step 3 – Swiper Integration

1. **Import TransactionDialogFragment**. In your Activity, import the TransactionDialogFragment.

**Example:**

import com.worldpay.ui.TransactionDialogFragment;

1. **Implement TransactionDialogFragmentListener.** This interface exposes delegate methods used to communicate the transaction status to the hosting application.

Example:

public class MainActivity extends AppCompatActivity implements  
 TransactionDialogFragment.TransactionDialogFragmentListener {  
  
 @Override  
 public void onTransactionComplete(TransactionResult transactionResult,  
 PaymentResponse paymentResponse) {  
 if (transactionResult != TransactionResult.*APPROVED*) {  
 showErrorDialog("Transaction Failed:" + transactionResult.toString());  
 } else {  
 showSuccessDialog("Transaction Approved. " +  
 "\n transactionId = " + paymentResponse.getTransactionResponse().getId());  
 }  
 }  
  
 @Override  
 public void onTransactionError(  
 @NonNull TransactionDialogFragment.TransactionError transactionError,  
 @Nullable String s) {  
 showErrorDialog("Transaction Error: " + transactionError.toString() + "\n message: " + s);  
 }  
  
 @Override  
 public void onTransactionReversalFailed(ReversalRequest reversalRequest) {  
 showProgressDialog("Retrying failed reversal, please wait...");  
 PaymentRefundTask task = new PaymentRefundTask(reversalRequest) {  
 @Override  
 protected void onPostExecute(PaymentResponse paymentResponse) {  
 hideProgressDialog();  
 }  
 };  
 task.execute();  
 }  
  
}

1. **Get a new instance of the TransactionDialogFragment.**

Example:

TransactionDialogFragment dialogFragment = TransactionDialogFragment.*newInstance*();

1. **Start the swiper.** Populate the instance of TransactionDialogFragment with the required fields then show it. The dialog will show transaction status messages if necessary, and will trigger the appropriate delegate methods of TransactionDialogFragmentListener.

Example:

dialogFragment.setAuthToken(WorldPaySampleApp.*getInstance*().getAuthToken());  
dialogFragment.setCaptureMode(CaptureMode.*MANUAL*);  
dialogFragment.setDeveloperId(BuildConfig.*DEVELOPER\_ID*);  
dialogFragment.setMerchantId(BuildConfig.*GATEWAY\_ID*);  
dialogFragment.setMerchantKey(BuildConfig.*GATEWAY\_KEY*);  
dialogFragment.setTransactionType(TransactionType.*SALE*);  
dialogFragment.setSwiper(Swiper.*MIURA\_M10*);  
  
TransactionData transactionData = new TransactionData();  
transactionData.setAmount(BigDecimal.*ONE*);  
transactionData.setCashBackAmount(BigDecimal.*ZERO*);  
dialogFragment.setTransactionData(transactionData);  
  
dialogFragment.show(getSupportFragmentManager(), TransactionDialogFragment.*TAG*);

# Additional Reference Information

You can find more information from the following sources:

The developer website:

<http://worldpay.us/developers>

The Worldpay US Integrated Payments Client GitHub page:

<http://github.com/worldpayus>

The Worldpay IPC SDK for Andoid Sample Application:

<https://github.com/worldpayus/ipc_sdk_android/SampleApp>