Android API Developer Guide

This guide illustrates all the steps needed to add WAC's in-app payment functionality to your app.

Step 1: Run the Sample App

See how in-app payment would work in your app -- install the SDK and try out its sample application.

Step 2: Create Your WAC Account

If you do not already have a WAC account, start by creating one.

Step 3: Create Your API Keys

Define your application, planned operator markets, pricing. Then we can provide your API keys.

Step 4: Manage Your App

Customize in-app icons and localize purchase item names for different markets / countries.

Step 5: Update Your App

Add the WAC in-app payments functionality to your app.

Step 6: Test Your App

See if it works, test your in-app payments using WAC's test environment.

Step 7: Certify Your App

Tell WAC how to pay you, answer the compliance questions from your operators, and have WAC handle your publishing credentials.

Step 8: Set Your API Keys Live

Activate your API keys for the markets where you want your in-app purchases to be real monetary transactions.

Step 9: Push App to Markets

Make your app available to customers with WAC's in-app payments inside.

Step 1: Run the Sample App

WAC has included a sample app with its SDK so you can see WAC's in-app payments in action. And we recommend you start off by doing just that -- download WAC's SDK first off and run the sample app. Experience WAC in-app payments for yourself and experience what your customers will experience.

A) Set Up Your Environment

B) Set Up WAC's Payment SDK

C) Run the Sample App

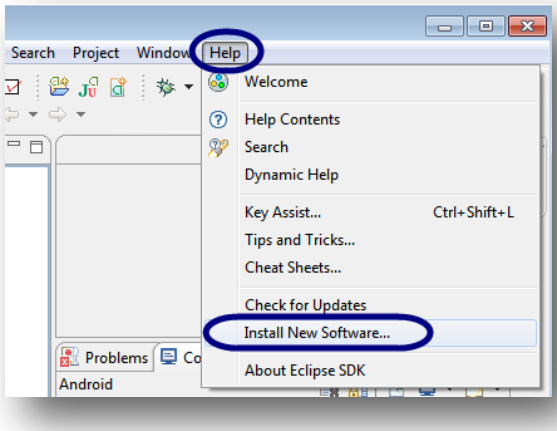
A) Set Up Your Environment

1) Ensure your system has the following correctly configured:

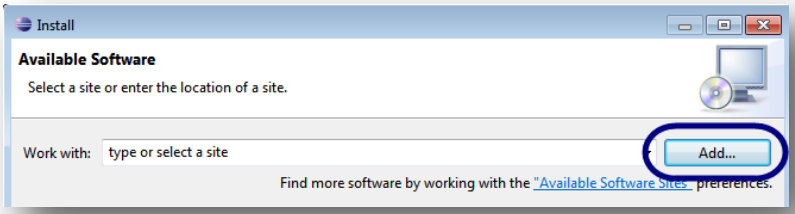
- JDK-6u30 or newer
- Apache ANT 1.8.2*
  **Note: ANT is automatically included with the Eclipse 3.7.1 installation.*
- Eclipse Classic 3.71
- Android SDK 2.2 (or newer)

2) Ensure you have installed the Android ADT plugin for Eclipse:

   a. In Eclipse, click **Help** and then click **Install New Software**.

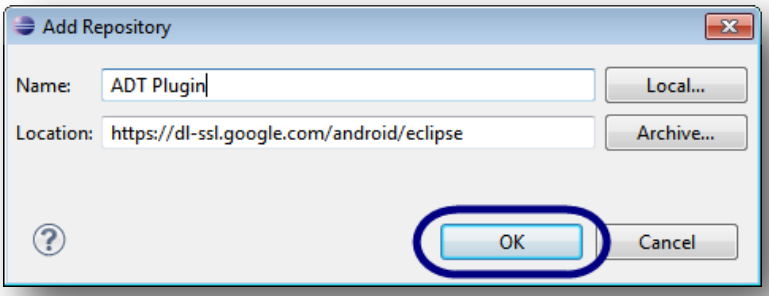


b. Click **Add**.

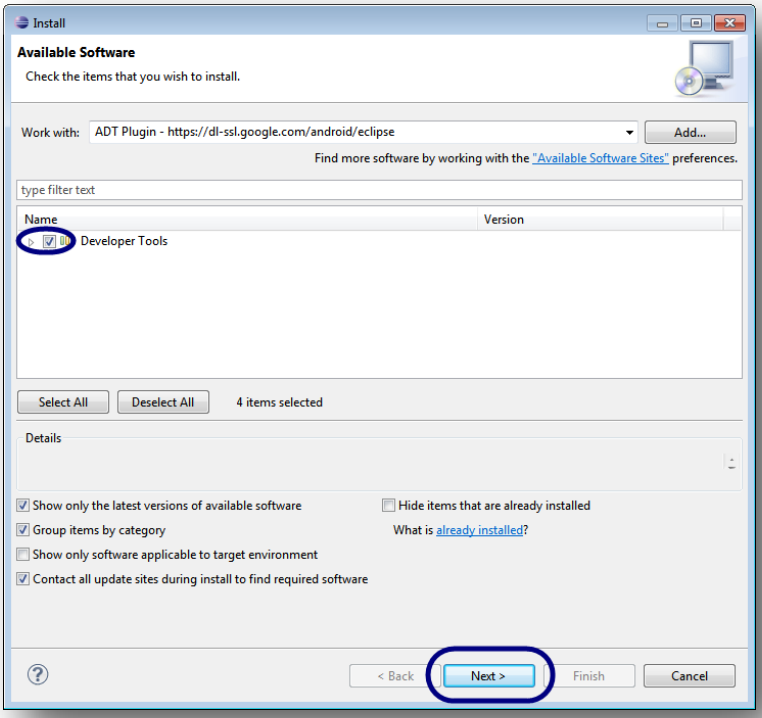


c. Enter ADT Plugin and **https://dl-ssl.google.com/android/eclipse**.

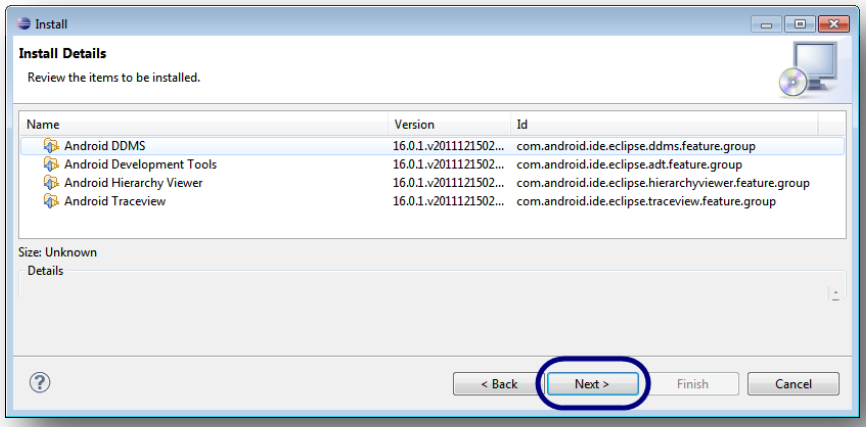
Then click **OK**.



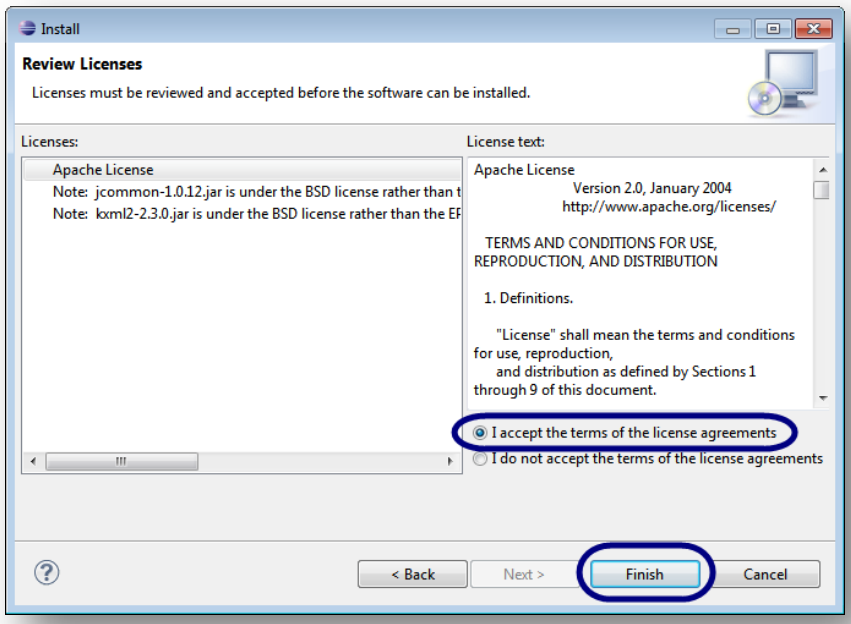
d. Mark the Developer Tools checkbox and click **Next**.



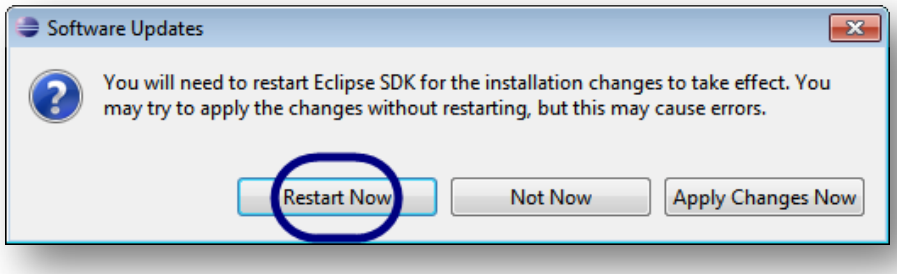
e. Click **Next**.



f. Accept the terms and conditions and click **Finish.**



g. Eclipse installs the ADT. When prompted, click **Restart Now**.

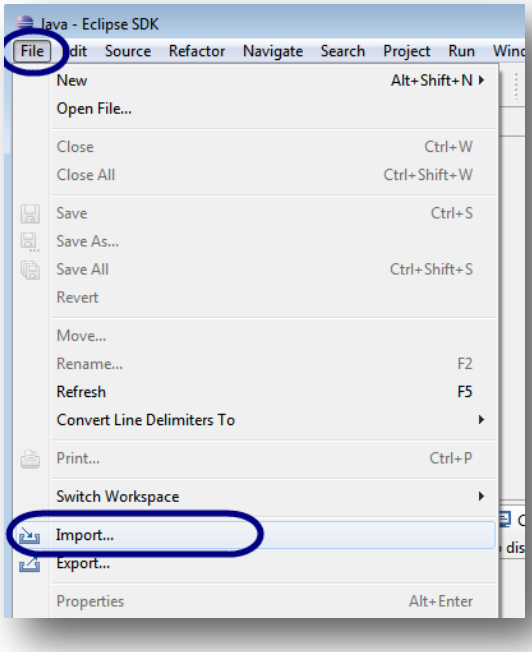


B) Set Up WAC's Payment SDK

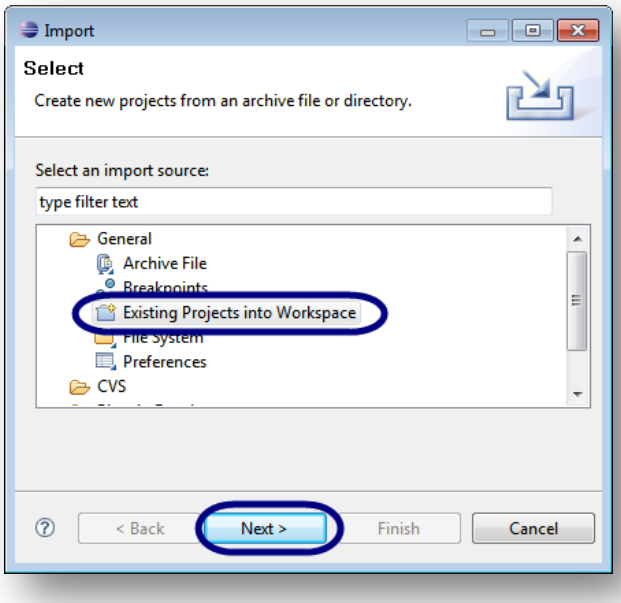
1) Download and extract WAC's SDK: https://www.wacapps.net/sdks.

   Note: This guide's instructions are for **version 1.0.2 of WAC's Android SDK** – make certain that this guide is correct for your version of the SDK. The most current developer guide is always available at https://www.wacapps.net/developer-guide.

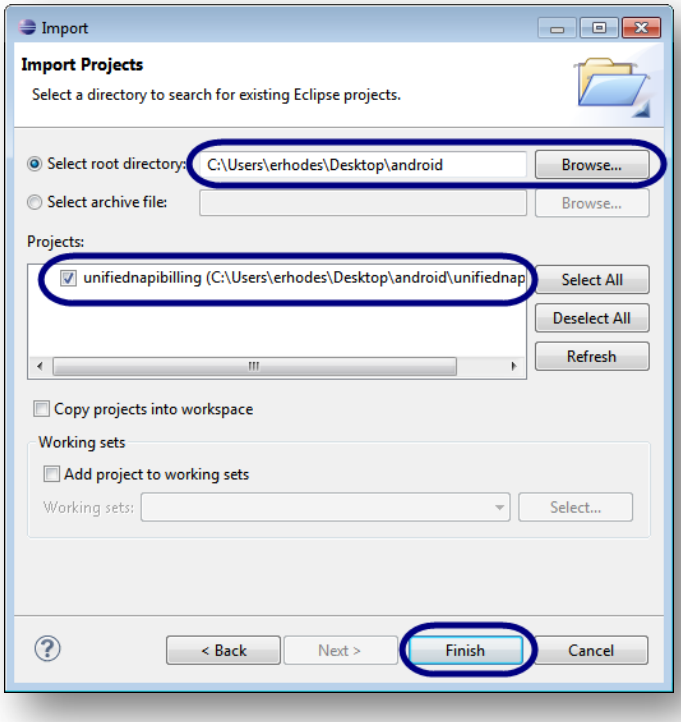
2) Click **File** and **Import**.



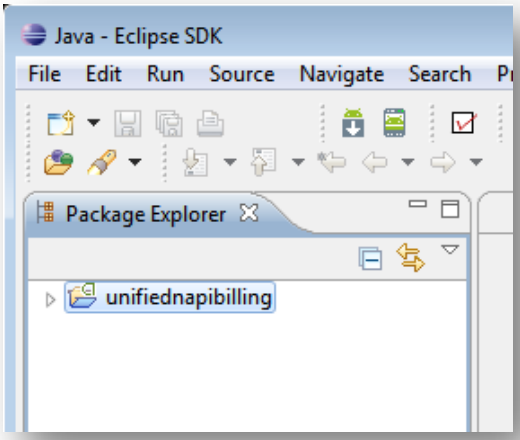
3) Choose **Existing Projects into Workspace**. Then click **Next**.



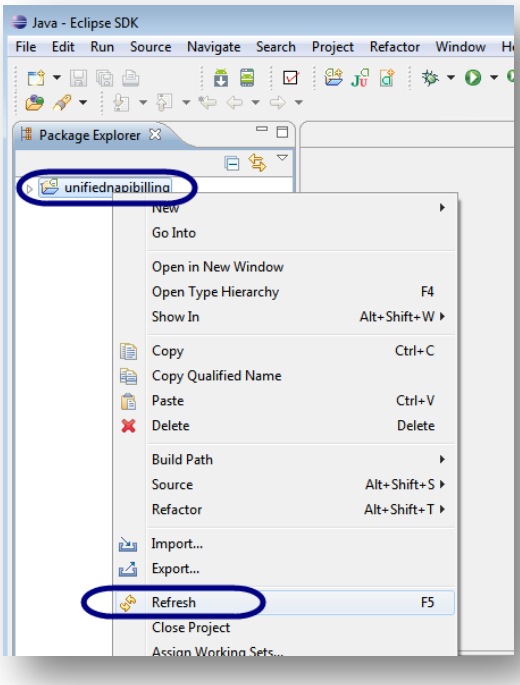
4) Choose your WAC SDK folder and click **Finish**.



5) The SDK package is imported into Eclipse.



6) Right-click the package and click **Refresh.**

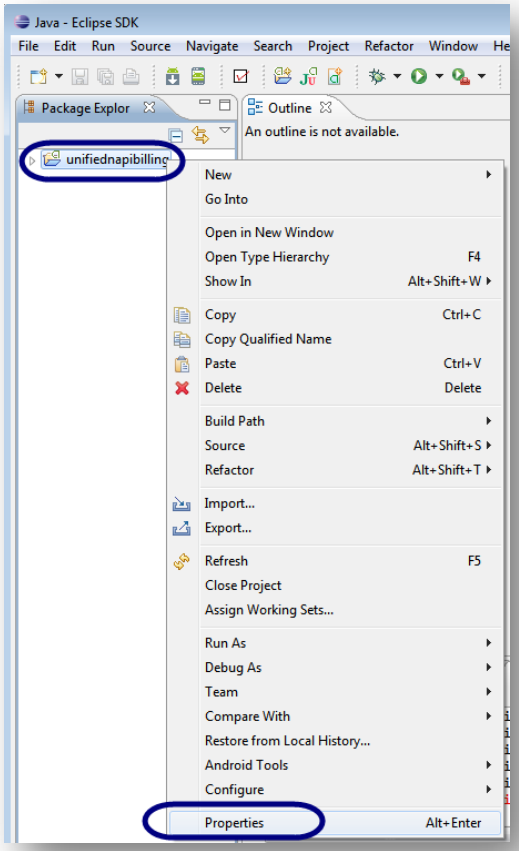


7) If you see the package "**!**" icon and your Eclipse console reports this error message, you will need to update your compiler target version:

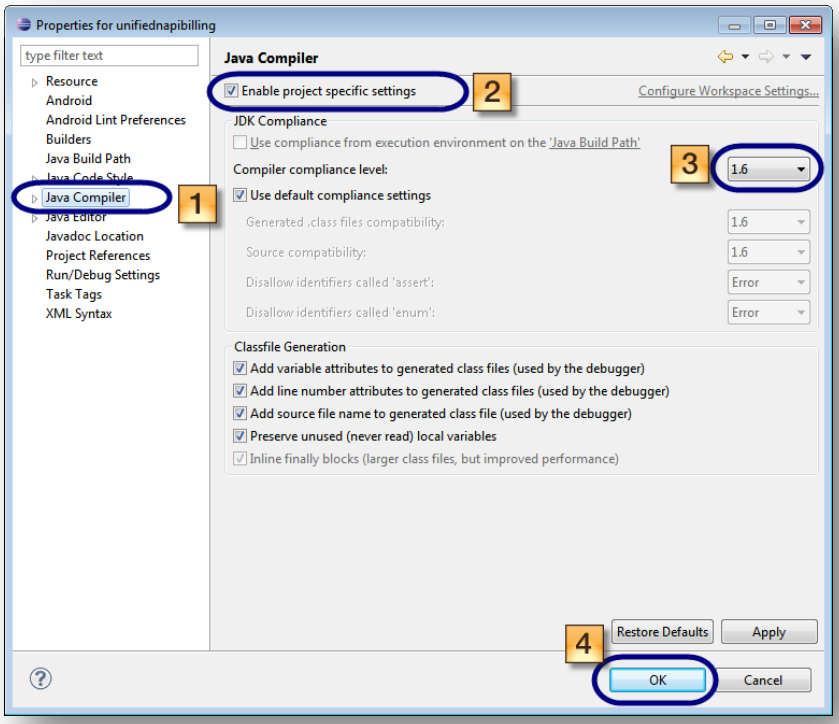
```
Android requires compiler compliance level 5.0 or 6.0. Found '1.4' instead. Please
use Android Tools > Fix Project Properties.
```

Do so as follows:

a. Right-click the package and click **Properties**.



8) Ensure that the Java Compiler option is set to version 1.6 as shown.

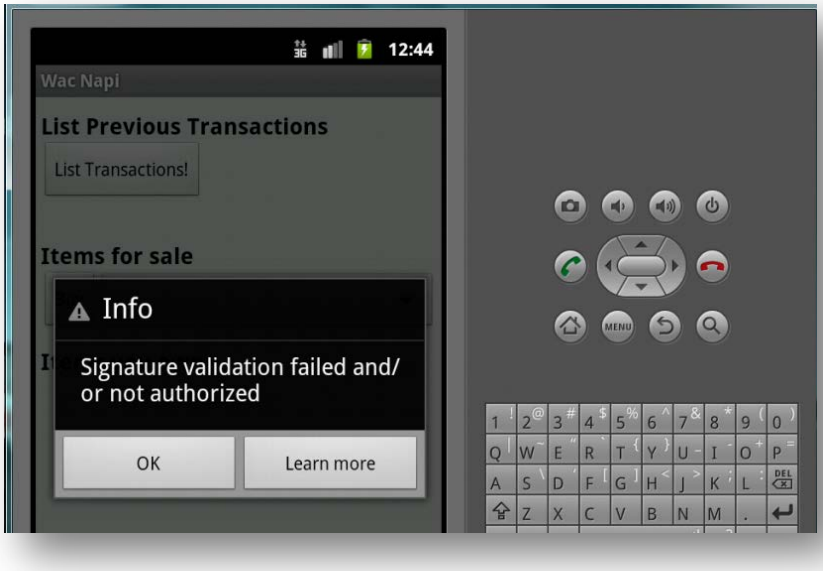


C) Run the Sample App

This sample app must initiate its API calls from a computer with a United States IP address. This is because the SDK sample app's purchase items and prices have been defined for this market. If you are operating outside the United States, be certain to start the app with the correct steps below.

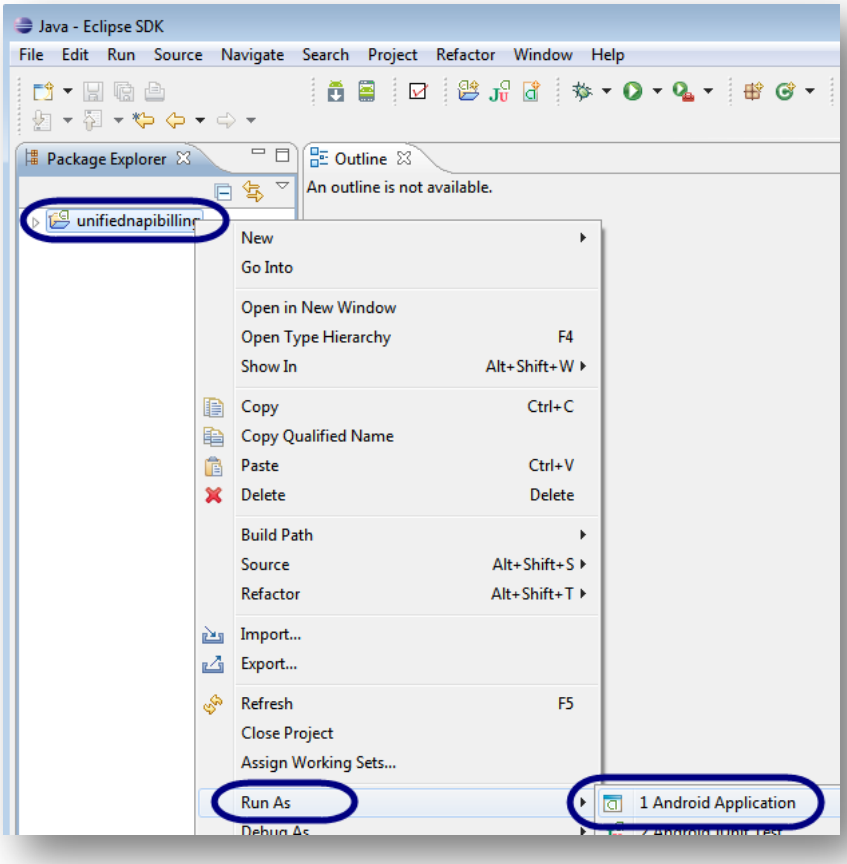
- **Inside the United States**
- **Outside the United States**

Important: Before running the sample app, ensure your computer's clock is set to your correct local time – WAC's payment gateway will report the following error if it receives calls from a system whose time varies more than 3 minutes from the app's local time:



Inside the United States

1) Right-click the package and run as an Android application.



2) After a few moments, the **sample app** should start. Slide to unlock the screen.

Outside the United States

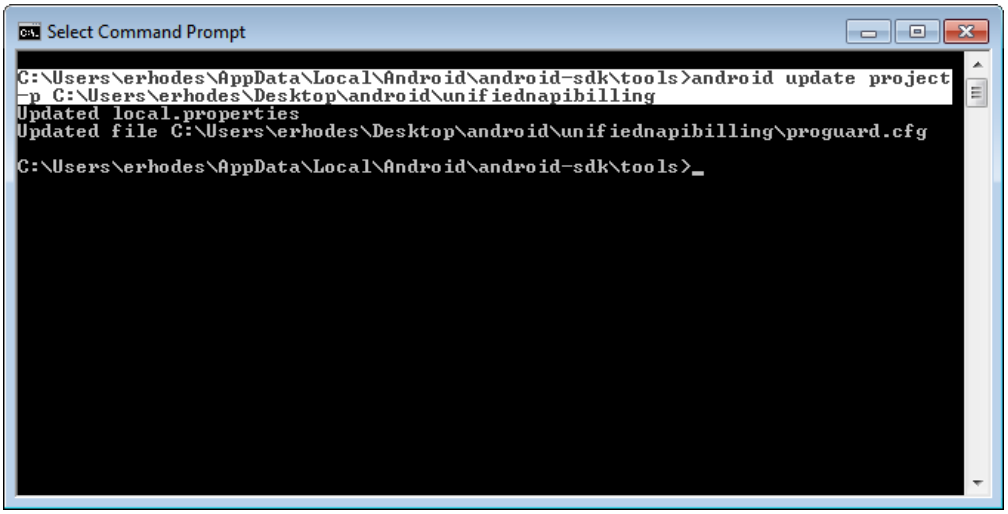
1. Set up the Apache ANT project properties by entering the following in a terminal window: **<Android SDK install folder>/tools/android update project –p <your SDK folder>/unifiednapibilling**.

   Note: Be certain that you have already installed and properly configured Apache ANT 1.8.2, as noted in A) Set Up Your Environment.

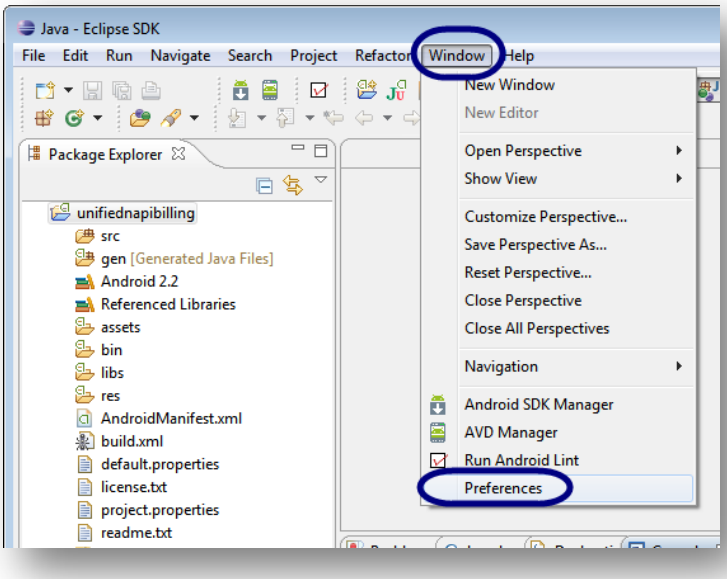
2. Full setup instructions are provided here: http://ant.apache.org/manual/install.html.

Windows Example

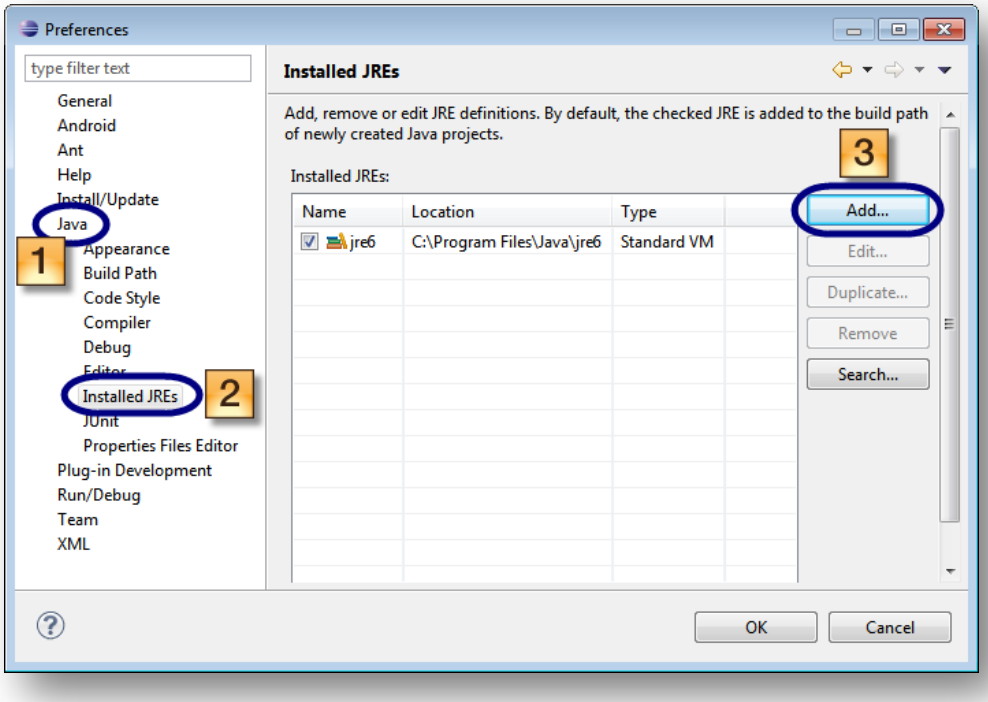
Note: *Your Android SDK install folder will be different.*



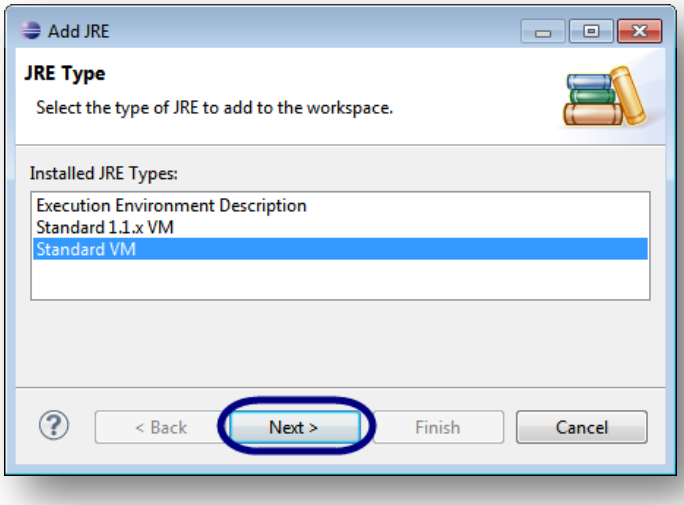
3. In Eclipse, click **Window**/**Preferences**.



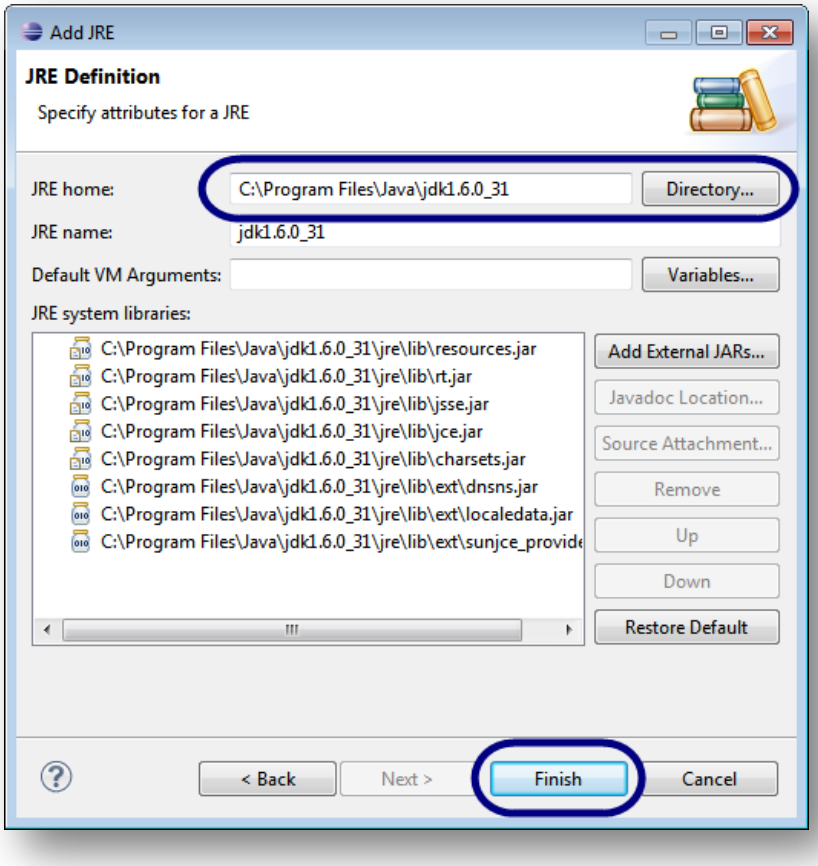
4. Open the **Installed JREs** screen and click **Add**.



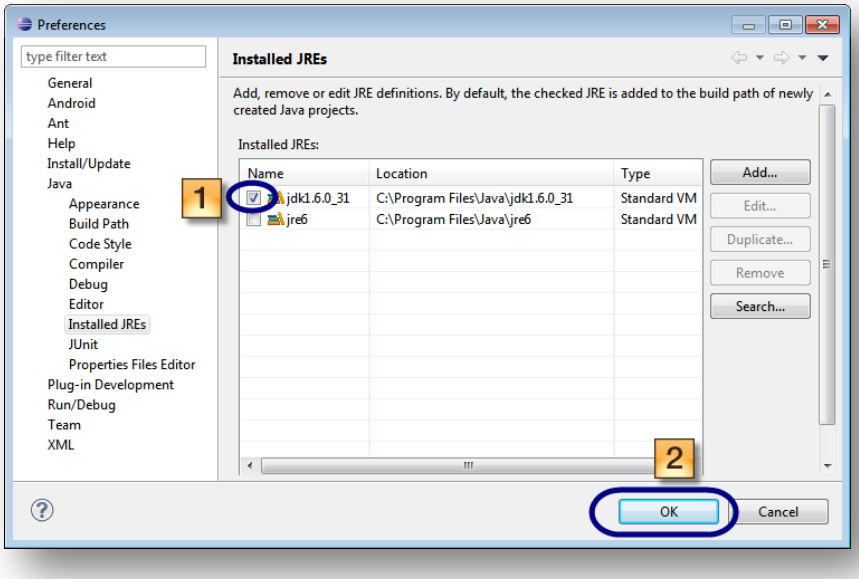
5. Choose **Standard VM** and click **Next**.



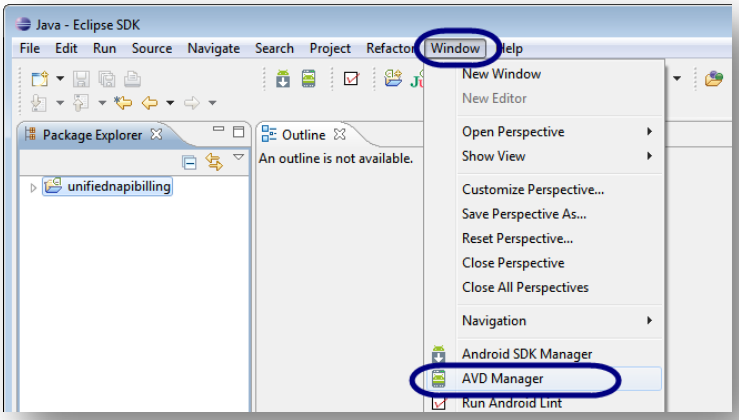
6. Enter the home directory of the JRE and click **Finish**.



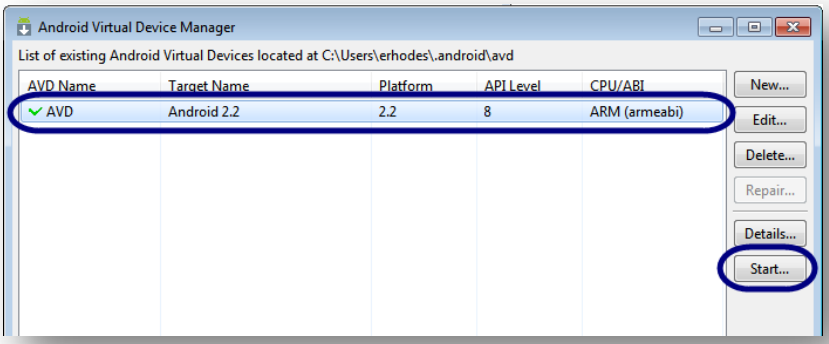
7. Ensure the JDK is selected and then click **OK**.



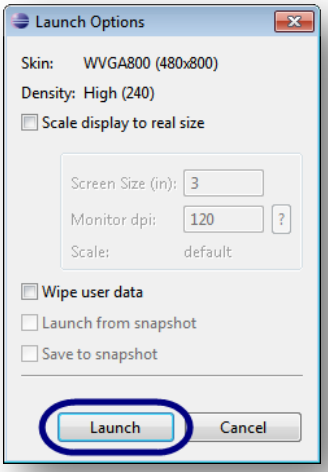
8. In the **Windows menu**, open **AVD Manager**.



9. Select your virtual device and click **Start**.



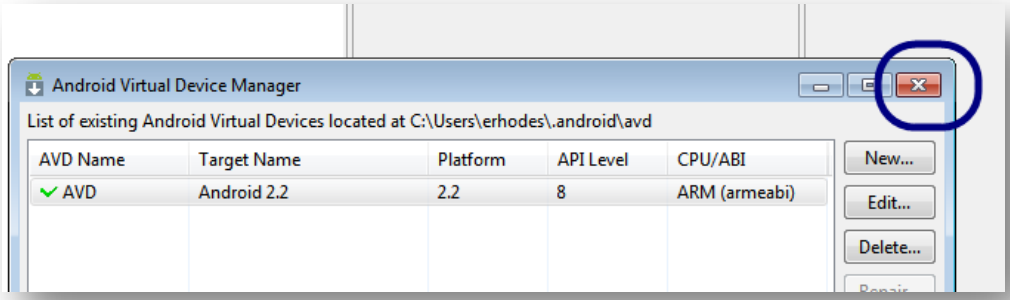
10. Click **Launch**.



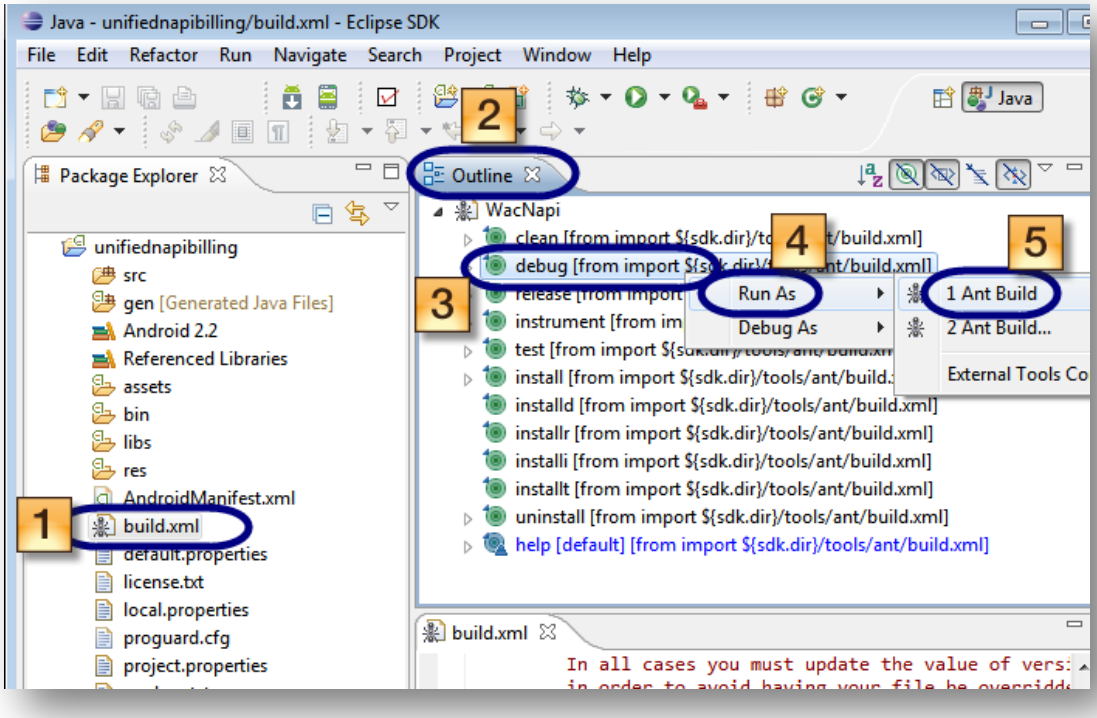
11. When the Android emulator starts, slide to unlock the screen.

    Note: It may take a few minutes for the emulator to start.

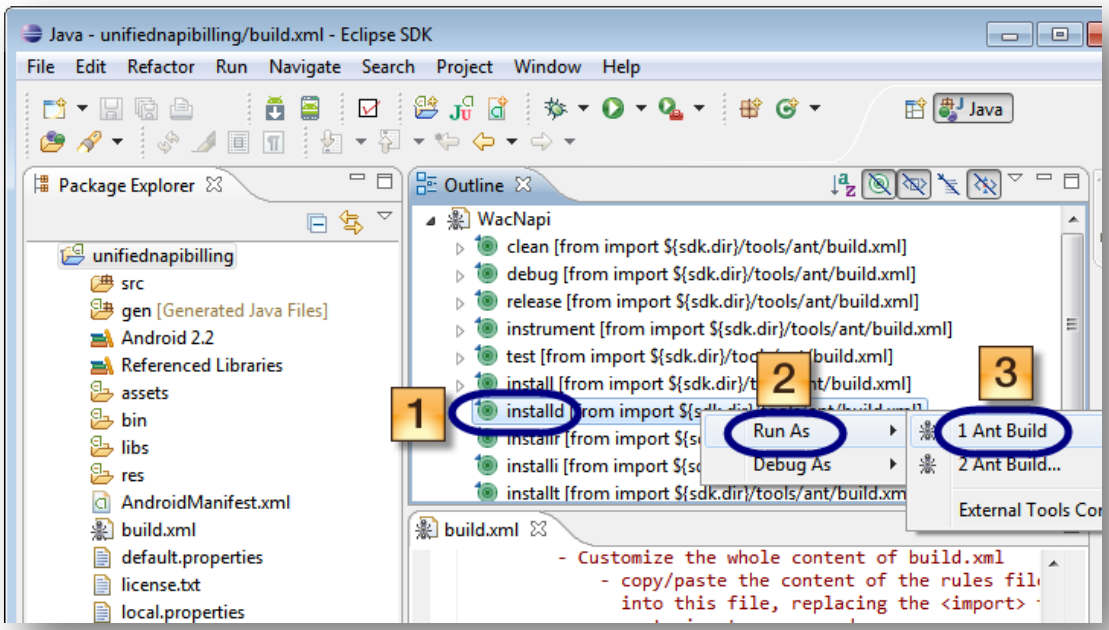
12. Close the Android Device Manager window.



13. In Eclipse, open the **build.xml file** in the **Outline view**.
    Then run the **debug target** by right-clicking it and choosing **Run As>Ant Build**.



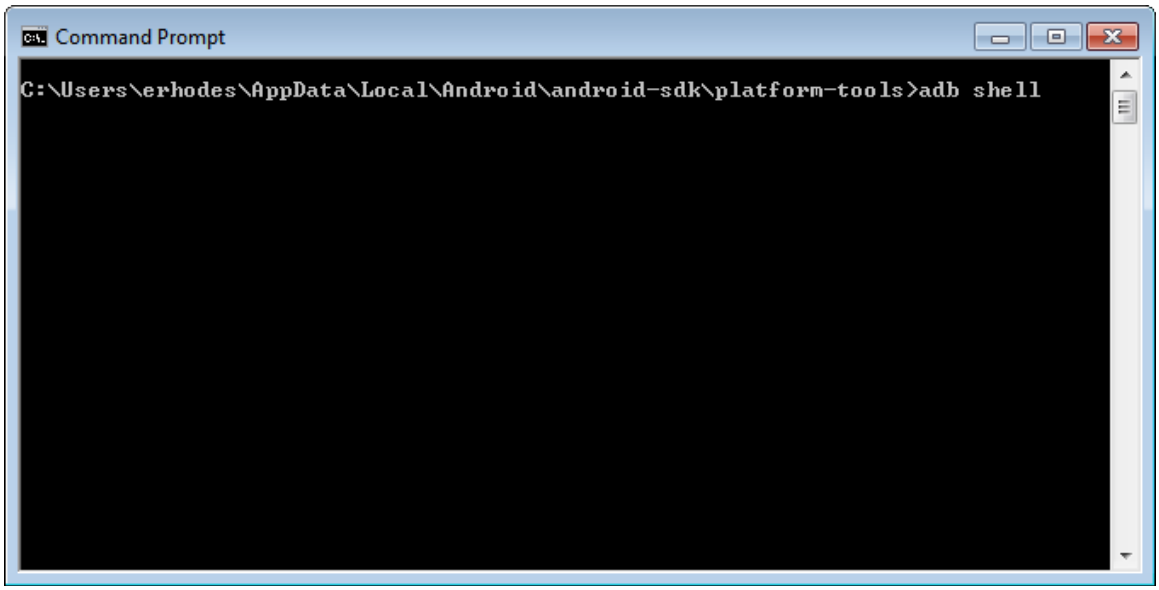
14. Run the **installd** target the same way.



15. Start the ADB shell: In your **<Android SDK install folder>/platform-tools** folder, enter **adb shell**.

Windows Example

Note: *Your Android SDK install folder will be different.*

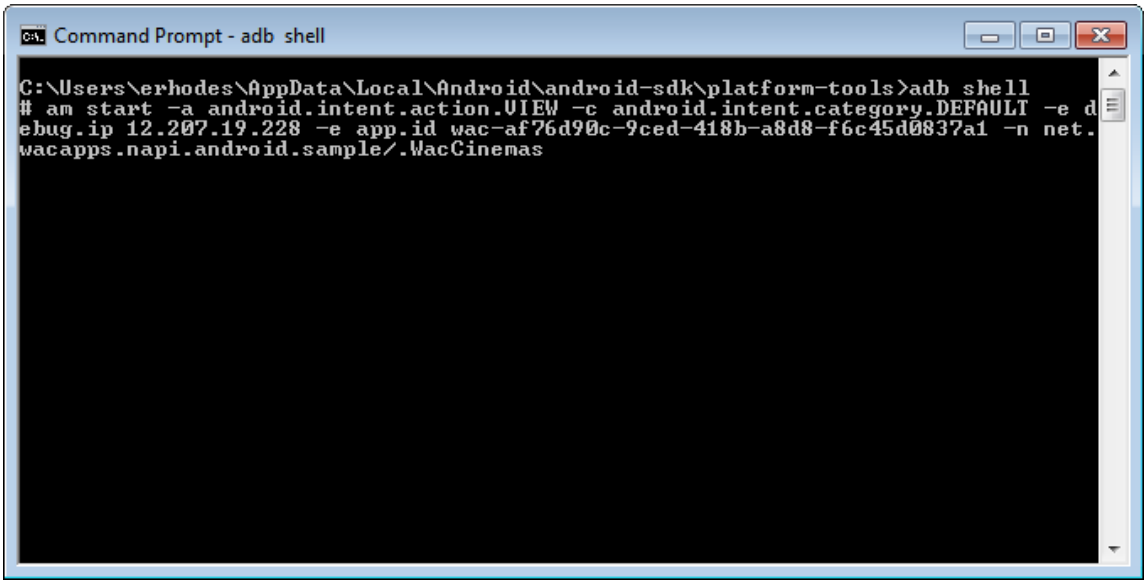


16. Have your computer to "spoof," imitate, this U.S. IP address: **12.207.19.228**. Do this by entering this command line in the ADB shell:

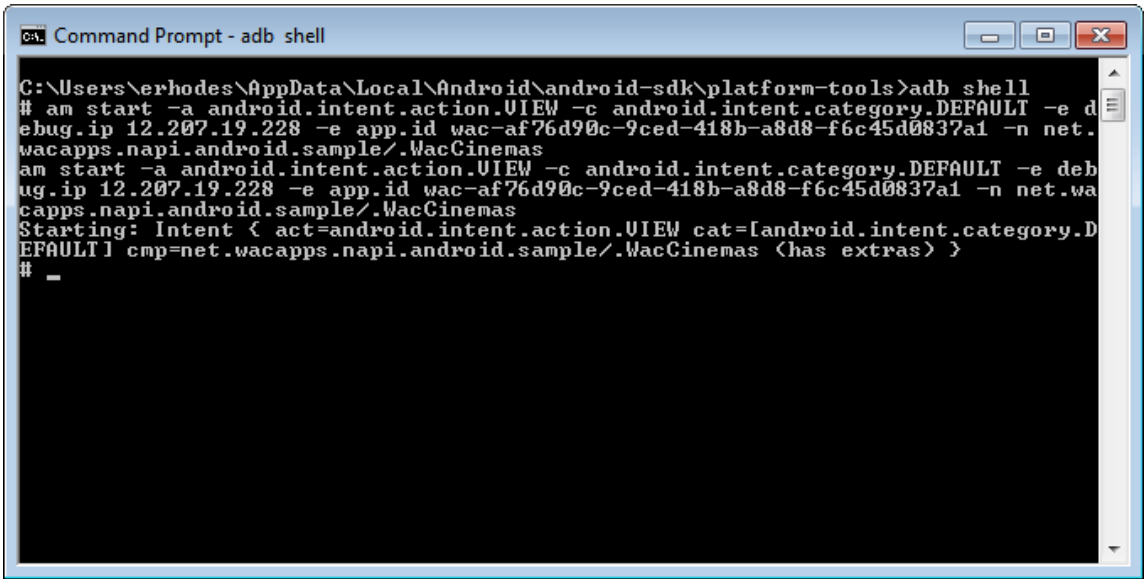
OS	Command
Windows	`am start -a android.intent.action.VIEW -c android.intent.category.DEFAULT -e debug.ip 12.207.19.228 –e app.id wac-af76d90c-9ced-418b-a8d8-f6c45d0837a1 -n net.wacapps.napi.android.sample/.WacCinemas`
OSX / UNIX	`./am start -a android.intent.action.VIEW -c android.intent.category.DEFAULT -e debug.ip 12.207.19.228 -e app.id wac-af76d90c-9ced-418b-a8d8-f6c45d0837a1 -n net.wacapps.napi.android.sample/.WacCinemas`

Windows Example

A) Spoofing command



B) System reply

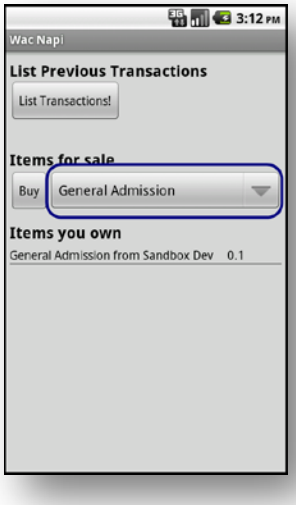


13) The **WacCinemas** sample app should now be running.

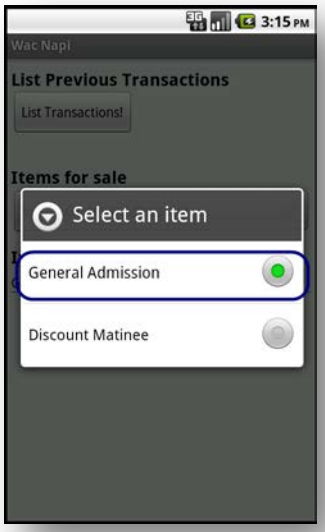
WAC's payment APIs are implemented in this app with these key functions:

Function	Description
Purchases	App users can purchase items in the app and be billed for them by their mobile operator. Later, purchases are automatically reconciled, consolidated, and revenue deposited in your financial account.
Transaction History	Users can see all their past transactions with you, the app provider.
Transaction Validation	The SDK provides transaction validation. Developers can use the sample app to execute and test this functionality.

14) Click the sale item selector.

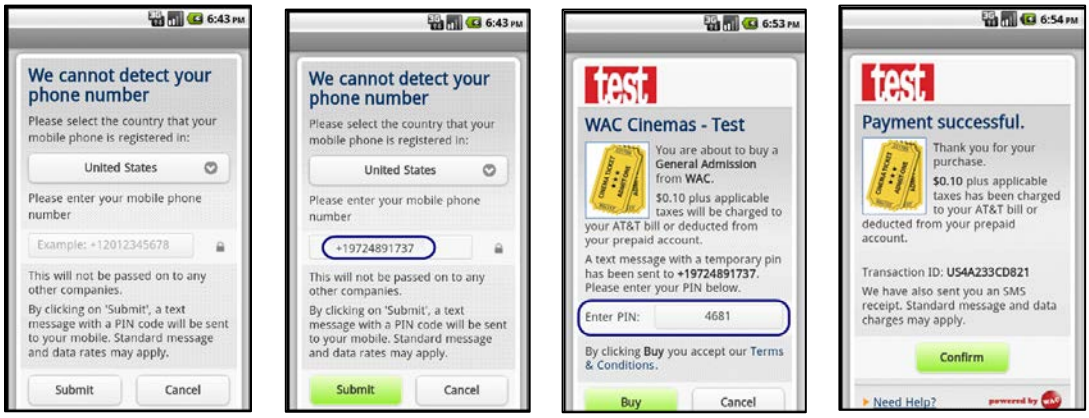


15) Choose an item to purchase.



16) Click **Buy**.
Customers using their cellular connection jump straight to the confirmation screen and complete their transaction.

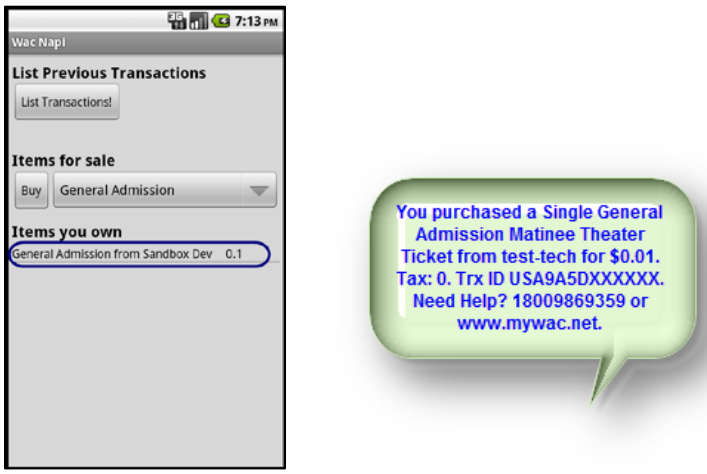
17) Customers using Wi-Fi instead of a cell connection cannot immediately be associated with their operator. For identification, these customers will be asked to provide their cell number and confirm receipt of a PIN sent to the number by SMS. Then the customer can confirm the purchase and the charge is applied to the customer's mobile operator billing.



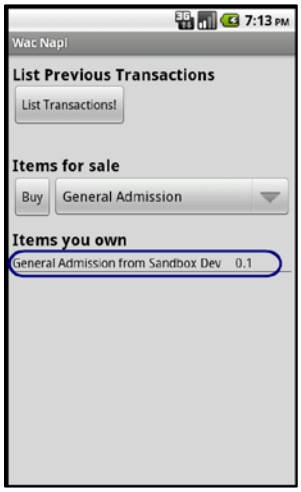
The sample app has been set up with WAC's sandbox test environment to let test each possible transaction. Do so by entering the following phone numbers and using PIN **4681**.

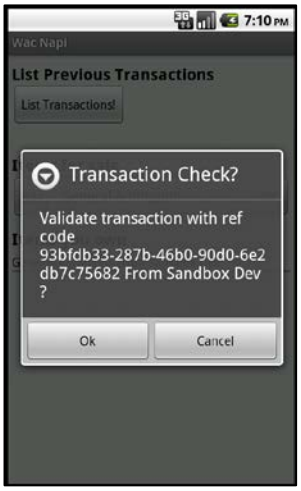
Scenario	What to Enter	Message
Successful payment	+19724891737	Payment successful.
Spend limit error	+19724891648	You have reached your operator spend limit. Please contact your operator for help.
Payment failed	+19724892325	Payment failed, please try again later. You will not be charged for this transaction.
Operator not supported	Any operator number not live on WAC	Operator not supported.

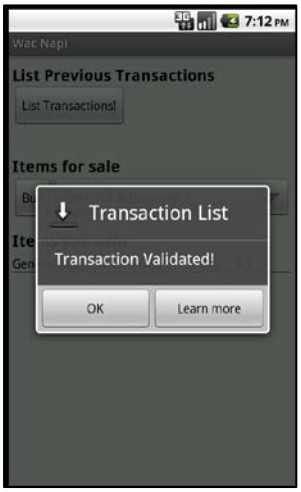
18) With a successful purchase, the customer will see the purchase item and receive an SMS confirmation:



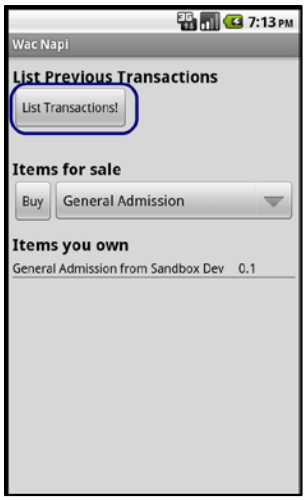
19) The sample app lets developers trigger the SDK's transaction validation functionality while testing. Do this by clicking the purchased item under "Items you own."

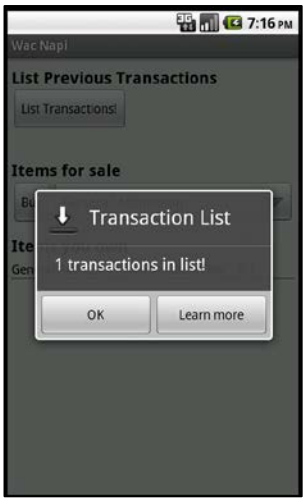






20) To initiate the transaction list functionality, click **List Transactions**.

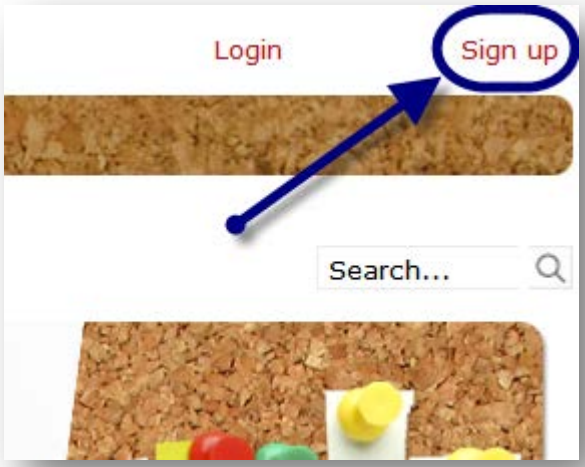




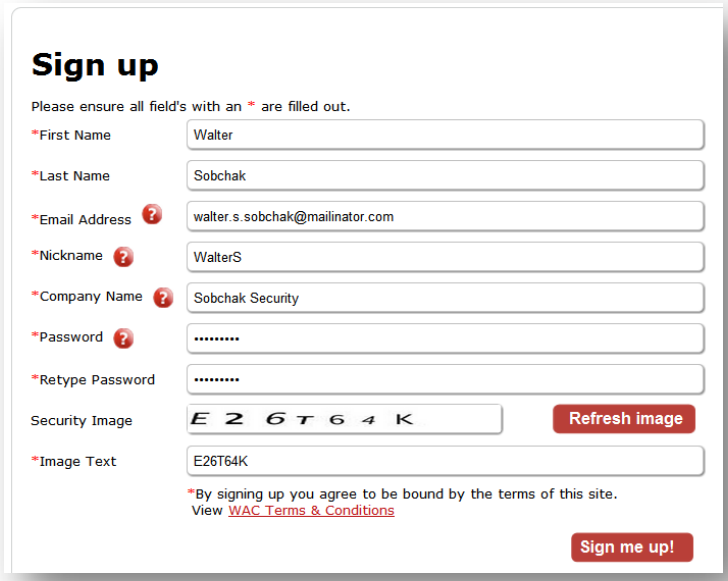
Step 2: Create Your WAC Account

If you do not already have a WAC account, create one now to tell WAC who you are and how to deposit the proceeds from your in-app sales.

1) Go to http://www.wacapps.net/ and click **Sign up**.



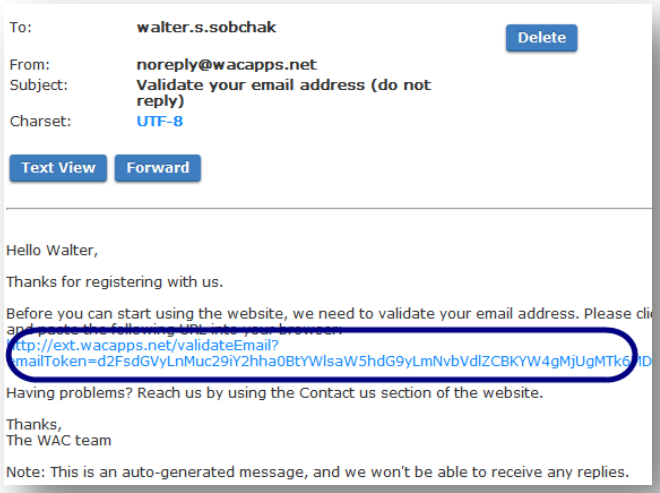
2) Complete the Sign up screen and click **Sign me up**.



Field	Description
Email Address	We will use this email to send critical updates, be certain to use an account you check frequently.
Nickname	This will identify you when you post to forums and must be unique.
Company Name	Just enter your name if you do not represent a company.
Password	Must be composed of between 6 and 20 letters and numbers.

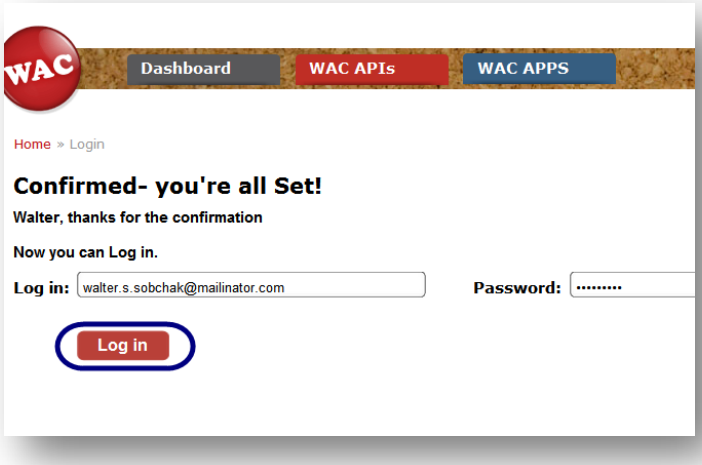
3) You should immediately receive an email asking you to validate your registration.

   Click the validation link.

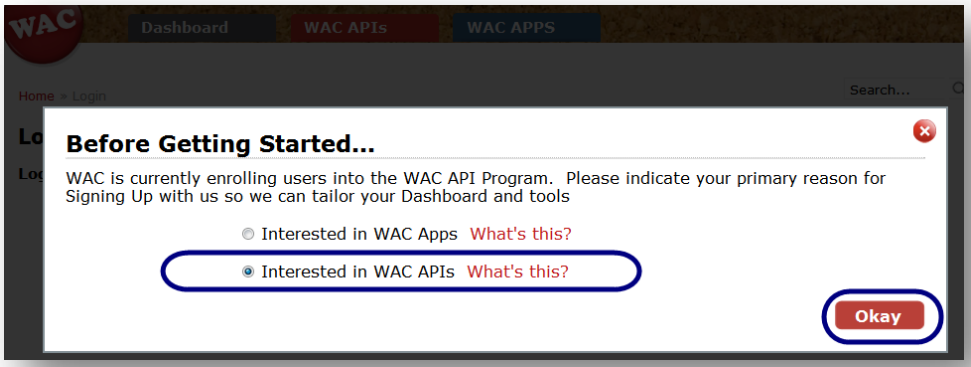


4) Your browser opens and you see your account creation is confirmed.

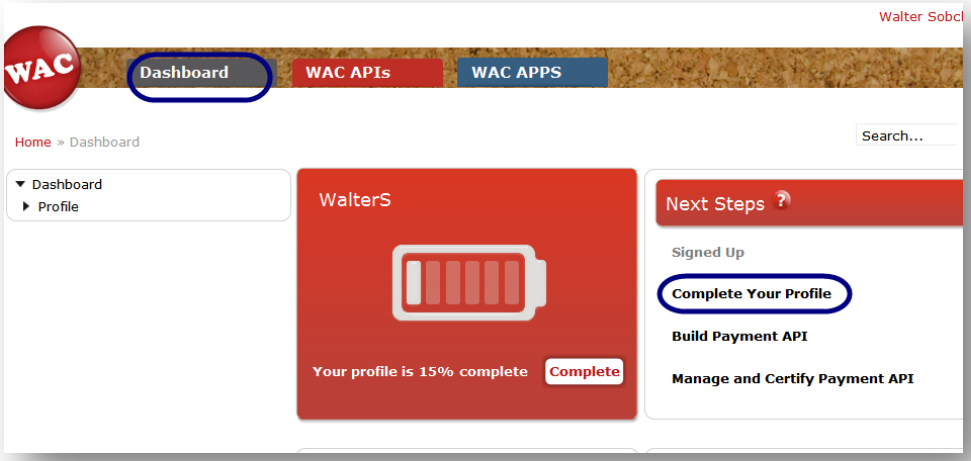
   Log in with the credentials you just created.



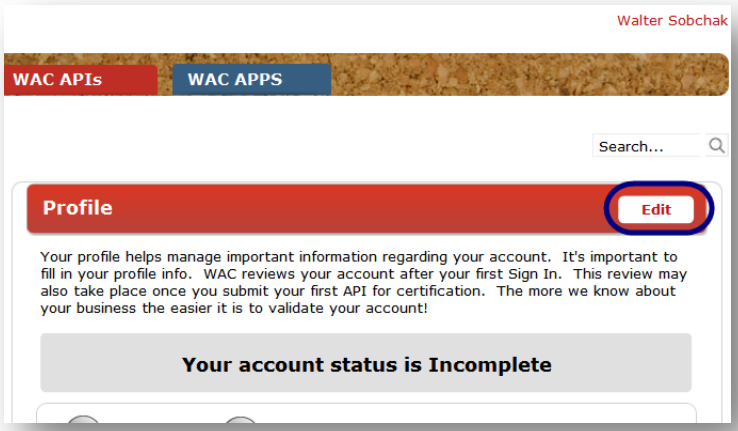
5) Choose the **WAC APIs option** and click **Okay**.



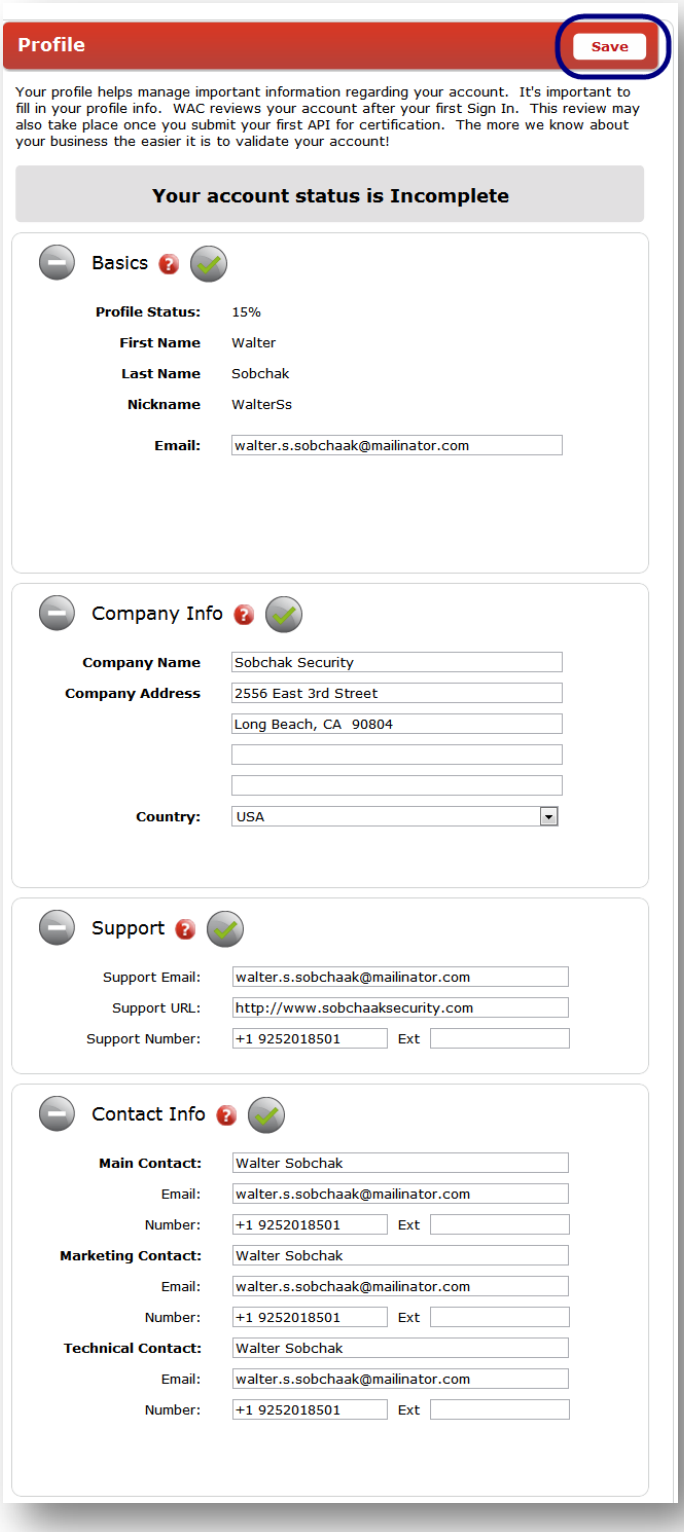
6) On the Dashboard screen, click **Complete Your Profile**.



7) On the Profile screen, click **Edit**.

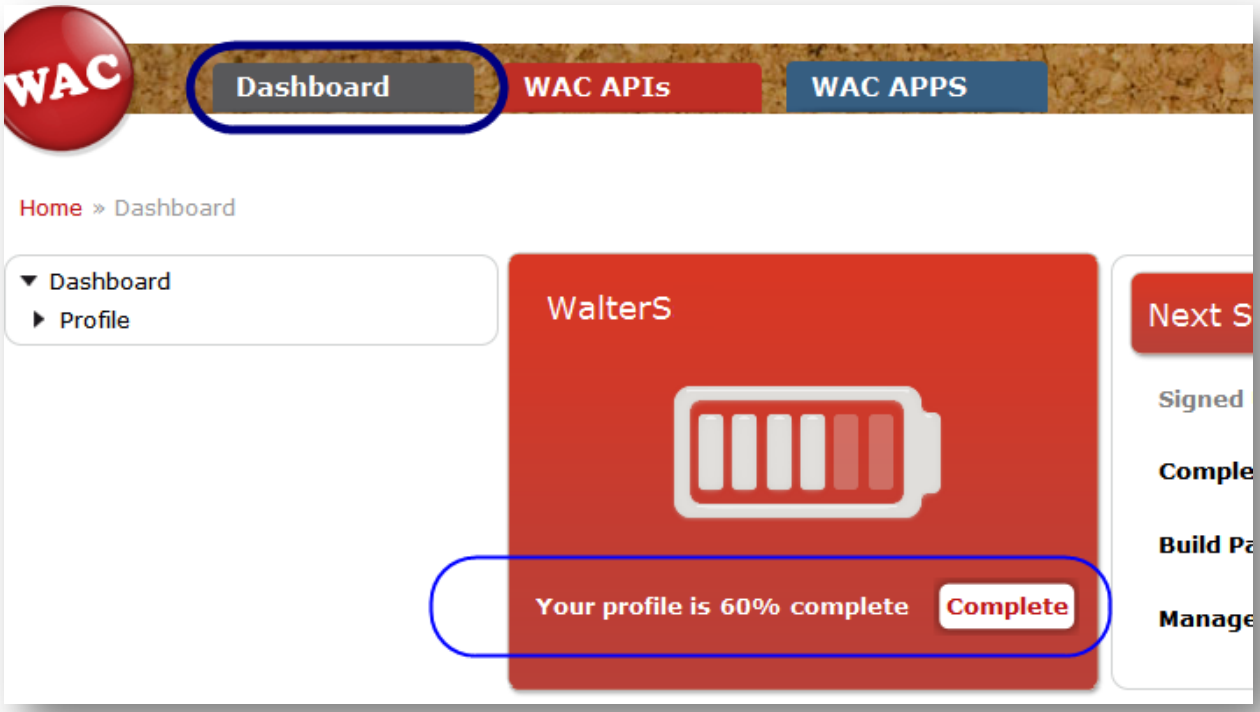


8) Enter your information as shown and click **Save**.



9) Click **Dashboard.**

Your profile is not yet complete because your banking details that tell WAC how to pay you for your app sales are not yet entered. You do not need to do this yet, leave your profile now and move on to Step 3: Create Your API Keys.

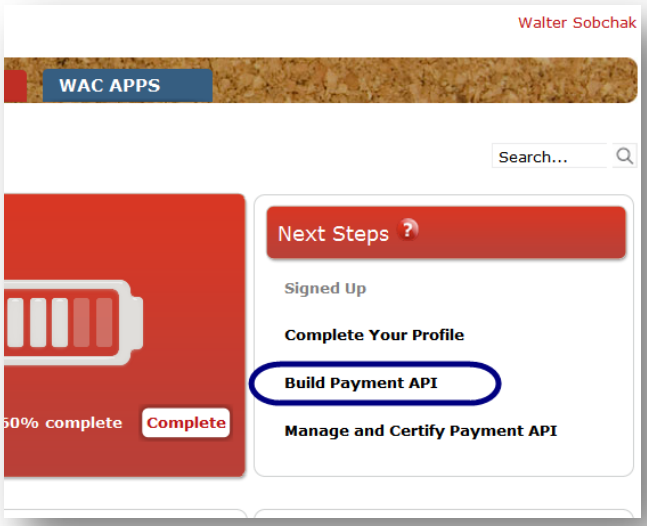


Step 3: Create Your API Keys

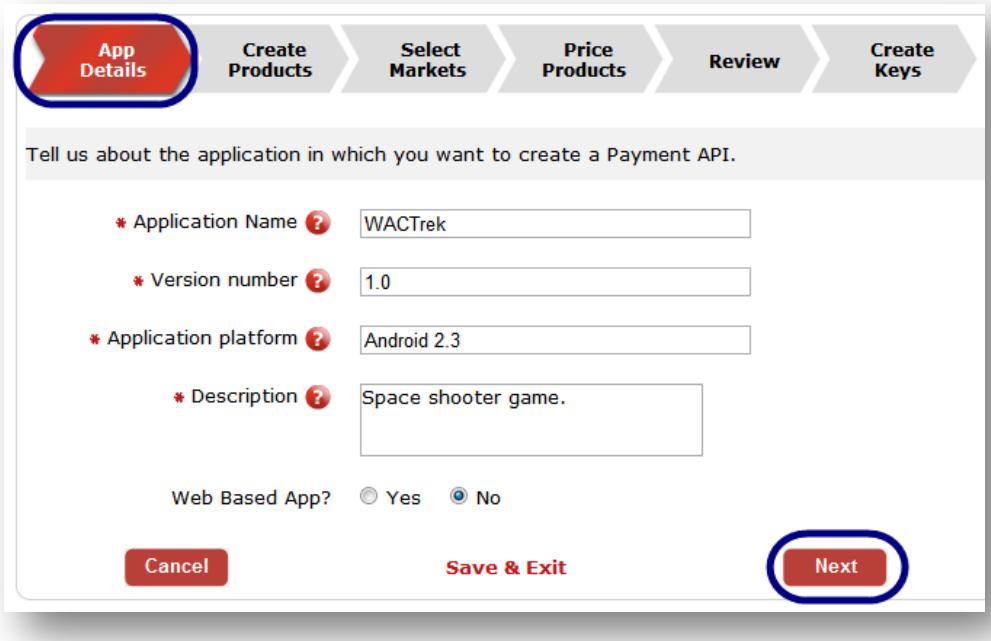
Now tell WAC what you will sell inside your app, the operator markets to target, and how much to charge in each market.

After you enter this information, WAC can provide you the API keys that your app can use to conduct in-app payments with WAC's servers.

1) Click **Build Payment API**.



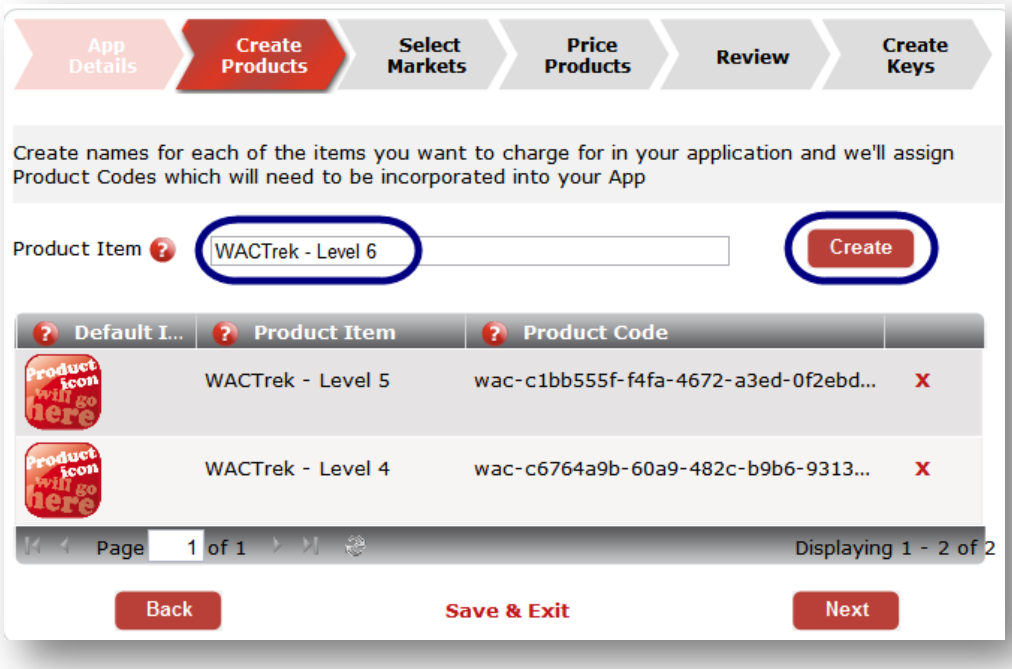
2) Complete the **App Details** screen and click **Next**.



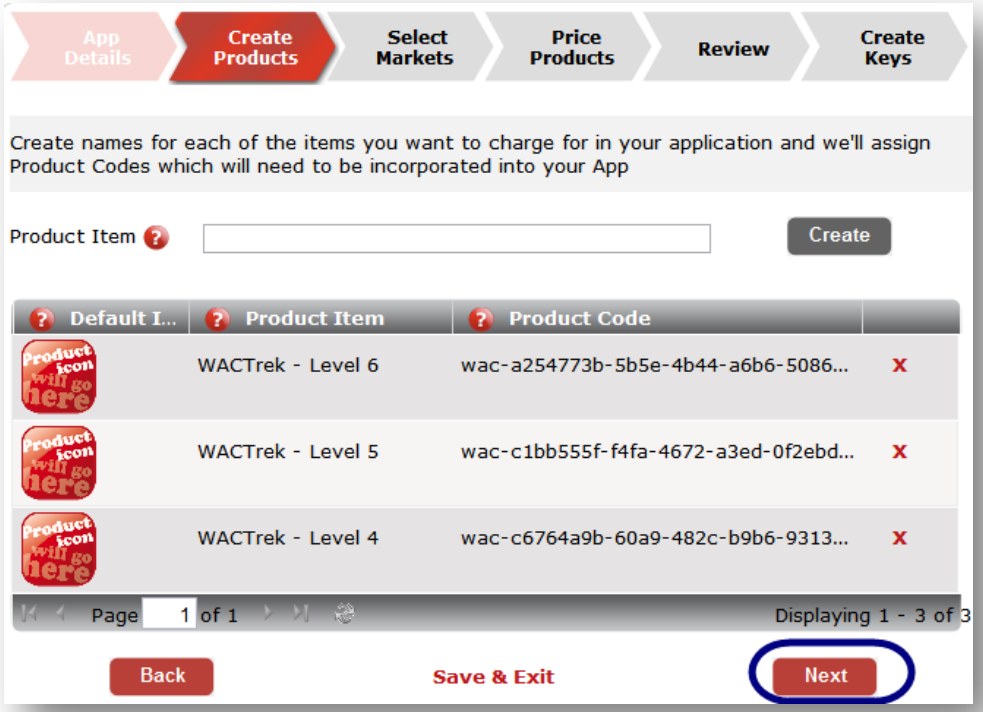
Field	Description
Application Name	Enter the name customers will see when they view their purchase history.
Version Number	Enter no more than five digits (numbers or "." only) to identify this payment API to you and to customers viewing their purchase history. **Note:** You can use the same version number and application name for multiple in-app payment APIs. However, choose a version number that makes it easy for you and for customers viewing their purchase history to tell which item was purchased.
Application Platform	Indicate which platform the app runs on.
Description	This description (1) helps the WAC Compliance Team understand your app's functionality and (2) helps differentiate between apps with the same name.
Web Based App?	For Android applications, choose **No**.

3) On the **Create Products screen**, you define each item customers can purchase in your app:

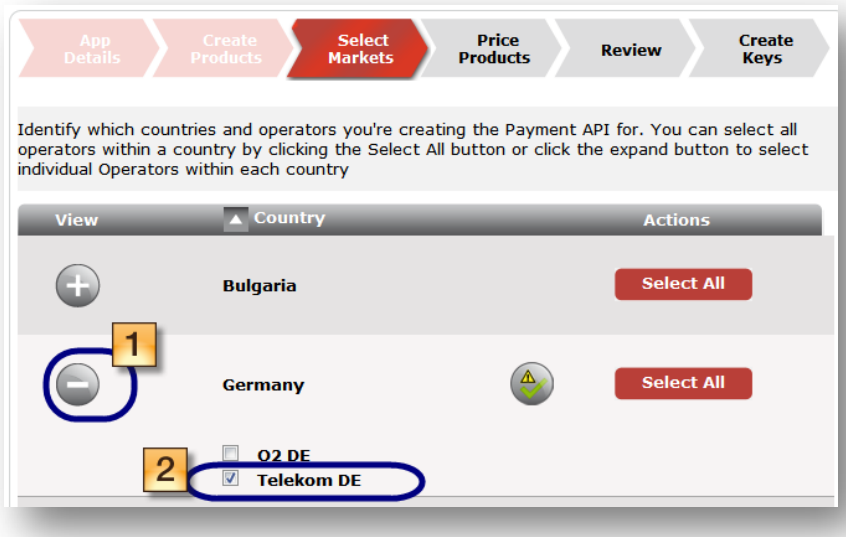
For each item customers can purchase in your app, enter a product item name (*25 characters max*) and click **Create**.



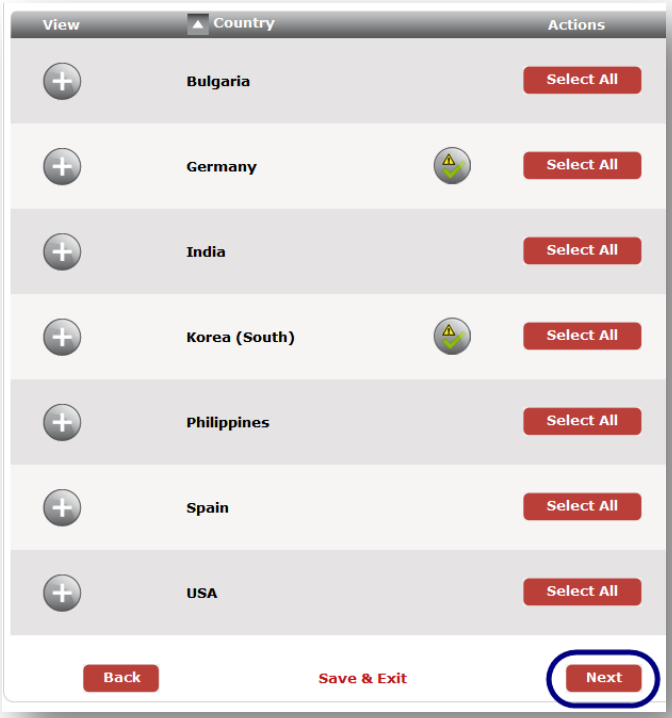
4) When finished, click **Next**. (You can always return to edit and add additional items.)



5) On the **Select Market screen**, mark the operator checkbox for each operator in each country with whom you will market your app.



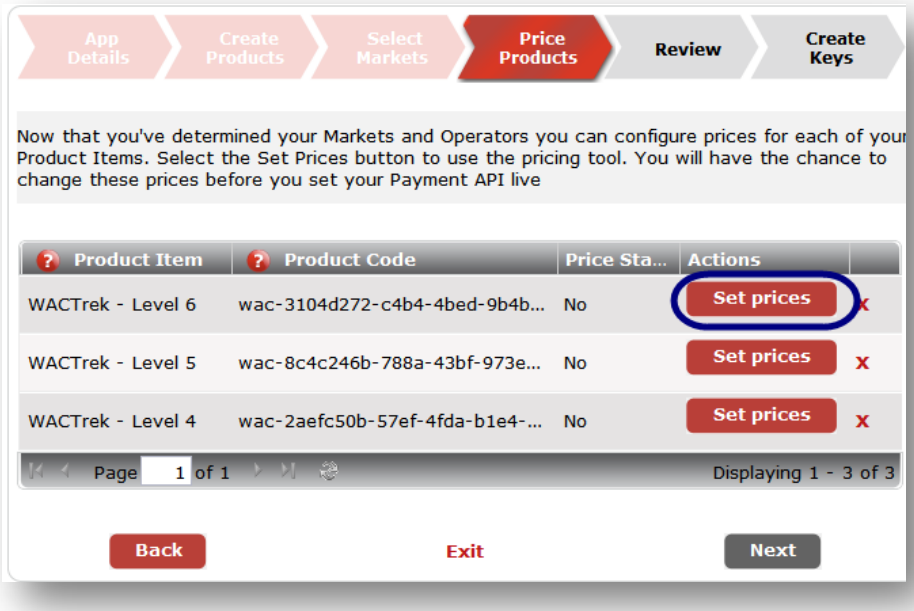
6) Click **Next** when finished.



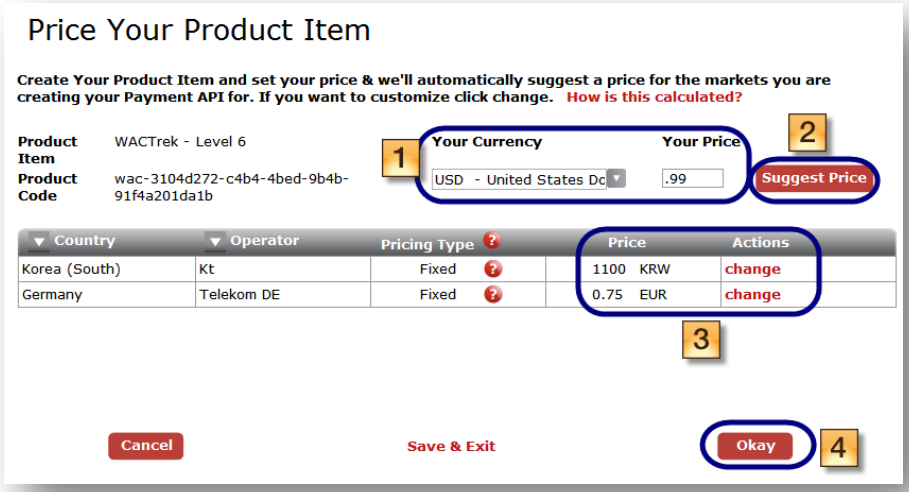
7) On the **Price Products** screen , you are ready to enter prices for each in-app sale item you defined.

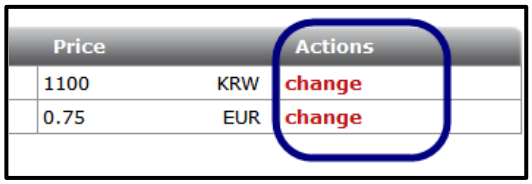
Note: You can edit these prices later, including after the app is pushed live for market purchase.

Click the **Set prices button** for the first item.



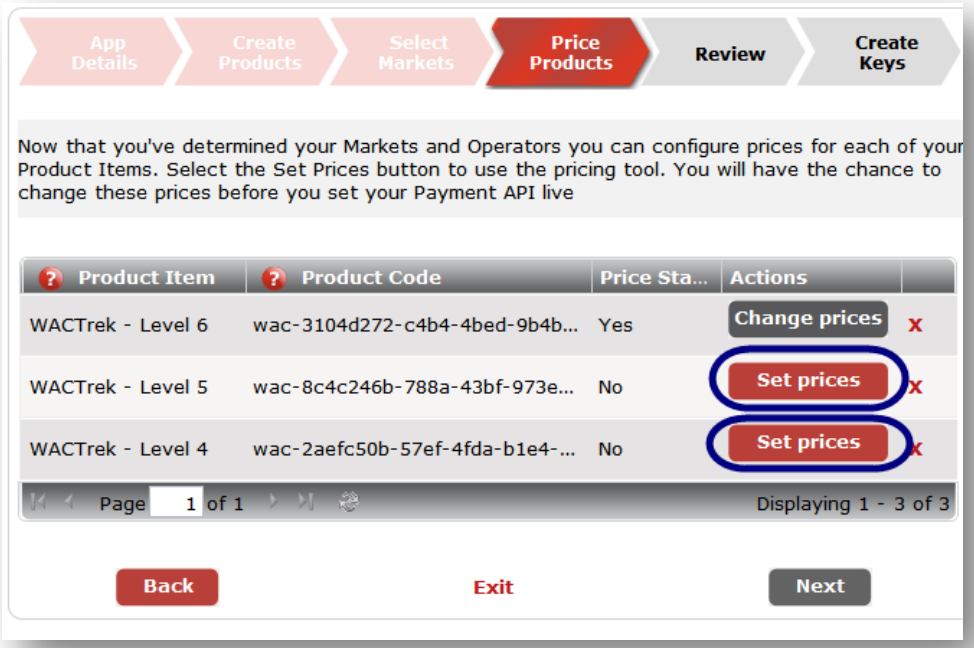
8) Enter the sales price of the in-app product.



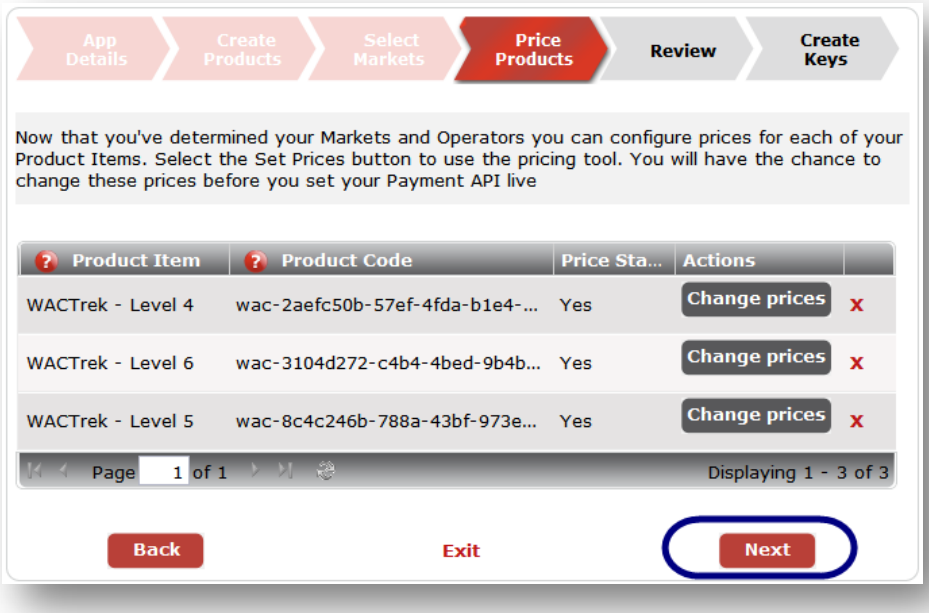
Step	Action
1	Enter your own currency and the item's desired price in this currency.
2	Click **Suggest Price**.
3	WAC suggests the equivalent in the currency of each market you selected. WAC calculates this suggestion by (a) converting your base currency to the operator's currency using daily exchange rates and then (b) rounding it up to the nearest whole value. **Note:** If the Pricing Type field is **Fixed**, you can only select one of the available price points, which have been defined by the operator. If the Pricing Type field is **Flexible**, you may enter any price point up to the maximum transaction limit. All price points for all operators can be found in the **Operator Pricing Guide** at https://www.wacapps.net/pricing. To enter a price other than the suggested price, click the **change link** and you will see will see a drop down menu that shows all the available price points. Choose one of these price points.  ***Note:*** *The mobile operator for each market determines whether these item prices include applicable value added or sales tax or other such taxes and how they are added to the customers total at checkout (see the WAC Operator Pricing Guide for all operator VAT/TAX models). In addition, local consumer laws may require disclosure of price variations. In the next step, this guide will provide an example of where and how to post this notice for customers.*
4	Click **Okay**.

9) You are returned to the **Price Products screen**.

Set the price for each remaining item and click **Next** when finished.

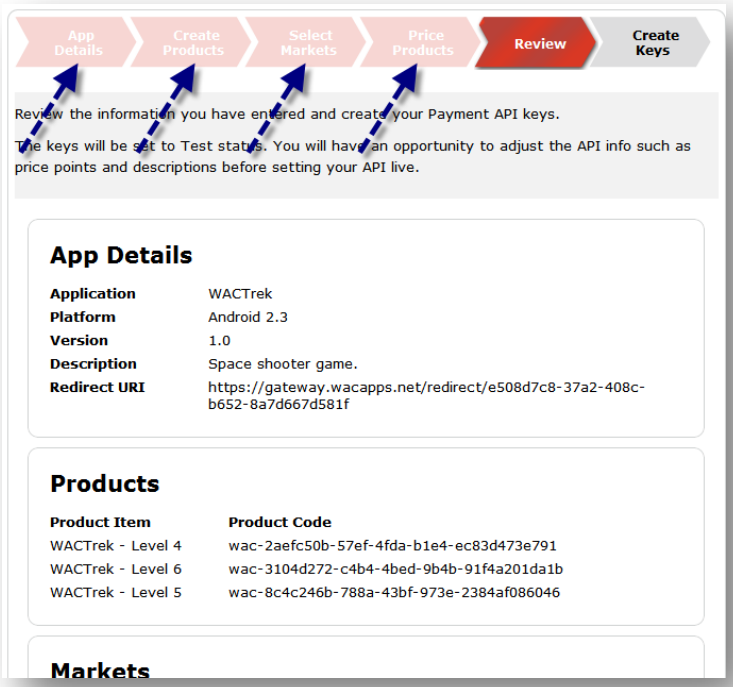


10) When finished setting prices, click **Next**.

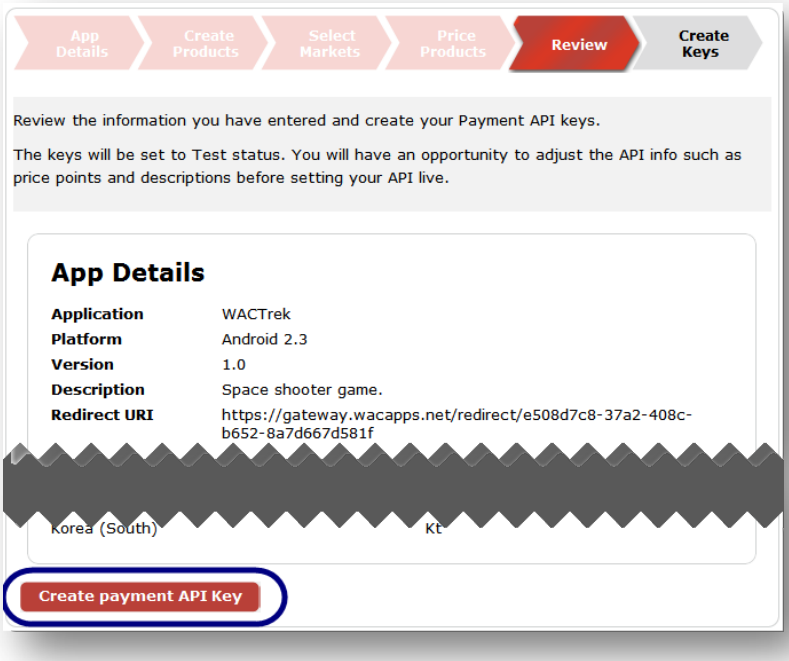


11) On the **Review screen**, verify (a) your app details are correct and (b) you have entered all products you currently know you want to sell in your app (you can return later to add more).

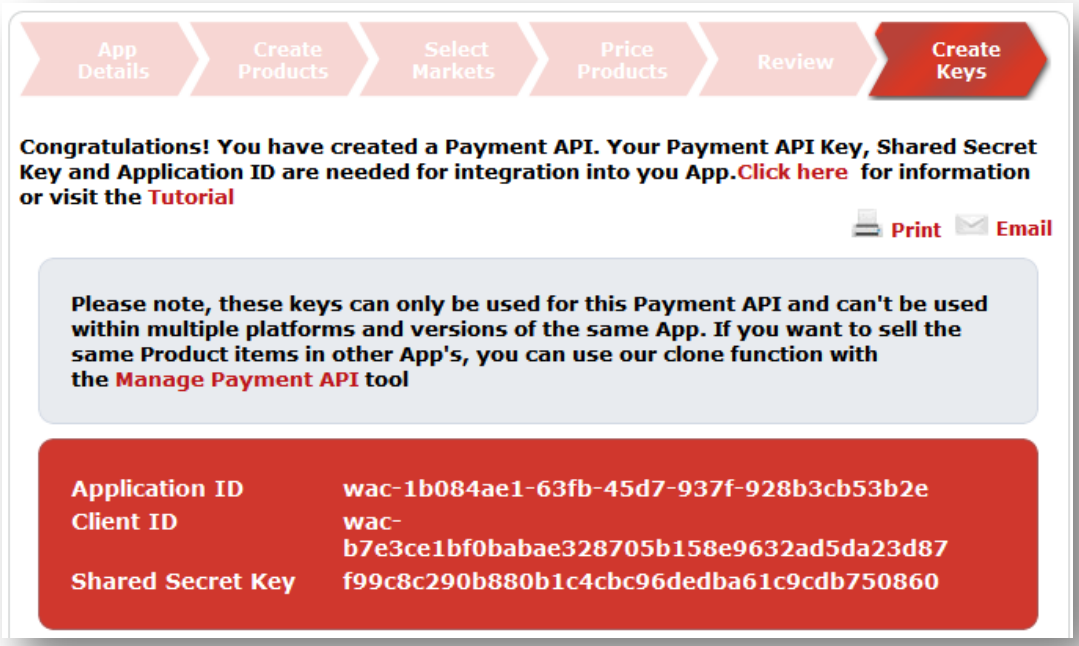
    If you need to make edits, click to open and edit the relevant page.



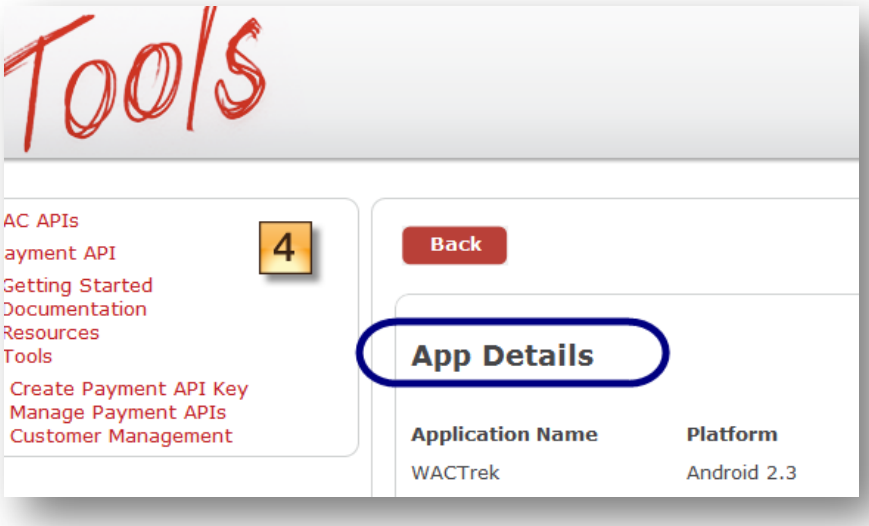
12) When you are satisfied that your app details and the products in it are correct, click **Create Payment API Key**.



13) Your API keys are created and provided on the **Create Keys screen**.



After your keys are created, you can see them any time on the **App Details screen**:



Step 4: Manage Your App

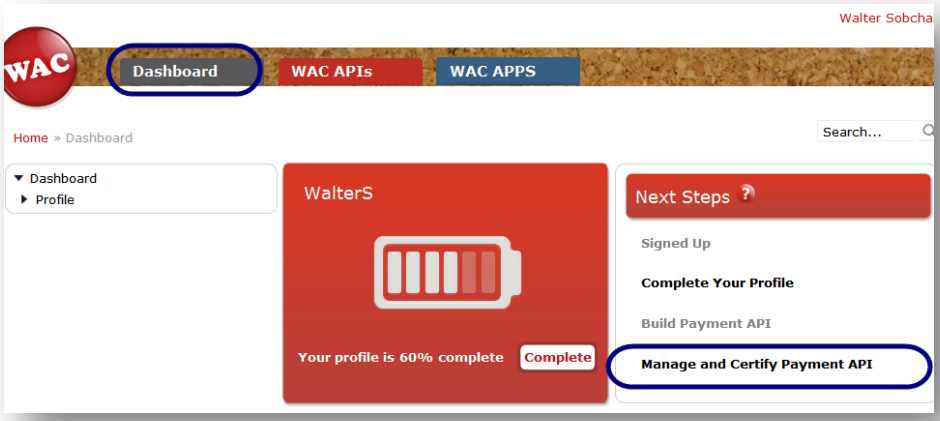
You are ready to add the product icons and descriptions your customers will select to purchase your app's products.

A) Add Icons & Descriptions

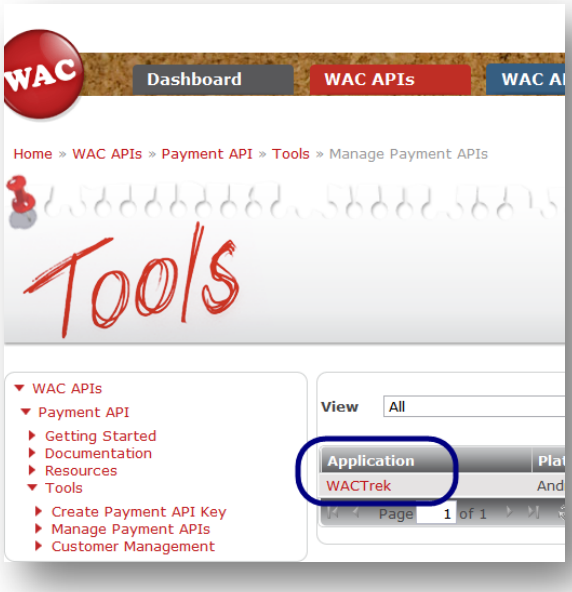
B) Localize Product Names for Each Market

A) Add Icons & Descriptions

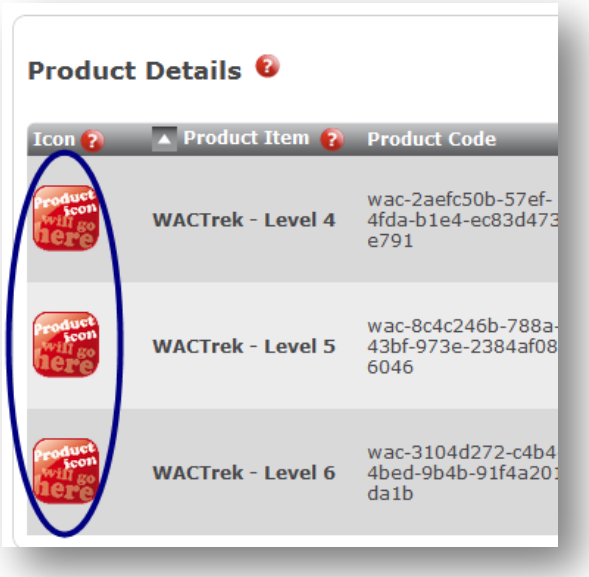
1) Click **Dashboard** and then **Manage and Certify Payment API**.



2) Open the app you just registered.



3) WAC provided default icons while you were defining your app's product items. However, you must replace these with your own icons before your app can be accepted for market.



*** The following are requirements and guidelines for the icons you upload to your in-app purchase items.**

WAC Icon Guidelines

Requirements

- **Size:** 200 x 200 pixels
- **Format:** PNG

An Easy-to-Manage Name

Make it easy to keep track if you're publishing to multiple markets.

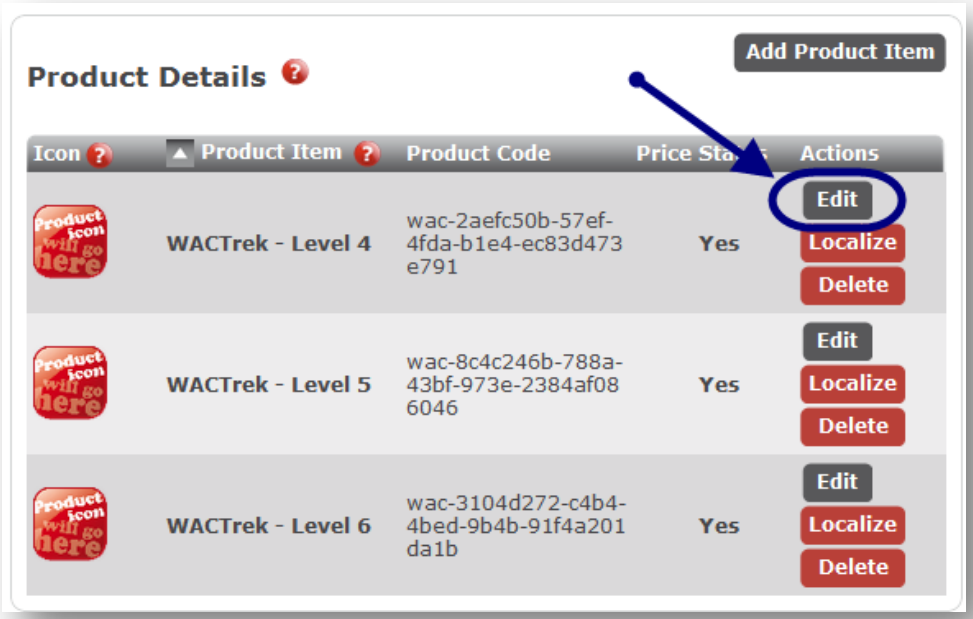
<app name>_<item name>_<market>.PNG

An Effective Design

Icons are small, so make them easy to understand -- simple and distinct gets noticed.

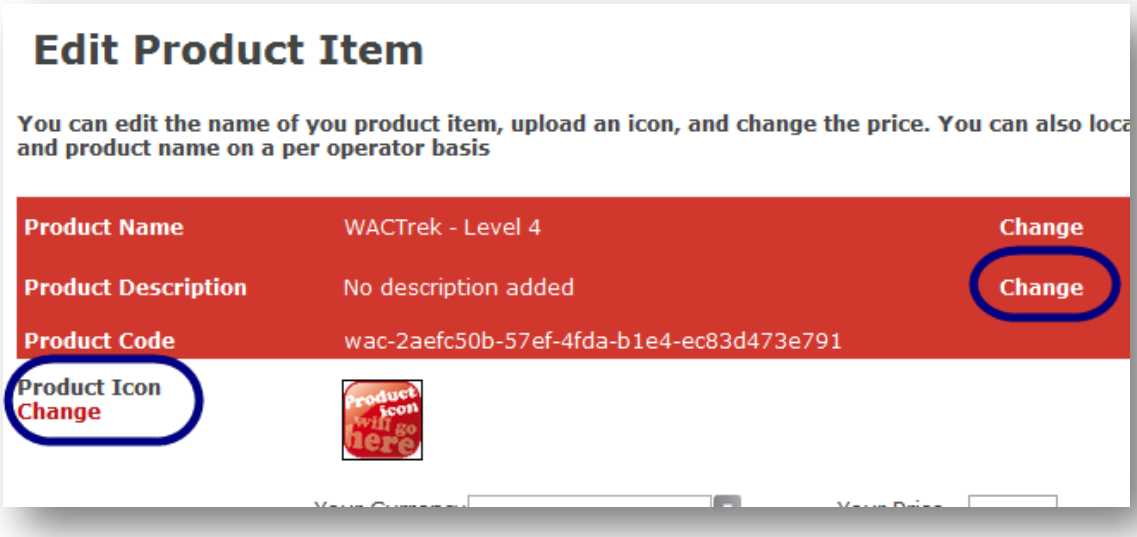
Effective Icons	Ineffective Icons
• Distinct centered image(s) • Distinct text (if any) • Distinct colors	• Cluttered images • Small or competing text • Colors that blend together

4) When you are ready to update your product icons, start by clicking edit for the first one.

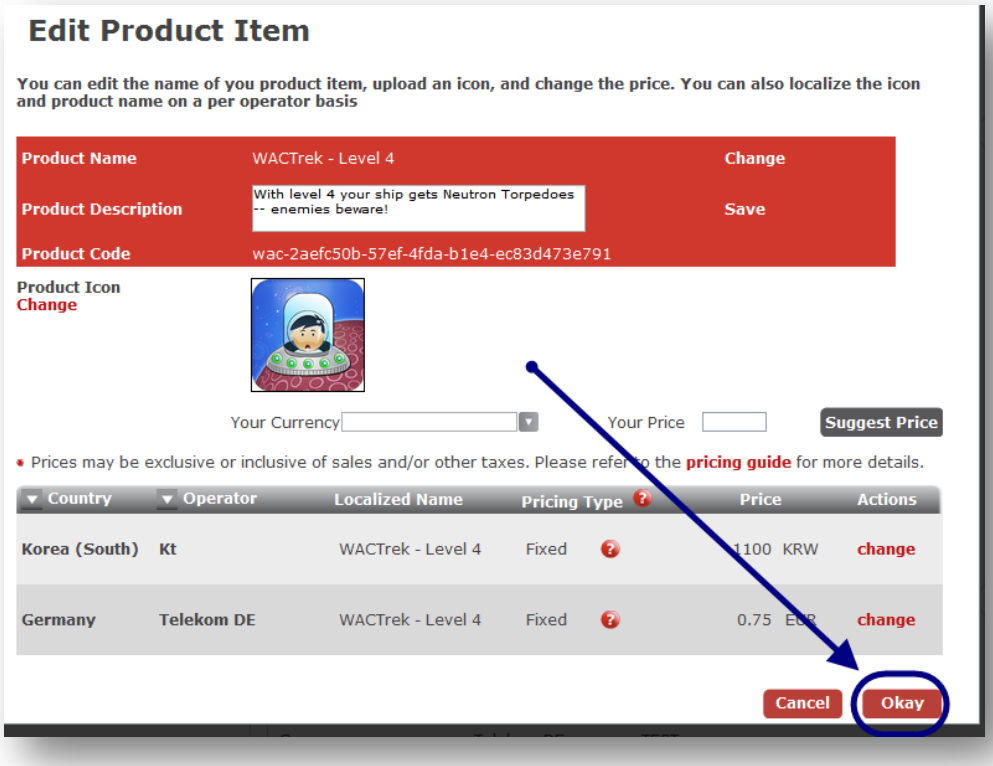


5) Use the **change links** to update this app product's description and icon.

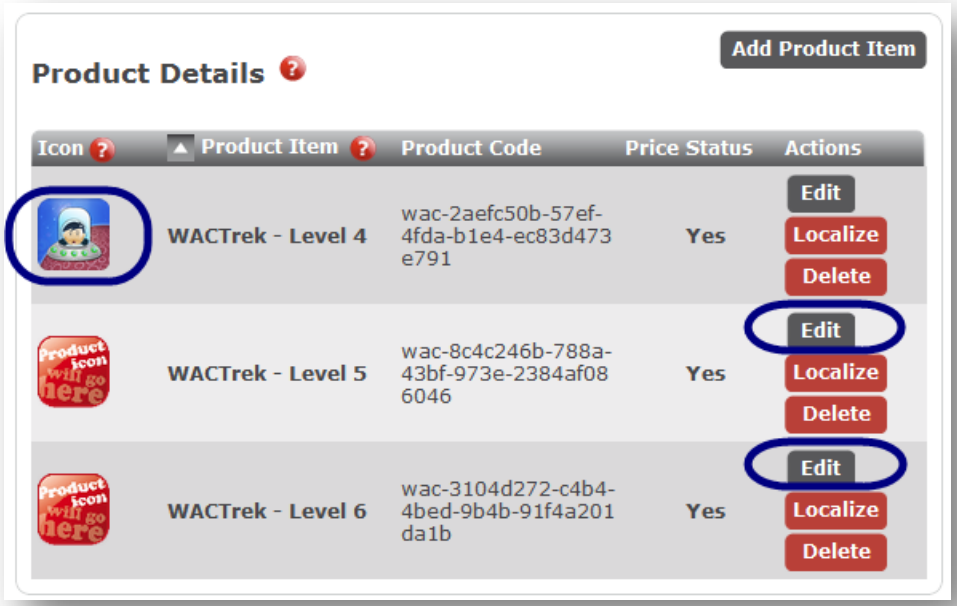
Note: The icon must be 200 x 200 pixels and in PNG format.



6) When ready, click **Okay**.



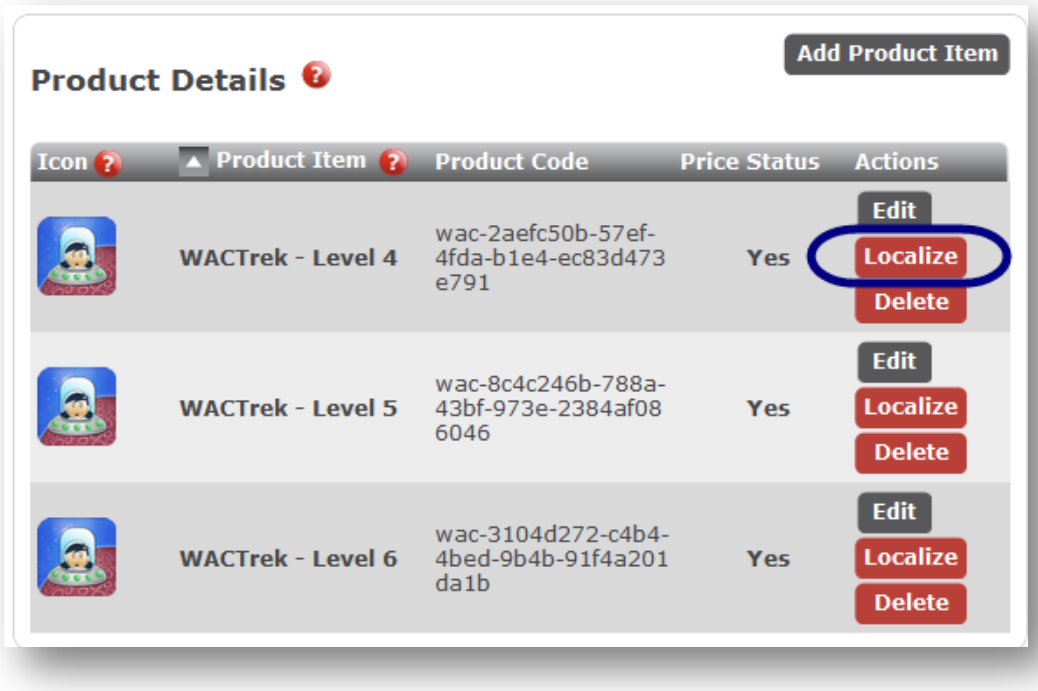
7) You see that the icon is updated. Update the rest of your product icons and descriptions.



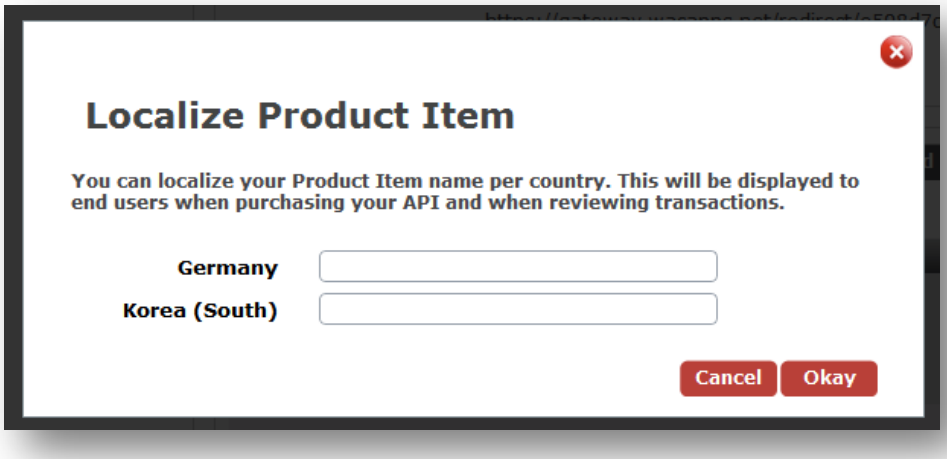
B) Localize Product Names for Each Market

You can elect to have each product's name appear in the language of the purchaser's market. This section shows how.

1) Click **Localize**.



2) Each market you chose previously appears on the screen. Enter the product name customers in those markets will see and click **Okay**. Then repeat for your other app products.



Step 5: Update Your App

Now that you have told WAC's payment gateway what to do when customers make purchases inside your app, it's time to add the actual WAC in-app payment functionality to your app.

This section explains how to update your app and illustrates each step by looking at the working code of the SDK sample app. (*Reminder: You installed and ran WAC's sample app in Step 1: Run the Sample App – if you skipped that step, go complete it now before continuing.)*

A) Enable In-App Payments in Your Code

B) Add Your "Pay By" Icons

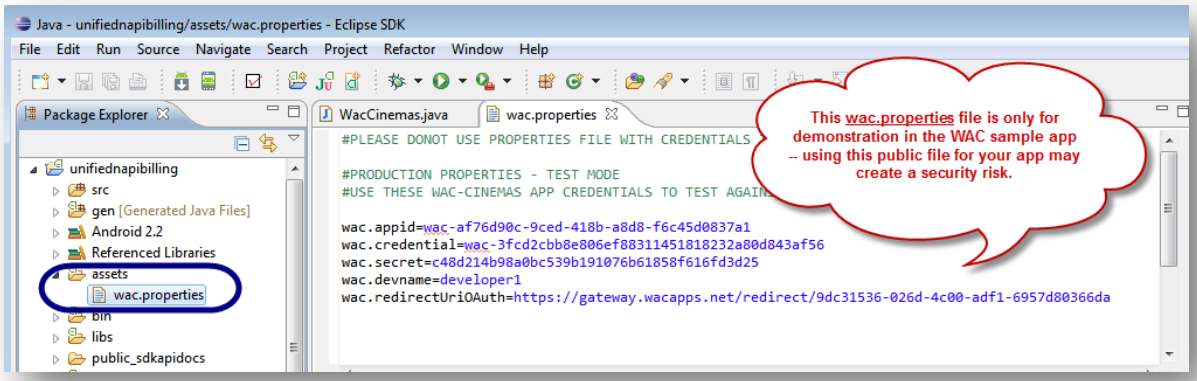
C) Add a Tax Notice for Your Customers

A) Enable In-App Payments in Your Code

1) Make certain that you have already completed Step 1: Run the Sample App.

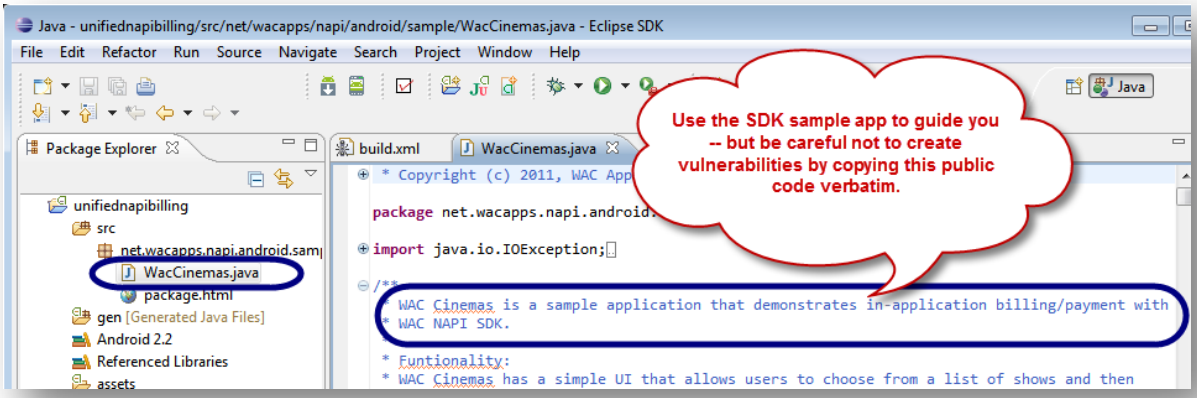
2) Update your application code with the API keys you created in **Step 3: Create Your API Keys**. **Reminder:** You can always find your WAC API keys on the **waccapps.net** App Details screen.

   Note: In your WAC SDK sample app, you can see that the app's API keys are used in the **assets/wac.properties** file. Be aware that this example is only provided to illustrate a method of storing your assigned API keys in your app. Do not use this same **wac.properties** file to deliver your API keys in your application – doing so would create a security risk because this approach is publicly documented here.



3) The following table summarizes the steps needed to get WAC billing functionality added to your app, followed by code samples of how these steps are implemented in the WACCinemas sample app included with your SDK.

Important: Remember that the WAC sample app is public code – exactly duplicating any part of its code in your app could create a security risk.



Action	Summary	How to Implement in Your App's Code
Initialize the payment service	Call **WacPaymentService**, which handles all payment processes and returns the transaction result.	1. Load the file that contains the developer registration information (in the sample app, this is the wac.properties file). 2. Create an AndroidWacPaymentService object when the Activity's onCreate() is invoked. 3. Initialize the object with the developer registration information (InitializeWACTask()). 4. Set this AndroidWacPaymentService object in the WacNapiContext.

<table>
<tr><th>Action</th><th>Summary</th><th>How to Implement in Your App's Code</th></tr>
<tr><td>Verify WAC billing is supported
(optional)</td><td>Call checkBillingAvailability to find out if WAC billing is available to the customer, which tells you whether to offer it as a payment option.

Note: This API is optional and is not currently included in the WacCinemas sample app. If you elect to perform this check, do so as explained here.</td><td>If the variable returns TRUE, WAC billing is supported; if false, WAC billing is not supported:

<code>WacNapiContext.getInstance().setEndPoints(new WacEndpoints(WacEndpoints.PRODUCTION));</code>

<code>AndroidWacPaymentService mService = new AndroidWacPaymentService();
WacNapiContext.getInstance().setPaymentService(mService);</code>

<code>boolean check = mService.checkBillingAvailability(mProps.getProperty("wac.appid"),</code>

<code>mProps.getProperty("wac.credential"),
mProps.getProperty("wac.secret"),
mProps.getProperty("wac.devname"));</code>

Note: Because the customer will not be identified this early in the WAC transaction, be aware of these possible outcomes and their causes:

<table>
<tr><th>Result</th><th>Cause</th></tr>
<tr><td>True</td><td>• App is LIVE, published to this operator, and individual user can use WAC</td></tr>
<tr><td>True</td><td>• Mobile device is on Wi-Fi and
• App published in the country of the access point IP address but not the operator of the user</td></tr>
<tr><td>False</td><td>• Mobile device is on operator network and
• App not LIVE or not published to this operator</td></tr>
<tr><td>False</td><td>• Mobile device is on Wi-Fi and
• App not LIVE or not published in the country of the access point IP address</td></tr>
<tr><td>*True</td><td>• WAC billing is technically available however the particular user is not accepted for this transaction (user is blacklisted, lacks adequate credit, etc.)

*Note: Operators do not yet provide customer-level access for this method, which results in a TRUE result. This will be addressed in a future release.</td></tr>
</table>
</td></tr>
</table>

Action	Summary	How to Implement in Your App's Code
Make payment with WAC	**AndroidWacPaymentService** provides Reserve & Capture for making payments. **Notes:** • By WAC-operator agreement, goods must be delivered to customers before customers are charged.	**Reserve & Capture** 1. During authorization, initiate the reserve transaction to reserve the funds with the operator: Call reservePayment() on AndroidWacPaymentService - (See onClick(View)). 2. After the good is delivered, get the reserved payment transaction and use it to capture the payment by invoking capturePayment (see onActivityResult(int, int, Intent)). **Notes:** • Reserved funds are released if capturePayment is not initiated within 5 minutes. • If good delivery fails, not calling capturePayment is sufficient to cancel the transaction. • Customers will be notified of cancelled transactions via SMS text message.
Track Transactions	User transactions are stored securely locally and are retrieved in these steps. **Note:** In the current release of the SDK, only transactions stored on the phone can be retrieved. This means that if the user clears the cache or re-installs the app, transaction tracking may not return complete results. WAC is currently working on server transaction retrieval.	• **Store Transactions:** Initialize the secure data store, DeviceTransactionStorage (See - getTransactionStorage()). • **List Transactions:** Read transaction information from the secure data store and load into local variables - (See setupWidgets()). • **Check Transactions:** Invoke checkTransaction() on the AndroidWacPaymentService - (See setupWidgets()).

Import Needed Objects

Import these payment objects -- this code details all the potential objects you might need.

```
import net.wacapps.napi.android.AndroidWacPaymentService;
import net.wacapps.napi.android.WacNapiContext;
import net.wacapps.napi.android.WacNapiPayment;
import net.wacapps.napi.api.NapiException;
import net.wacapps.napi.api.WacEndpoints;
import net.wacapps.napi.resource.jaxb.AmountTransaction;
import net.wacapps.napi.resource.jaxb.Item;
import net.wacapps.napi.resource.jaxb.ReservedTransaction;
import net.wacapps.napi.resource.jaxb.Transaction;
import net.wacapps.napi.resource.jaxb.TransactionList;
import net.wacapps.napi.util.crypto.DeviceTransactionStorage;
import net.wacapps.napi.util.crypto.SecureDeviceTransactionDatabase;
```

Initializing the payment service

Note: Before initializing the payment service, you can elect to have the app verify that WAC billing is available to the customer so you know whether to offer a WAC payment option to the customer. This option is not coded in the sample app but is explained in the table above.

```
/**
         * Called when the activity is first created.
         *
         * @param savedInstanceState
         *            the saved instance state
         */
        @Override
        public void onCreate(Bundle savedInstanceState) {
               super.onCreate(savedInstanceState);
               setContentView(R.layout.main);

               mHandler = new Handler();

               // Read from the assets directory
               if (mProps == null) {
                      try {
                             InputStream inputStream =
this.getResources().getAssets()
                                                    .open("wac.properties");
                             mProps = new Properties();
                             mProps.load(inputStream);
```

```
                        if (DEBUG)
                              Log.d(TAG, "The properties are now
loaded: " + mProps);
                  } catch (IOException e) {
                        System.err.println("Failed to open wac
property file");
                        e.printStackTrace();
                  }
            }

            // Initialize WAC NAPI API
            if (WacNapiContext.getInstance().getPaymentService()
== null) {
                  WacNapiContext.getInstance().setEndPoints(
                              new
WacEndpoints(WacEndpoints.PRODUCTION));
                  AndroidWacPaymentService.setDebug(DEBUG);

                  mService = new AndroidWacPaymentService();

      WacNapiContext.getInstance().setPaymentService(mService);
                  // Load available product items
                  this.pd = ProgressDialog.show(this, "Working..",
                              "Loading available Items...",
true, false);
                  new InitializeWACTask().execute();
            } else {
                  mService =
WacNapiContext.getInstance().getPaymentService();
                  setupWidgets();
            }
      }

    /**
       * The Class InitializeWACTask is an AsyncTask to perform
background operations for initializing the android WAC payment
service.
       */
      private class InitializeWACTask extends AsyncTask<String,
Void, Object> {

            protected Object doInBackground(String... args) {
                  Log.d(TAG, "Background thread for initialize
starting");
                  try {
```

```
            mService.initialize(mProps.getProperty(WAC_APPID_KEY),

            mProps.getProperty("wac.credential"),

            mProps.getProperty("wac.secret"),

            mProps.getProperty("wac.devname"),

            mProps.getProperty("wac.redirectUriOAuth"));
                        } catch (Exception e) {

                        // This exception indicates that WAC billing is
                        // not supported. If you want to use a specific
                        // API to check for WAC billing support, use the
                        // checkBillingAvailability API.

                              e.printStackTrace();
                              final Bundle b = new Bundle();
                              b.putString("failMessage",
                                          e.getMessage());
                              mHandler.post(new Runnable() {
                                    public void run() {

            showDialog(DIALOG_CHARGE_PAYMENT_FAIL, b);
                                    }
                              });
                              return null;
                        }
                        return mService;
                  }

                  protected void onPostExecute(Object result) {
                        // dismiss progress dialogs
                        if (WacCinemas.this.pd != null) {
                              WacCinemas.this.pd.dismiss();
                        }
                        // quick sanity check
                        if (result != null) {
                              // add items to list adapter
                              List<Item> items =
      mService.listProductItems();
                              for (Item item : items) {
                                    catalog.add(new
      CatalogEntry(item.getItemId(), item
```

```
                                                .getDescription(),
      item));
                                }
                                // create UI components in the UI thread
                                mHandler.post(new Runnable() {
                                      public void run() {
                                            setupWidgets();
                                      }
                                });
                          } else {
                                Log.d(TAG, "Service setup failure!!!");
                          }
                    }
      }
```

** OAuth copyright 2007 Andy Smith.*

Creating payment intent and starting activity

```
            public void onClick(View v) {
                  if (v == mBuyButton) {
                        if (DEBUG) {
                              Log.d(TAG, "buying: " + mItemName + "
      sku: " + mSku);
                        }

                        String refCode =
      UUID.randomUUID().toString().substring(0, 32);

                        //use reserve and capture

            WacNapiContext.getInstance().getPaymentService().reservePayme
      nt(this, mProps.getProperty(WAC_APPID_KEY), mSku, refCode);

                  } else if (v == mTxListButton) {
                        if (DEBUG) {
                              Log.d(TAG, "Getting Tx List");
                        }
                        WacNapiContext.getInstance()

            .getPaymentService().getTransactionList(this);
                  }
            }
```

Processing Charge Payment Transaction

```
	/**
	 * Called when an activity called by using
startActivityForResult finishes.
	 *
	 * @param requestCode
	 *            the request code
	 * @param resultCode
	 *            the result code
	 * @param data
	 *            the data
	 */
	@Override
	public void onActivityResult(int requestCode, int resultCode,
Intent data) {
		Log.d(TAG, "onActivityResult called");
		final Bundle b = new Bundle();
		switch (resultCode) {
		case WacNapiPayment.RESULT_RESERVE_PAYMENT_OK:
			Log.d(TAG, "onActivityResult called +
RESULT_RESERVE_PAYMENT_OK");
			ReservedTransaction reserve =
mService.processReservePaymentResults(data);
			Log.d(TAG, "got reserve with token " +
reserve.getAccessToken());

			//Add code here to fulfil the purchase

			try {
				mService.capturePayment(this, reserve);
//callback will come in RESULT_PAYMENT_OK
			} catch (NapiException e1) {
				e1.printStackTrace();
				b.putString("failMessage",
						"Capture payment failed!");
				mHandler.post(new Runnable() {
					public void run() {

	showDialog(DIALOG_CHARGE_PAYMENT_FAIL, b);
					}
				});
			}
			Log.d(TAG, "Capture payment request sent");
			break;
```

```
                case WacNapiPayment.RESULT_CHECK_TRANSACTION_OK:
                        Log.d(TAG, "onActivityResult called +
RESULT_CHECK_TRANSACTION_OK");
                        try {
                                final Transaction checkTransaction =
mService

        .processTransactionCheckResults(data);
                                if(checkTransaction != null &&
checkTransaction.getAmountTransaction().getServerReferenceCode() !=
null) {
                                        if(DEBUG)
                                                Log.d(TAG, "Check
Transaction Success!!! " +

        checkTransaction.getAmountTransaction().getServerReferenceCod
e());

        showDialog(DIALOG_TRANSACTIONS_CHECK_SUCCESS);
                                } else {
                                        b.putString("failMessage",
                                                        "Invalid
transaction!");
                                        mHandler.post(new Runnable() {
                                                public void run() {

        showDialog(DIALOG_CHARGE_PAYMENT_FAIL, b);
                                                }
                                        });
                                }
                        } catch (Exception e) {
                                e.printStackTrace();
                                b.putString("failMessage",
                                                "Invaid transaction!");
                                mHandler.post(new Runnable() {
                                        public void run() {

        showDialog(DIALOG_CHARGE_PAYMENT_FAIL, b);
                                        }
                                });
                        }
                        break;
                case WacNapiPayment.RESULT_PAYMENT_OK:
                        Log.d(TAG, "onActivityResult called +
RESULT_PAYMENT_OK");
                        final Transaction transaction = mService
```

```
            .processChargePaymentTransactionResults(data);
                        this.runOnUiThread(new Runnable() {
                              public void run() {
                                    HashMap<String, String> temp = new
      HashMap<String, String>();
                                    temp.put(FIRST_COLUMN,
      transaction.getAmountTransaction().getPaymentAmount().getChargingInf
      ormation().getDescription());
                                    temp.put(SECOND_COLUMN,
      transaction.getAmountTransaction()

            .getPaymentAmount().getChargingInformation()

            .getAmount().toString());
                                    mOwnedItems.add(temp);

            mOwnedItemsAdaptor.notifyDataSetChanged();

            mOwnedItemsTracker.add(transaction);
                              }
                        });
                        break;
                  case WacNapiPayment.RESULT_TRANSACTION_LIST_OK:
                        TransactionList list =
      mService.processTransactionListResults(data);
                        int tSize =
      list.getPaymentTransactionList().getAmountTransaction()
                                    .size();
                        Log.d(TAG, "TxList size " + tSize);
                        b.putInt("tSize", tSize);
                        mHandler.post(new Runnable() {
                              public void run() {

            showDialog(DIALOG_TRANSACTIONS_LIST_SUCCESS, b);
                              }
                        });
                        break;
                  case WacNapiPayment.RESULT_OPERATOR_BAD:
                        Log.d(TAG, "onActivityResult called +
      RESULT_OPERATOR_BAD");
                        processError(data);
                        break;
                  case WacNapiPayment.RESULT_PAYMENT_BAD:
                        Log.d(TAG, "onActivityResult called +
      RESULT_PAYMENT_BAD");
```

```
                        processError(data);
                        break;
                case WacNapiPayment.RESULT_TRANSACTION_LIST_BAD:
                        Log.d(TAG, "onActivityResult called +
RESULT_TRANSACTION_LIST_BAD");
                        processError(data);
                        break;
                default:
                        break;
                }
        }
```

Getting database handle for DeviceTransactionStorage

```
        /**
                * Gets transaction database handle
                */
               private DeviceTransactionStorage getTransactionStorage(){
                      return new
        SecureDeviceTransactionDatabase(getApplicationContext(),
        mProps.getProperty(WAC_APPID_KEY));
               }
```

Fetching transaction data from DeviceTransactionStorage

```
        /**
                * Sets up the UI.
                */
               private void setupWidgets() {


                      mBuyButton = (Button) findViewById(R.id.buy_button);
                      mBuyButton.setOnClickListener(this);


                      mTxListButton = (Button)
        findViewById(R.id.tx_list_button);
                      mTxListButton.setOnClickListener(this);


                      mSelectItemSpinner = (Spinner)
        findViewById(R.id.item_choices);
                      mCatalogAdapter = new CatalogAdapter(this, catalog);
                      mSelectItemSpinner.setAdapter(mCatalogAdapter);
                      mSelectItemSpinner.setOnItemSelectedListener(this);
```

```
                mOwnedItemsTable = (ListView)
      findViewById(R.id.owned_items);
                mOwnedItemsAdaptor = new ListViewAdapter(this,
      mOwnedItems);
                mOwnedItemsTable.setAdapter(mOwnedItemsAdaptor);
                final Activity ctx = this;
                mOwnedItemsTable.setOnItemClickListener(new
      OnItemClickListener() {
                      @Override
                      public void onItemClick(AdapterView<?> a, View
      v, int position,
                                  long id) {
                            AlertDialog.Builder adb = new
      AlertDialog.Builder(ctx);
                            adb.setTitle("Transaction Check?");
                            final Transaction t =
      mOwnedItemsTracker.get(position);
                            adb.setMessage("Validate transaction with
      ref code " + t.getAmountTransaction().getServerReferenceCode() + "
      ?");
                            adb.setPositiveButton("Ok", new
      DialogInterface.OnClickListener() {
                                  @Override
                                  public void
      onClick(DialogInterface dialog, int which) {
                                        Log.d(TAG, "Running check
      transactions!");

            mService.checkTransaction(ctx,
      t.getAmountTransaction().getServerReferenceCode());
                                  }
                            });
                            adb.setNegativeButton("Cancel", null);
                            adb.show();
                      }
                });

                DeviceTransactionStorage pdb = null;
                try {
                      pdb = getTransactionStorage();
                      //Fetch the transaction list from local storage.
                      TransactionList list = pdb.getTransactionList();
                      pdb.close();
                      for (AmountTransaction transaction : list

            .getPaymentTransactionList().getAmountTransaction()) {
                            HashMap<String, String> temp = new
```

```
HashMap<String, String>();
                        temp.put(FIRST_COLUMN,
transaction.getPaymentAmount().getChargingInformation().getDescripti
on());
                        String amount =
transaction.getPaymentAmount().getChargingInformation().getAmount().
toString();
                        temp.put(SECOND_COLUMN, amount);
                        mOwnedItems.add(temp);

      mOwnedItemsAdaptor.notifyDataSetChanged();
                        Transaction tx = new Transaction();
                        tx.setAmountTransaction(transaction);
                        mOwnedItemsTracker.add(tx);
                  }
            } catch (Exception ee) {
                  ee.printStackTrace();
            } finally {
                  pdb.close();
            }
      }                       });
                        adb.setNegativeButton("Cancel", null);
                        adb.show();
                  }
            });

            DeviceTransactionStorage pdb = null;
            try {
                  pdb = getTransactionStorage();
                  TransactionList list = pdb.getTransactionList();
                  pdb.close();
                  for (AmountTransaction transaction : list

      .getPaymentTransactionList().getAmountTransaction()) {
                        HashMap<String, String> temp = new
HashMap<String, String>();
                        temp.put(FIRST_COLUMN,
transaction.getPaymentAmount().getChargingInformation().getDescripti
on());
                        String amount =
transaction.getPaymentAmount().getChargingInformation().getAmount().
toString();
                        temp.put(SECOND_COLUMN, amount);
                        mOwnedItems.add(temp);

      mOwnedItemsAdaptor.notifyDataSetChanged();
```

```
                                Transaction tx = new Transaction();
                                tx.setAmountTransaction(transaction);
                                mOwnedItemsTracker.add(tx);
                        }
                } catch (Exception ee) {
                        ee.printStackTrace();
                } finally {
                        pdb.close();
                }
        }
```

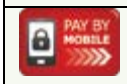
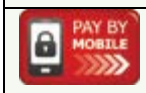
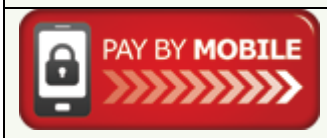
B) Add Your "Pay By" Icons

For the screen in your app where your customers choose how they want to pay, WAC provides a variety of icons you can use for the WAC payment option.

Note: The WAC billing option button is only needed if there are multiple billing choices available to the customer. When WAC billing is the only option, your app can simply take the customer directly to the purchase confirmation screen.

There are two ways to use WAC's payment buttons in your app:

- **Embed a link to the graphic in you app:** If you embed a link to the button graphic (links provided in the table below) and WAC's server has a button available in the IP address' local language, then this button will automatically appear localized to the purchaser. WAC recommends this method because a localized payment button helps inspire trust for customers in non-English speaking countries.
- **Embed the graphic directly in your app:** If you embed WAC's payment button directly in your app it will *not* automatically match to the customer's local market. You can download graphics from the links in the table below. You can also download buttons all languages available at https://www.wacapps.net/pay-by-mobile-icon.

Image	URL
	http://icon.wacapps.net/icon/37w-x-23h-px.png
	http://icon.wacapps.net/icon/50w-x-34h-px.png
	http://icon.wacapps.net/icon/60w-x-38h-px.png
	http://icon.wacapps.net/icon/120w-x-30h-px.png
	http://icon.wacapps.net/icon/150w-x-40h-px.png
	http://icon.wacapps.net/icon/150w-x-60h-px.png
	http://icon.wacapps.net/icon/180w-x-113h-px.png
	http://icon.wacapps.net/icon/reversed-37w-x-23h-px.png
	http://icon.wacapps.net/icon/reversed-50w-x-34h-px.png
	http://icon.wacapps.net/icon/reversed-60w-x-38h-px.png
	http://icon.wacapps.net/icon/reversed-120w-x-30h-px.png
	http://icon.wacapps.net/icon/reversed-150w-x-40h-px.png
	http://icon.wacapps.net/icon/reversed-150w-x-60h-px.png
	http://icon.wacapps.net/icon/reversed-180w-x-113h-px.png

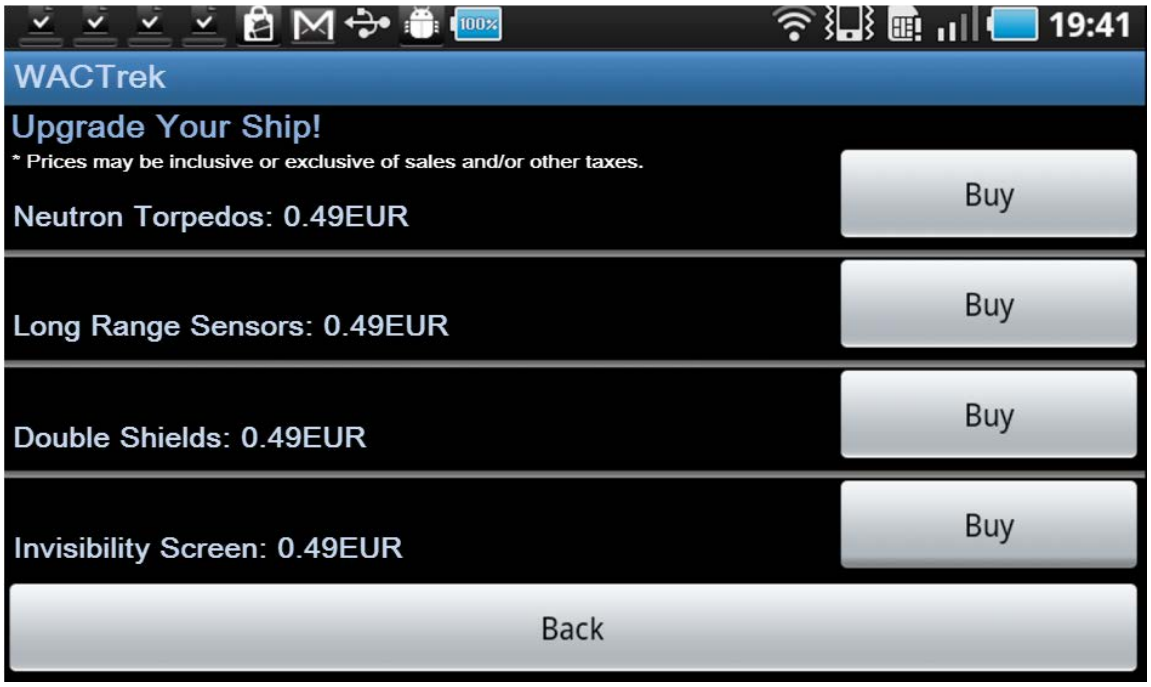
C) Add a Tax Notice for Your Customers

The mobile operator for each market determines whether item prices include applicable value added or sales tax or other such taxes and how they are added to the customers total at checkout. In addition, local consumer laws may require disclosure of price variations.

You are responsible for the wording and making sure that it is clear to consumers whether there is a price change and how taxes are added. Any such wording should appear clearly and concisely on the purchase screen.

Example for Initial Purchase Screen

Please note that the price of item, and the applicable VAT or sales tax, may vary depending on your mobile operator and/or the country in which you made your purchase.



Step 6: Test Your App

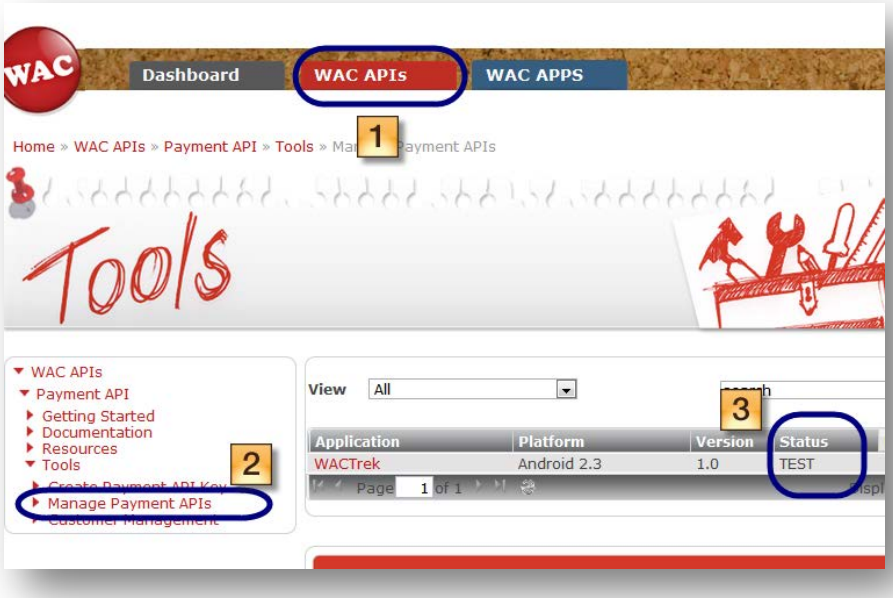
Now comes the fun part – buy some of your app's products and make sure it all works.

A) Verify In-App Payments Are Working

B) Test Prices in Different Regions

A) Verify In-App Payments Are Working

After you create your API keys, our app status updates to **TEST**, indicting you are ready to practice some test purchases and see how they go. At this stage all purchases are only simulations, so test all you want and no money will be transacted.



Tip: App Status Definitions

Status	Description
TEST	You have created your WAC API keys (Step 3: Create Your API Keys). During **TEST** status, in-app purchases occur only in WAC's testing "sandbox" – no money is transacted.
PROCESSING	You have requested that WAC certify your app (Step 7: Certify Your App). While WAC processes your certification, no updates are permitted to your app or your WAC account.
ACCEPTED	WAC has concluded certification and accepted your app to continue on to publishing (Step 8: Set Your API Keys Live).
LIVE	You have set your app's API keys live in at least one market, after which all in-app purchases from your app's WAC API keys in the market will be *real* monetary transactions.

Important Notes

- You can test WAC billing in your app against any operator by using a mobile device connected by Wi-Fi an Android emulator on PC. However, you can only test WAC billing on mobile devices connected by operator SIM if the operator is running on WAC's latest API build. For the most current list of these operators, see here: http://www.wacapps.net/live-operators.

  To test your app with an operator-SIM-connected mobile device whose operator has not yet migrated to WAC's latest platform, WAC recommends that you (a) perform the tests in this section via Wi-Fi and/or Android emulator and then (b) test again with the mobile device SIM-connected to the operator after your status is LIVE (see Step 9: Push App to Markets). If you have any questions, please contact us at support@wacapps.net.

- While WAC Operations and Support works hard to keep the Developer Sandbox available 24 hours a day and 7 days a week, occasional system downtime may occur. To check the real-time status of the sandbox at any time, click here.

- Before continuing, ensure that the device you run your app on – mobile phone or the computer your run the Android emulator from – is set to your correct local time. WAC's payment gateway will have your app report a 'failed-signature validation' error if it receives calls from a system whose time varies more than 3 minutes from your local time.
- Transactions and their downloads must complete within 5 minutes or reserved customer funds will be released – test to be certain your app's transactions can complete and initiate the capture call before this time limit.

<table>
<tr><th>Platform</th><th>Connection Type</th><th>Description</th></tr>
<tr><td>Phone</td><td>• SIM from enabled operator, and
• Wi-Fi off</td><td>1. Because the SIM connection identifies you as a customer of the operator, you can complete the transaction successfully. Purchases appear on screen under Items you own. (Note: This test is only available for operators updated to WAC's latest API build.)
2. Customers using the live app will receive SMS confirmation of the purchase (however during testing SMS messages are not sent).</td></tr>
<tr><td>Phone</td><td>• SIM from another operator, or
• No SIM, or
• Wi-Fi only</td><td>1. App reports it cannot detect your phone number with the market operator.
2. App prompts for your phone number.
3. WAC's test environment lets you simulate different scenarios by entering these phone numbers:
<table>
<tr><th>Scenario</th><th>What to Enter</th><th>Message</th></tr>
<tr><td>Successful payment</td><td>+19724891737</td><td>Payment successful.</td></tr>
<tr><td>Spend limit error</td><td>+19724891648</td><td>You have reached your operator spend limit. Please contact your operator for help.</td></tr>
<tr><td>Payment failed</td><td>+19724892325</td><td>Payment failed, please try again later. You will not be charged for this transaction.</td></tr>
<tr><td>Operator not supported</td><td>Any operator number not live on WAC</td><td>Operator not supported.</td></tr>
</table>
4. App prompts you for PIN you received by SMS, which customers will receive for live apps. In testing, no PIN is sent. Instead, enter PIN 4681.
5. Results match scenario. Successful payments are followed by purchased items appearing on screen under Items you own.
6. Customers using the live app will receive SMS confirmation of the purchase, however these are not sent during your testing.</td></tr>
<tr><td>Emulator</td><td>PC connection</td><td>Same as above.</td></tr>
</table>

B) Test Prices in Different Regions

You can view your in-app prices under different regional operators by "spoofing," imitating, different regional IPs:

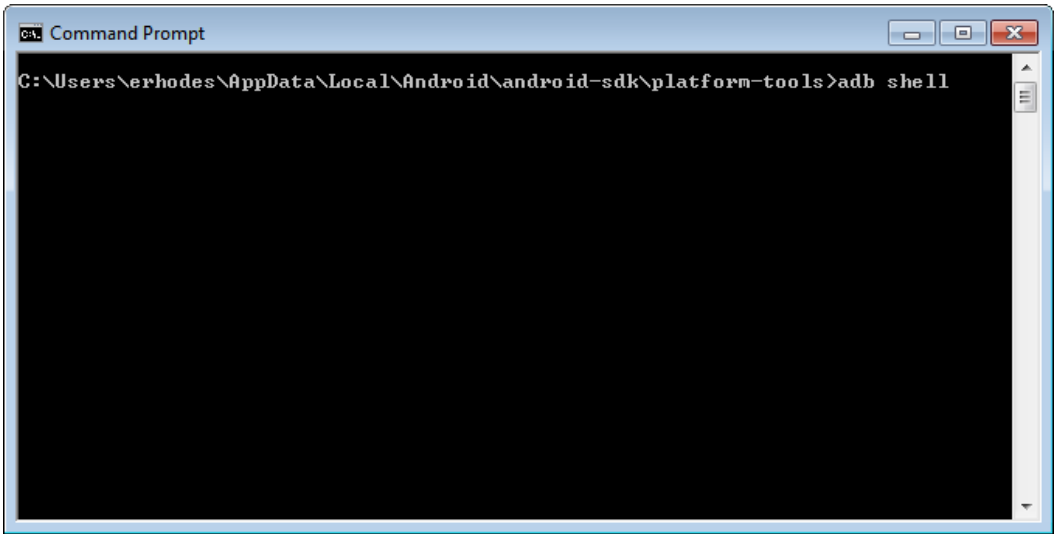
Region	Operator(s)	IP
US	ATT	12.207.19.228
Germany	DT, Telefonica	80.187.110.132
Spain	Telefonica	79.146.82.130
Bulgaria	TAG	31.211.128.0
Korea	KT, SKT, LGU+	61.47.192.22
UK	Vodafone	192.165.213.18
Philippines	SMART	120.28.64.69

To spoof a regional IP, do the following:

1) Start the ADB shell:
   In your **<Android SDK install folder>/platform-tools** folder, enter **adb shell**.

Windows Example

Note: *Your Android SDK install folder will be different.*



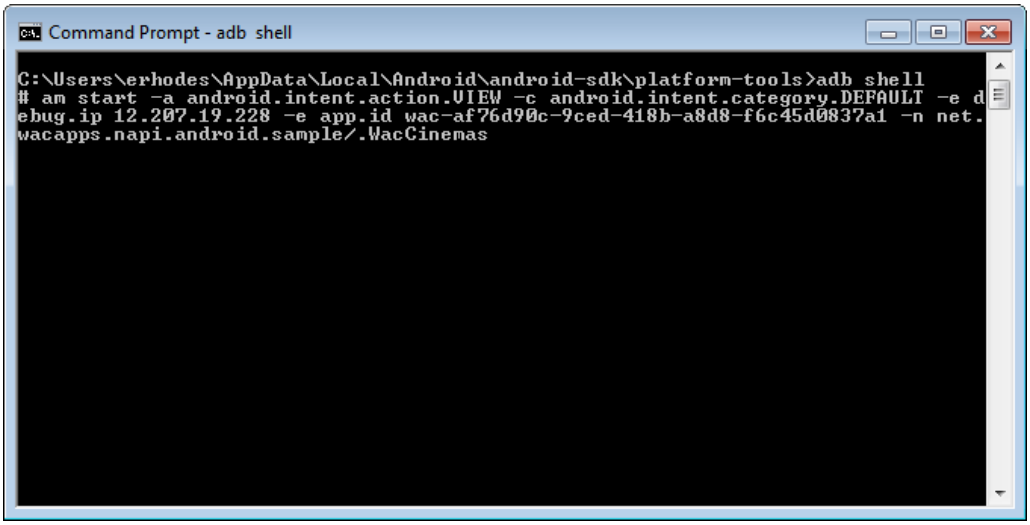
2) Enter the following command with the IP of the region you want to spoof and your WAC APP ID.

Important: <your app id> refers to the unique WAC app ID explained here.

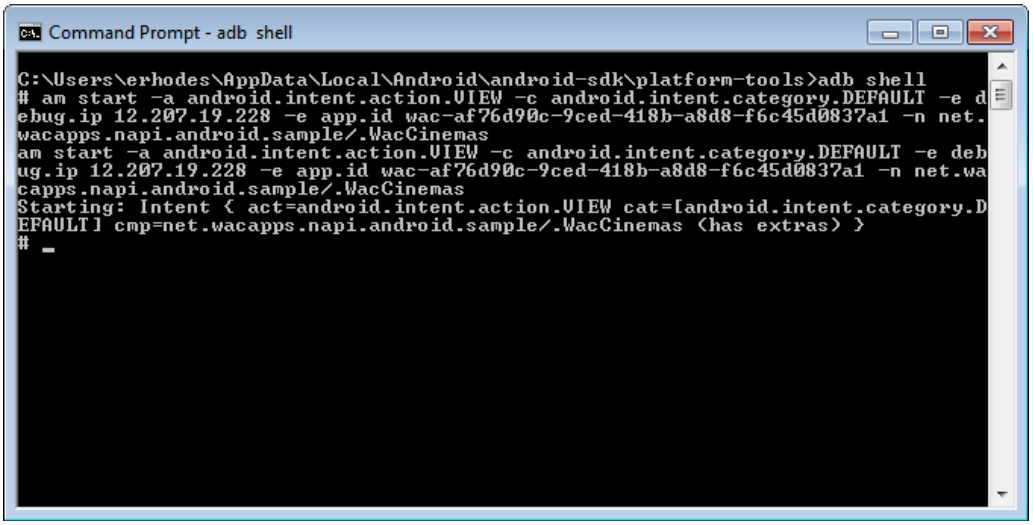
OS	Command
Windows Example	`am start -a android.intent.action.VIEW -c android.intent.category.DEFAULT -e debug.ip <IP you want to imitate> -e <your app.id from WAC> -n net.wacapps.napi.android.sample/.WacCinemas`
OSX / UNIX Example	`./am start -a android.intent.action.VIEW -c android.intent.category.DEFAULT -e debug.ip <IP you want to imitate> -e <your app.id from WAC> -n net.wacapps.napi.android.sample/.WacCinemas`

Windows Example

A) Spoofing command



B) System Reply



Step 7: Certify Your App

When your in-app payments are working to your satisfaction, you are ready to have WAC certify your API keys so your app can initiate real purchases.

To do this, first complete your banking details so WAC knows how to pay you for your in-app sales. Then complete the compliance questionnaire for the operator markets you will sell your app in.

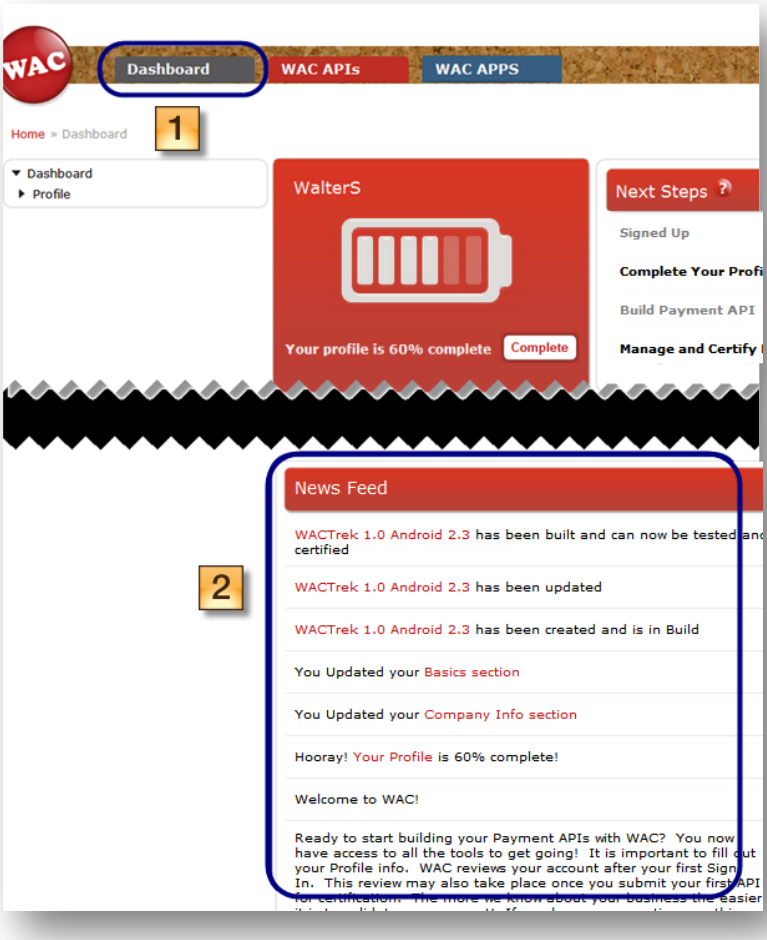
A) Tell WAC How to Pay You

B) Have WAC Certify Your App

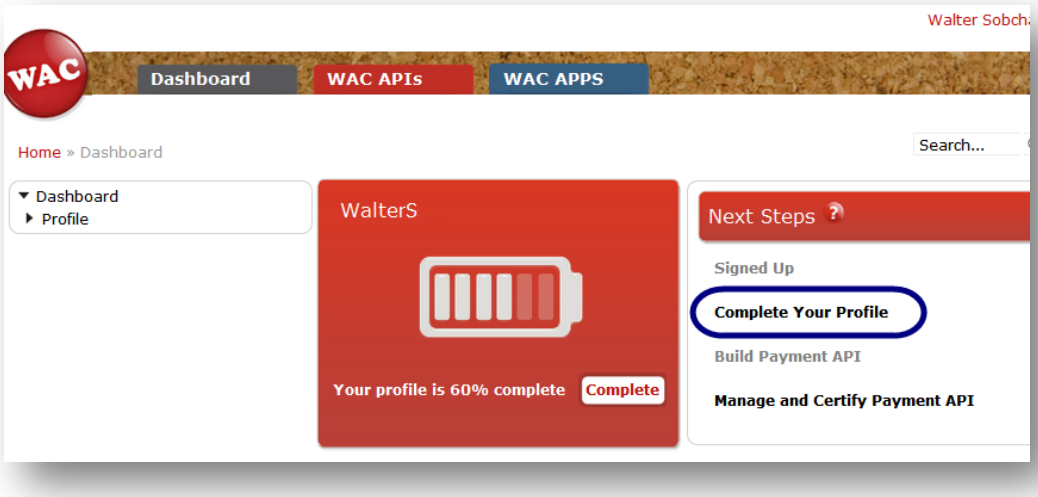
A) Tell WAC How to Pay You

1) Click **Dashboard**.

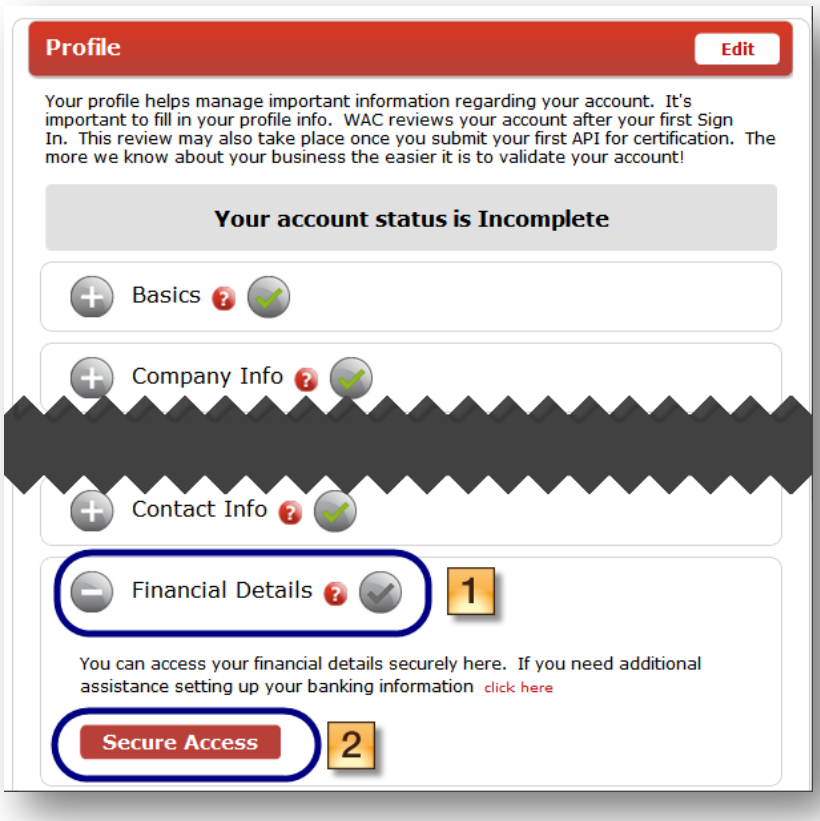
Tip: Notice that you can always see your current status and previous actions in the Newsfeed section of this screen.



2) Click **Complete Your Profile**.



3) In the Financial Details section, click **Secure Access**.

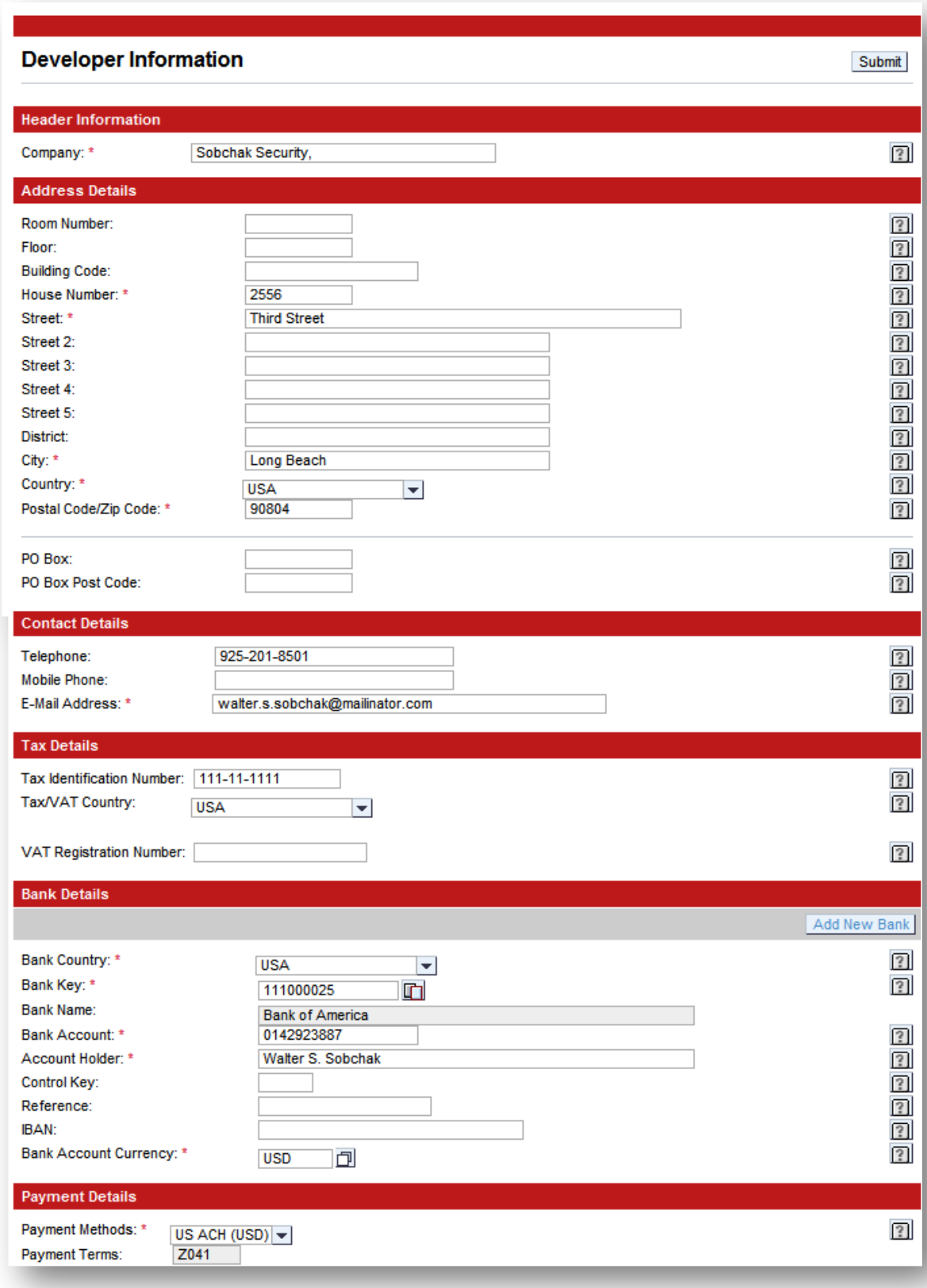


4) Complete the **Developer Information banking screen**.

Important Notes:

- Before you start entering your details, be certain you have a bank statement or your bank details handy.
- Ensure that your banking details are correct -- WAC cannot make payments to you unless your information is accurate.
- Bank details must be provided in English.
- After you have submitted your details you will receive confirmation that your account has been successfully set up. If there is a problem, you will be notified that your account has not been set up and why.
- Be certain to review WAC's Settlement FAQ for important details about getting credited for your app sales.
- If you have any questions about this form or your financial account with WAC, please contact settlement@wacapps.net.

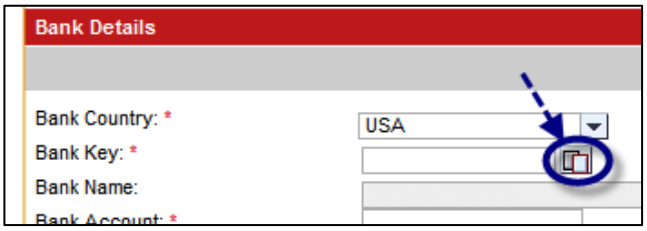
Fields marked with a red asterisk (*) must be completed.



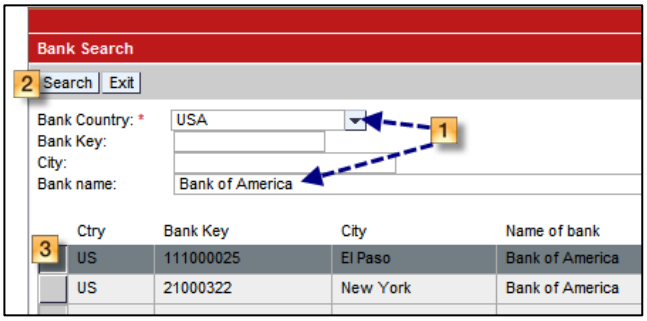
Field	Description
Name	This field is important for tax and legal purposes: If you are a registered company, enter your company legal name. Otherwise, enter your full name.
Address & Contact Details	If you are a company, enter the company information. Otherwise enter your personal information.
Tax Identification Number	For developers selling within the U.S., this is a nine-digit number obtained from the U.S. Social Security Administration (SSA) or the Internal Revenue Service (IRS). Leave this field blank if you do not have a Tax Identification Number.
TAX/VAT Country	If you are VAT/TAX registered in a country other than your country of residence and you wish to use this VAT/TAX registration for the sale of your app(s), enter that country's VAT/TAX registration. If you are not registered for VAT, enter your own country.
VAT Registration Number	Enter your VAT ID, i.e. VAT registration, number. This number identifies you to your tax authority for VAT purposes.
Bank Country	Enter the country of your bank

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>Bank Key</td><td>Your bank key -- also known as your sort code or routing code in the US -- is the code that identifies your bank branch. You will find it on your bank statements and printed as the middle group of computer-type figures at the bottom of your checks. It can be up to 15 figures depending on country. US dollar payments to US bank accounts always require a valid bank key/sort code/routing code.
<table>
<tr><th>Country</th><th>Sort Code</th><th>Bank Account</th><th>IBAN</th><th>SWIFT</th></tr>
<tr><td>Germany</td><td>29050101</td><td>12226411</td><td>DE75290501010012226411</td><td>SBREDE22XXX</td></tr>
<tr><td>Spain</td><td>21000178</td><td>0200452310</td><td>ES5221000178750200452310</td><td>CAIXESBBXXX</td></tr>
<tr><td>Great Britain</td><td>160079</td><td>10084423</td><td>GB59RBOS16007910084423</td><td>RBOSGB21021</td></tr>
<tr><td>Ireland</td><td>900199</td><td>85856560</td><td>IE37BOFI90019985856560</td><td>BOFIIE2D</td></tr>
<tr><td>Italy</td><td>0102501603</td><td>100000018690</td><td>IT53A0102501603100000018690</td><td>IBSPITTM</td></tr>
<tr><td>Netherlands</td><td>SOGE</td><td>0270202641</td><td>NL89SOGE0270202641</td><td>SOGENL2A</td></tr>
<tr><td>Portugal</td><td>00070038</td><td>00041560008</td><td>PT50000700380004156000831</td><td>BESCPTPL</td></tr>
<tr><td>Greece</td><td>0710033</td><td>0000033031352140</td><td>GR8307100330000033031352140</td><td>MIDLGRAA</td></tr>
</table>
Find and add your bank key as follows:

1) Click the finder icon.

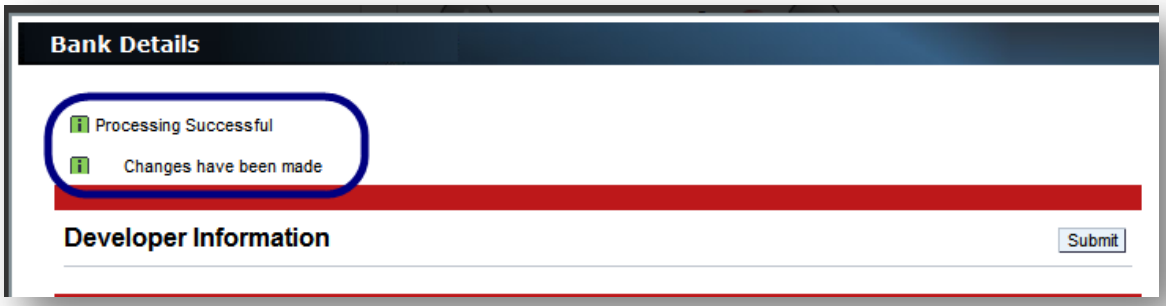
2) Search for your bank and click to add its bank key.

 
</td></tr>
<tr><td>Bank Name</td><td>Be certain to enter the full bank name.</td></tr>
<tr><td>Bank Branch</td><td>Enter the bank branch name or bank branch number for Japanese bank accounts. If none enter the bank city.</td></tr>
<tr><td>Bank Street</td><td>Enter the number and street where your bank branch is located.</td></tr>
<tr><td>Bank City</td><td>Enter the city where your bank branch is located.</td></tr>
<tr><td>Bank Account</td><td>Enter your bank account number.</td></tr>
<tr><td>Account Holder</td><td>Enter account holder name exactly as it appears on your statement.</td></tr>
</table>

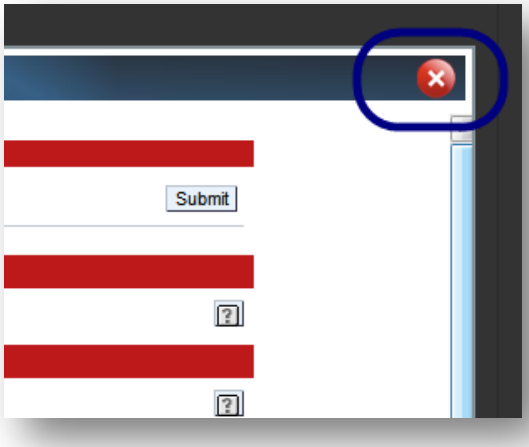
Field	Description
Control Key	The control key is also known as "account type" and indicates the type of account: checking or savings: • 01: Checking • 02: Savings **Important:** This key is mandatory for all U.S. dollar payments and mandatory if your bank account is held in Spain.
Reference	Enter the reference that you would like to appear on your bank statement. For example, 'Network API Sales'
IBAN	Enter your International Bank Account Number if your bank account is held in the European Union. The IBAN consists of a two-letter country code, followed by two check digits and up to thirty alphanumeric characters known as the Basic Bank Account Number (BBAN). For example AD12 0001 2030 2003 5910 0100. To find out what your IBAN is, look on your paper statement - it is usually near your name and address along with the SWIFT code. **Note:** This field is mandatory in the European Union and optional everywhere else.
Bank Account Currency	Indicate your bank's type of currency.
SWIFT Code	The unique identification code of a particular bank; this code is sometimes found on account statements. This bank identifier code is eight or 11 characters long and composed of the following: 1. Four characters - bank code (only letters) 2. Two characters -Country code (only letters) 3. Two characters - location code (letters and digits). 4. Three characters - branch code, optional. Letters and digits.
Payment Methods and Payment Terms	Indicate the currency and method of payment transfer you would like for your app sale proceeds.

5) Click **Submit**.

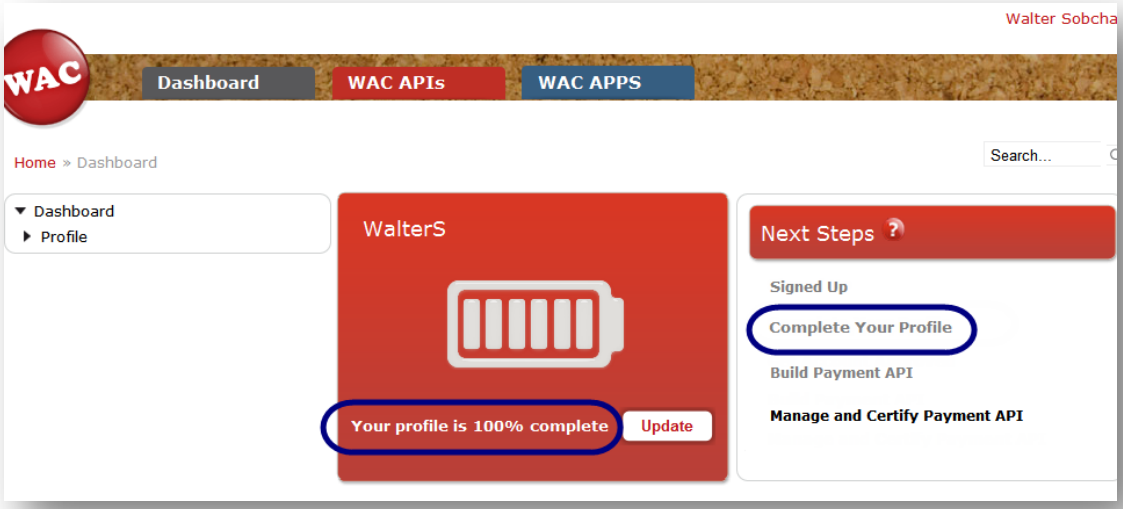
6) Be certain you see the submission is complete.
If there are errors, you will need to make corrections and resubmit or your changes will be lost.



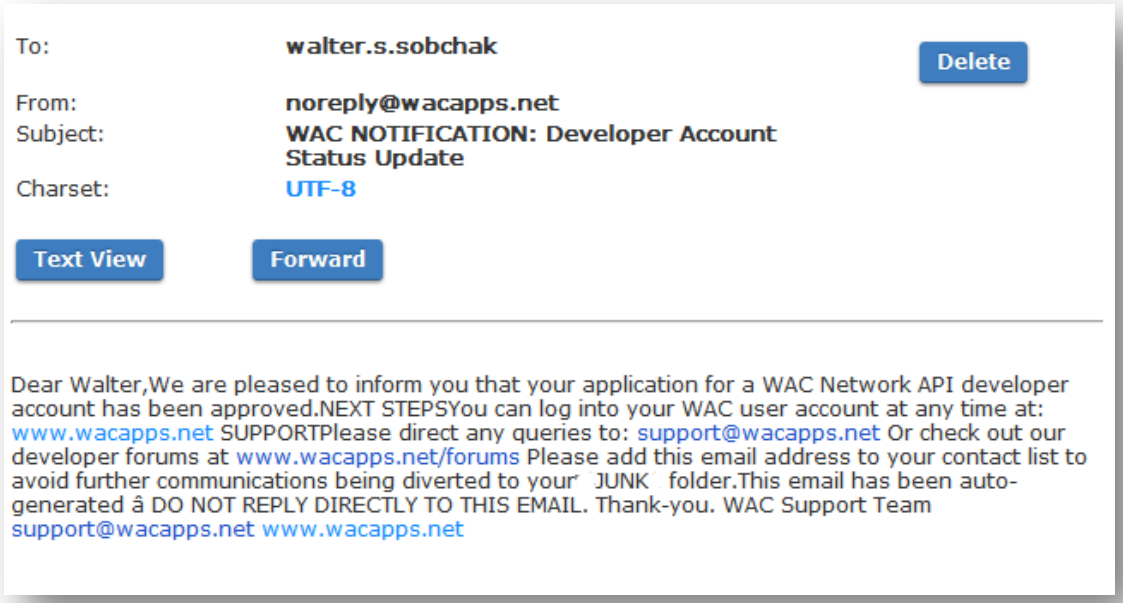
7) Close the Bank Details popup.



8) The **Complete Your Profile link** should no longer be active – adding your banking details completes your WAC profile.



Within 24 hours, WAC's compliance team will verify your banking information and emailed you with the results.



B) Have WAC Certify Your App

You are ready to have WAC certify your app, after which your WAC APIs will be activated and your in-app purchases will be live transactions. When you submit your app for WAC certification, WAC verifies that:

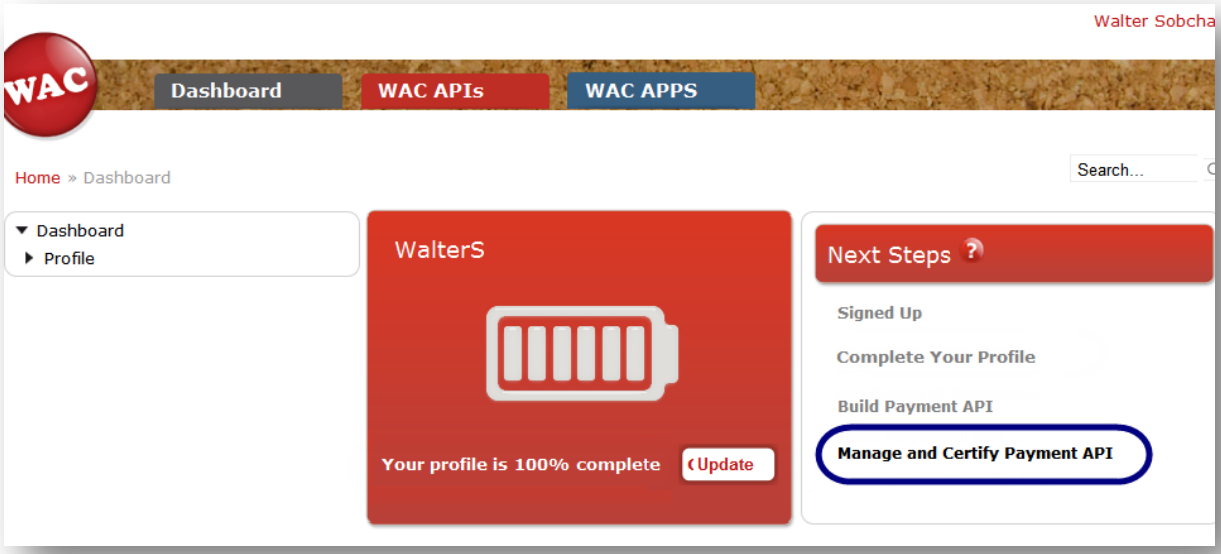
- Your app is working correctly with your WAC API keys.
- Your app is set up correctly with the operator markets you selected.
- Your selected operator markets accept your app's content ratings.

Note: The time required for app certification is different for each operator. Please plan for the following certification times.

Operator	Country	Time to Certification
AT&T	United States	Within 48 hours
O2	Germany	Within 5 days
Telekom	Germany	Between 7 to 16 days
SMART	Philippines	Within 7 days
Mobitel	Bulgaria	Within 7 days
KT	Korea	Within 7 days
SKT	Korea	Within 7 days
LGU+	Korea	Within 7 days

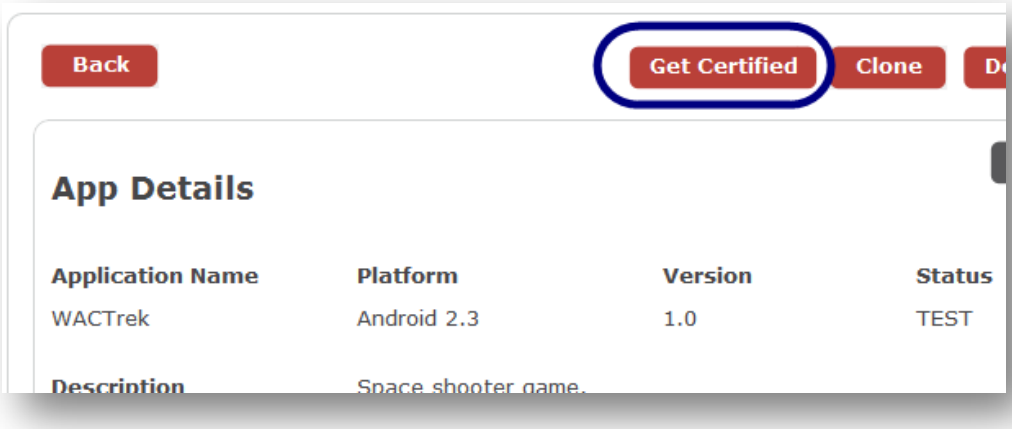
Submit your app for WAC certification as follows:

1) Click **Manage and Certify Payment API**.

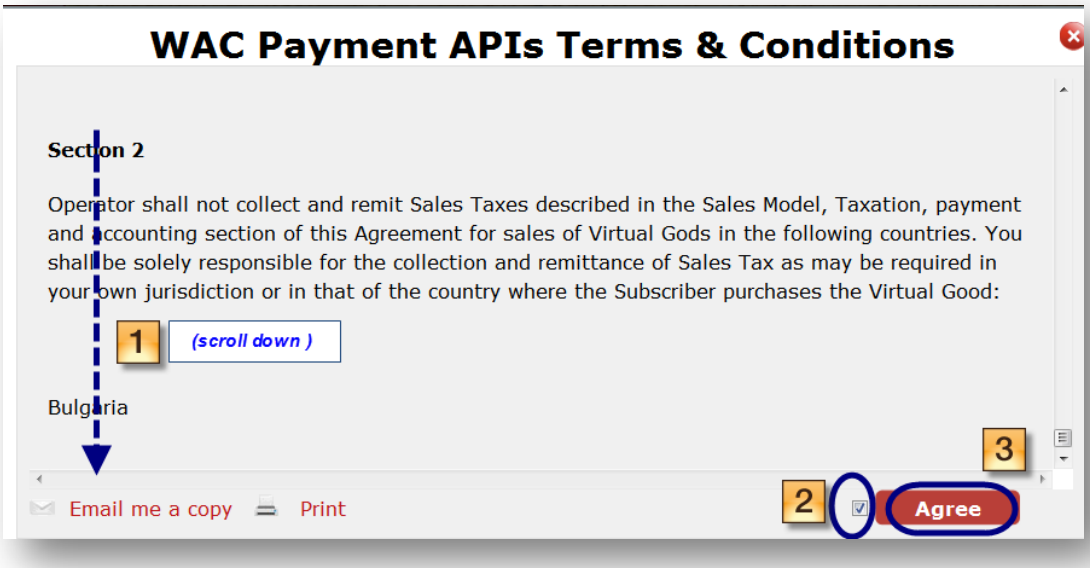


2) Click your app.

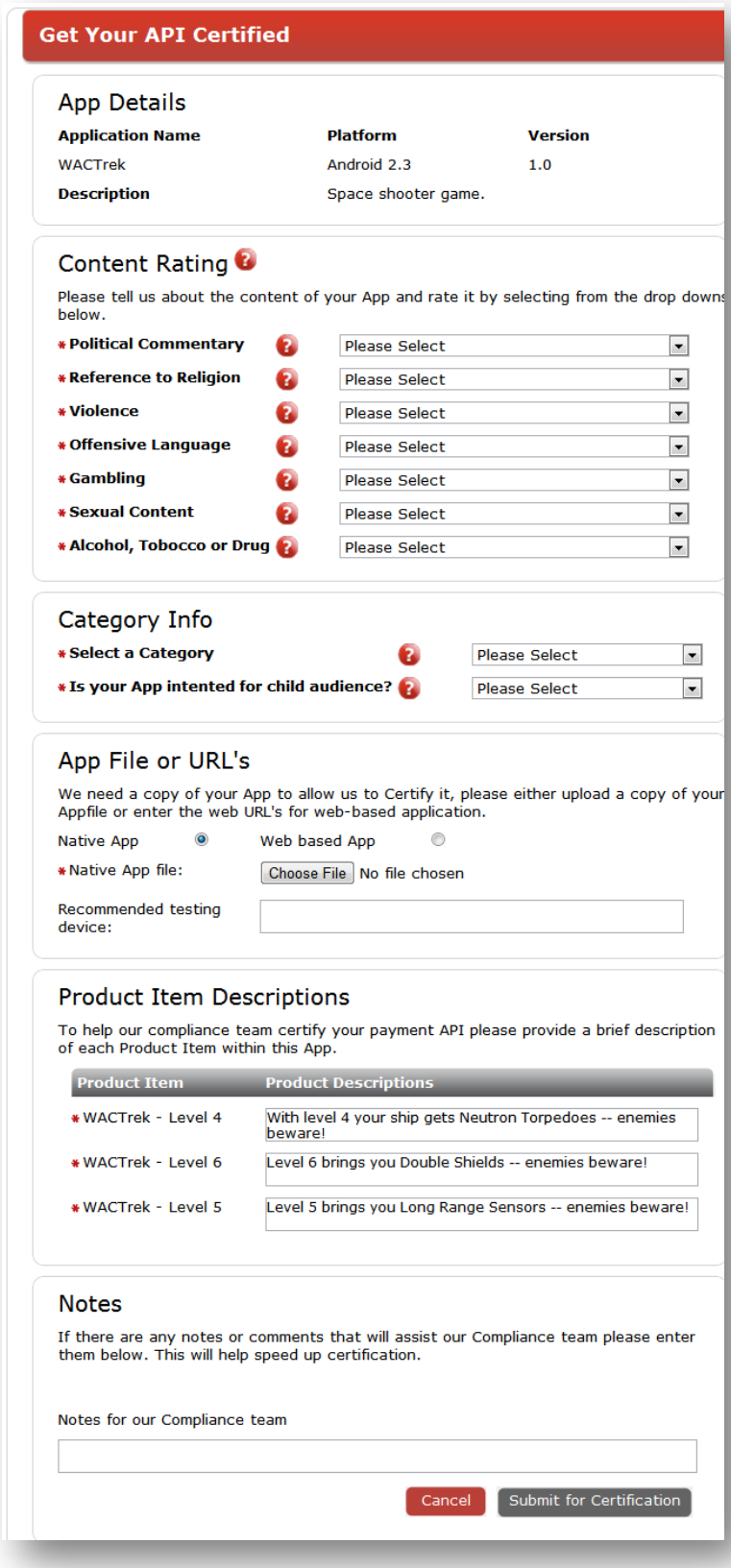
3) Click **Get Certified**.



4) Review and agree to the terms and conditions. (You will receive an email with confirmation of this agreement.)

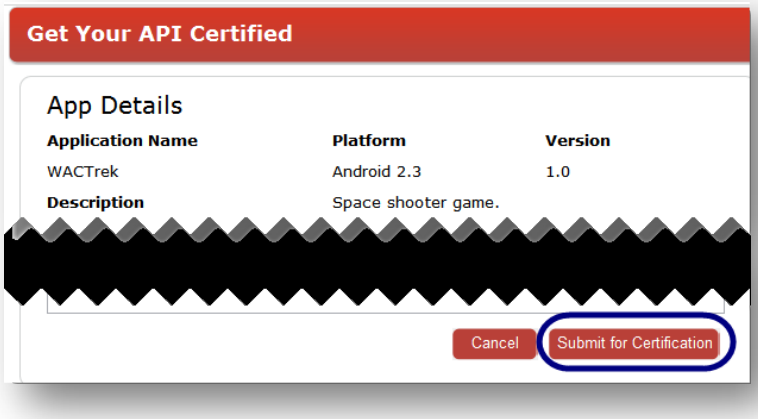


5) Complete the certification screen as indicated in the table.

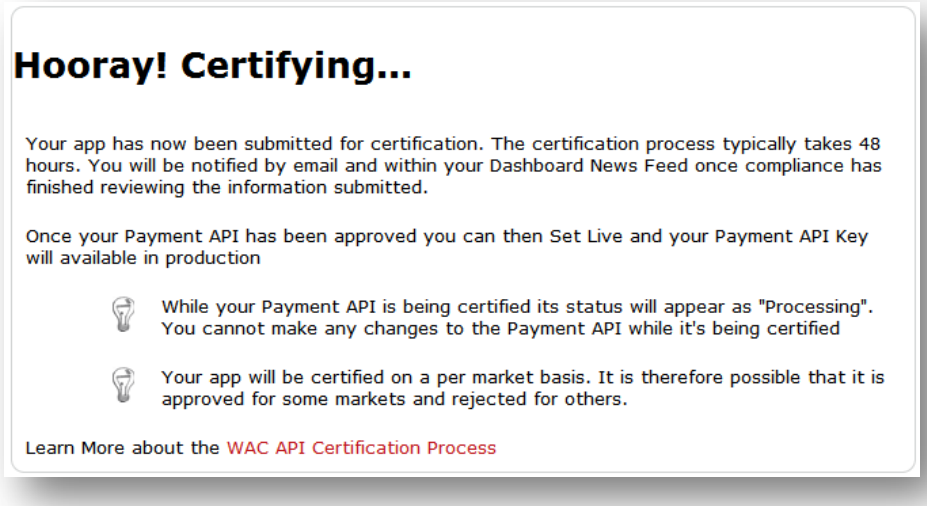


Section	Description
Content Rating	Operators require this information to ensure apps comply with their market's content guidelines, such as the assignment of age ratings to an app's content.
Category Info	Indicate your app's category (such as game, book, etc.) and whether it is intended primarily for children.
Add File or URL's	Use the **Choose File button** and upload your app. If there is a particular device this app should be tested on, enter this.
Product Item Descriptions	Verify your descriptions.
Notes	Enter any special considerations that will help WAC's compliance team assess your app.

6) Click **Submit for Certification**.



7) The certification process begins.

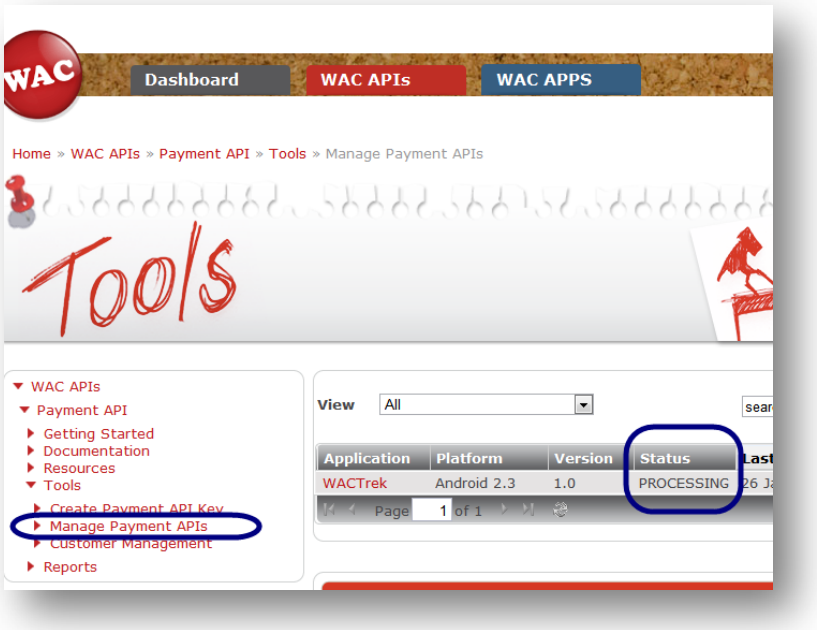


You will receive an email from WAC compliance (compliance@wacapps.net) when your app's certification begins.

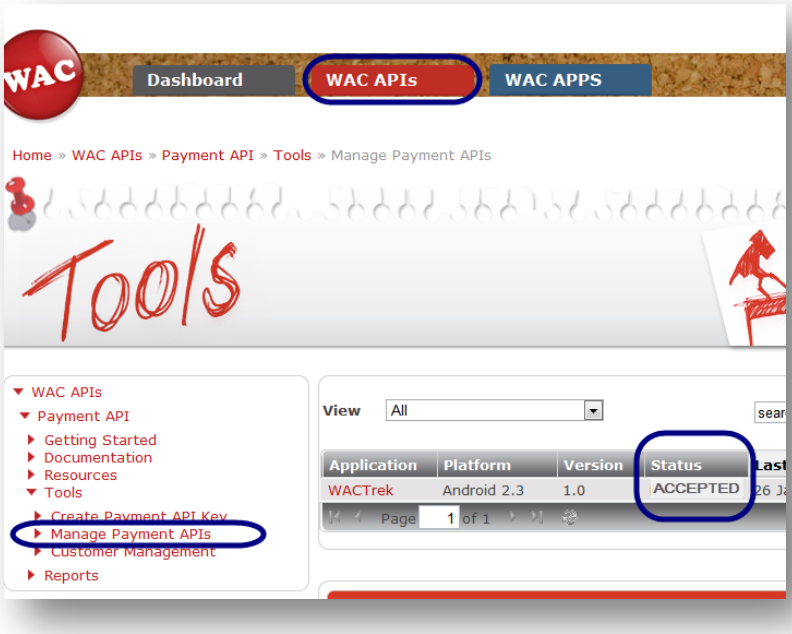
Note: The time required for app certification is different for each operator, note the certification duration for the operator market(s) you selected:

Operator	Country	Time to Certification
AT&T	United States	Within 48 hours
O2	Germany	Within 5 days
Telekom	Germany	Between 7 and 16 days
SMART	Philippines	Within 7 days
Mobitel	Bulgaria	Within 7 days
KT	Korea	Within 7 days
SKT	Korea	Within 7 days
LGU+	Korea	Within 7 days

While certification is underway, your app's status will be **PROCESSING** and no WAC application or account updates will be permitted until certification is complete.



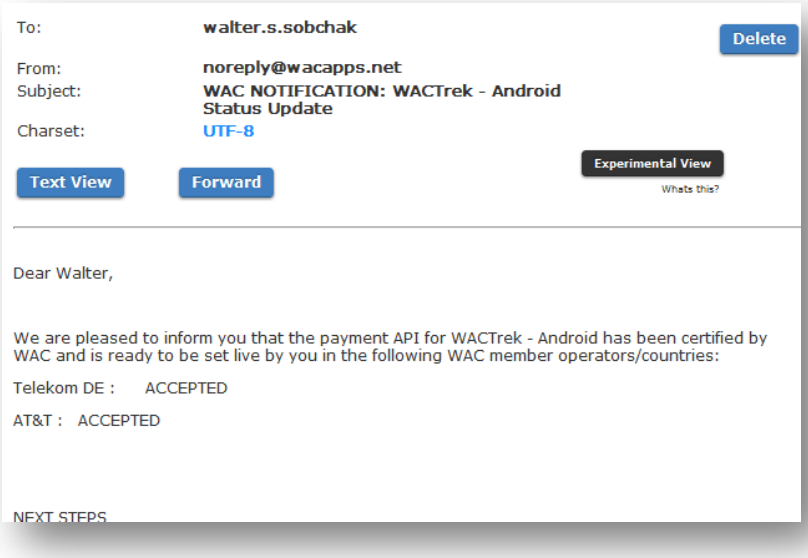
When your app is certified, this status will change to **ACCEPTED.**



Tip: App Status Definitions

Status	Description
TEST	You have created your WAC API keys (Step 3: Create Your API Keys). During **TEST** status, in-app purchases occur only in WAC's testing "sandbox" – no money is transacted.
PROCESSING	You have requested that WAC certify your app (Step 7: Certify Your App). While WAC processes your certification, no updates are permitted to your app or your WAC account.
ACCEPTED	WAC has concluded certification and accepted your app to continue on to publishing (Step 8: Set Your API Keys Live).
LIVE	You have set your app's API keys live in at least one market, after which all in-app purchases from your app's WAC API keys in the market will be *real* monetary transactions.

WAC will email you the certification results.

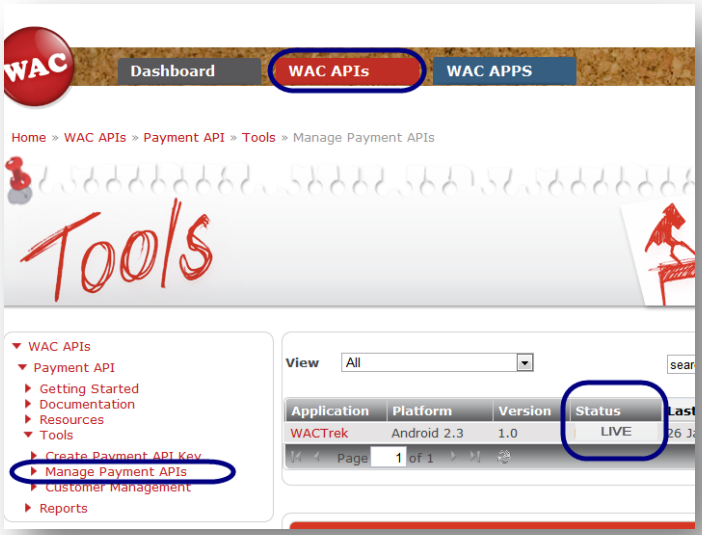


Finally, when your app is certified / accepted, contact WAC at support@wacapps.net to do the following:

- Perform a formal run-through of your app.
- Have WAC help you set up your production credentials.
- Have WAC ensure your chosen operator markets have your production credentials.

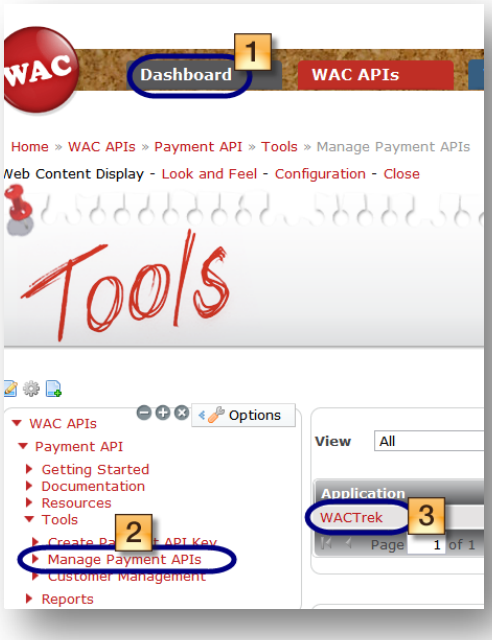
Step 8: Set Your API Keys Live

You are ready to set your WAC API keys to status **LIVE**.

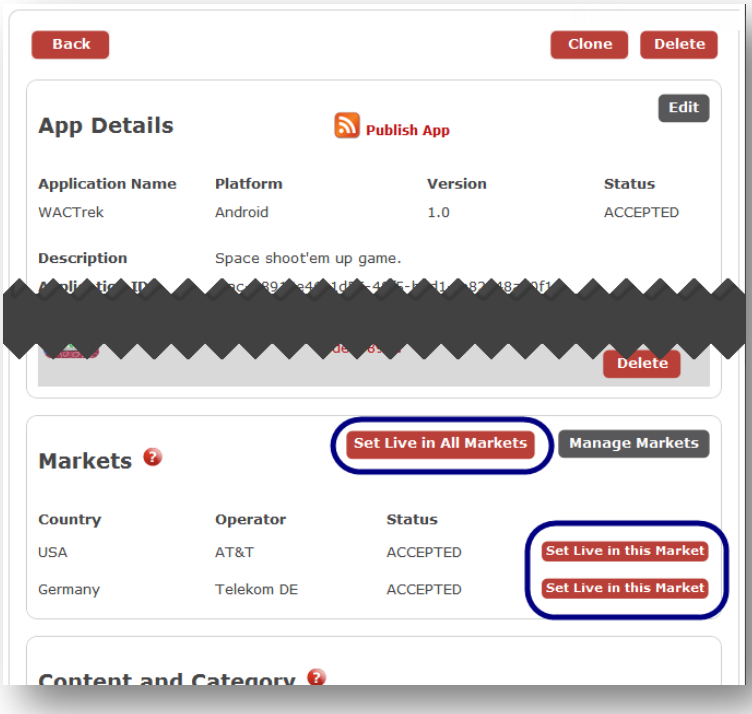


This means that all transactions made with your app against your WAC API keys will be real monetary transactions.

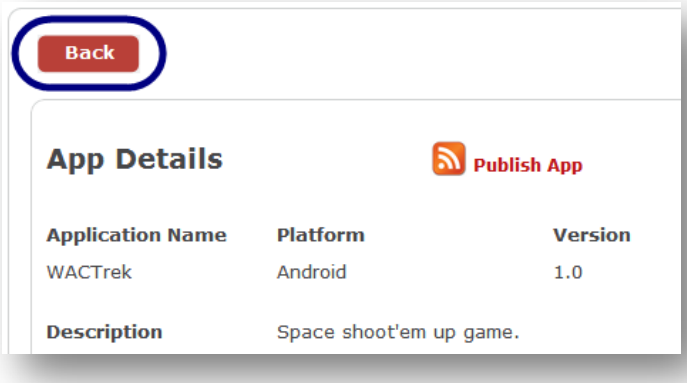
1) Return to your **App Details screen**.



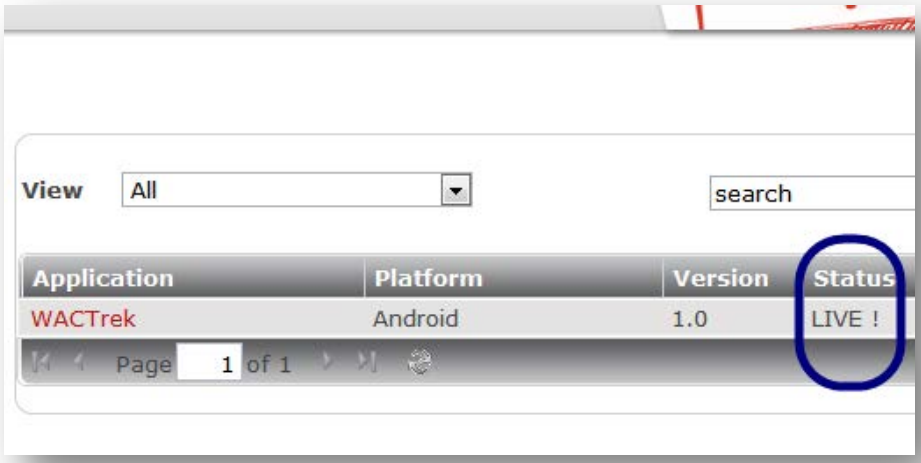
2) In the **Markets section**, set your API keys live (a) in particular markets or (b) in all your selected markets at once.



3) Click **Back**.

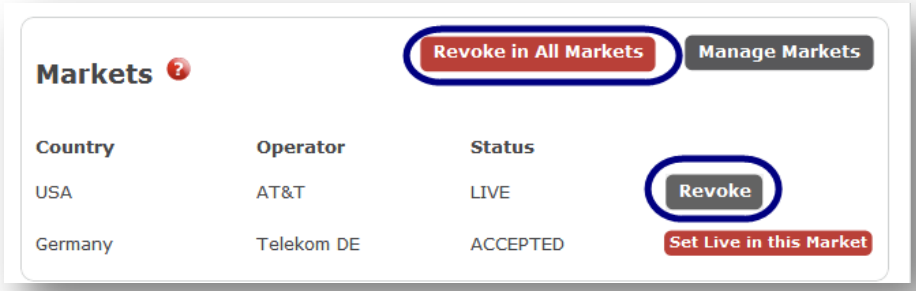


4) You see that your app's API status is now **LIVE**.

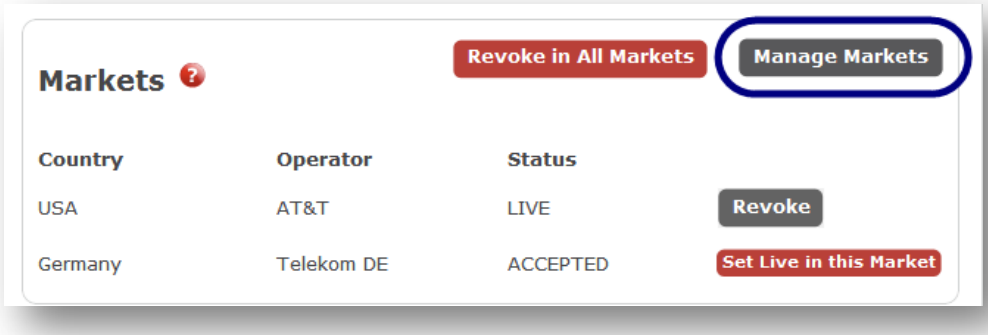


Notes:

- This status is LIVE if your app API keys are live in any one market – a status of LIVE does not indicate that your API keys are live in all markets.
- To remove LIVE status for your API keys in a market, simply return to the App Details screen and revoke the live status for one or all markets.



- You can add also additional markets at any time, click **Manage Markets**.



Step 9: Push App to Markets

That's it – your WAC APIs have LIVE status, indicating that you are ready to publish your app to its respective markets. Consider using WAC's publishing service, which makes it easy to publish to many major operators.

- **Having WAC Push Your App to Mobile Operators**
- **WAC's Support Policy**

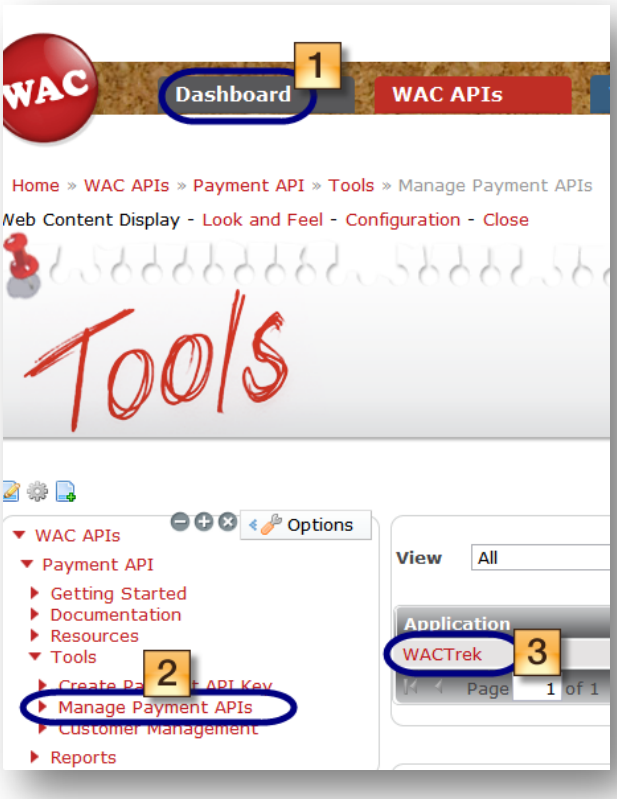
Having WAC Push Your App to Mobile Operators

You can use WAC's easy publishing service to publish your app to subscribers of the world's leading mobile operators.

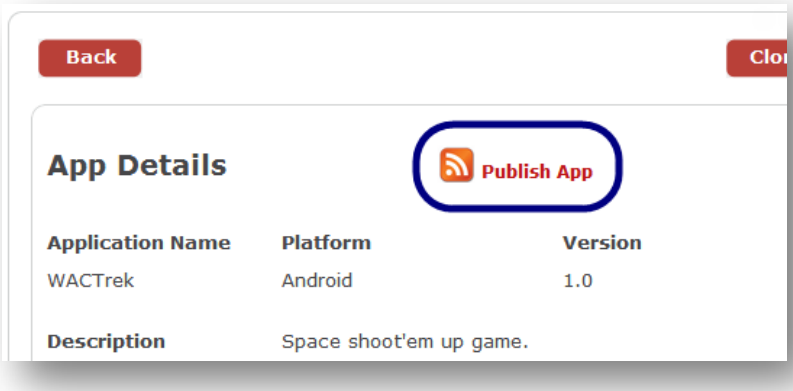
- See the list of operator markets: https://www.wacapps.net/live-operators1
- Find out more about WAC's easy publishing service: https://www.wacapps.net/wac-apps

If you decide to use this service for your app, start by activating your WAC API account for WAC's publishing service:

1. Go to your app's **App Details screen**.



2. Click the **Publish App icon**.



3. Your WAC account is now activated for WAC's publishing service and you are logged in there. From here follow the WAC Apps Developer Guide for illustrated instructions on how to continue.

WAC Support Policy

WAC's best practice is to resolve app user questions and issues within 3 working days. This commitment to quality customer service is both a good idea and a necessity:

- Customers who receive prompt, high-quality support are more likely to make repeat purchases and recommend the service to others.
- More than this, WAC's payment service is required to satisfy Ofcom requirements for speed and ease of query and issue resolution. Developers play a key role in ensuring they and WAC satisfy these quality support requirements.

Quality support helps everyone – please be responsive when your customer needs help.