

WP8 Native Plugin

Anyone can comment

[Short Overview](#)

[In App Purchases](#)

[Set up Guide](#)

[API References](#)

[Native Pop-ups](#)

[PlayMaker Actions](#)

[Support](#)

Shot Overview

This plugin provides the easy and flexible functionality of Windows Phone native functions. Function list will constantly grow, according to your feature request!

In App purchases:

- Purchasing
- Retrieving product details
- Event driven implementation

Native Pop-ups:

- Rate App
- Dialog
- Message

Features:

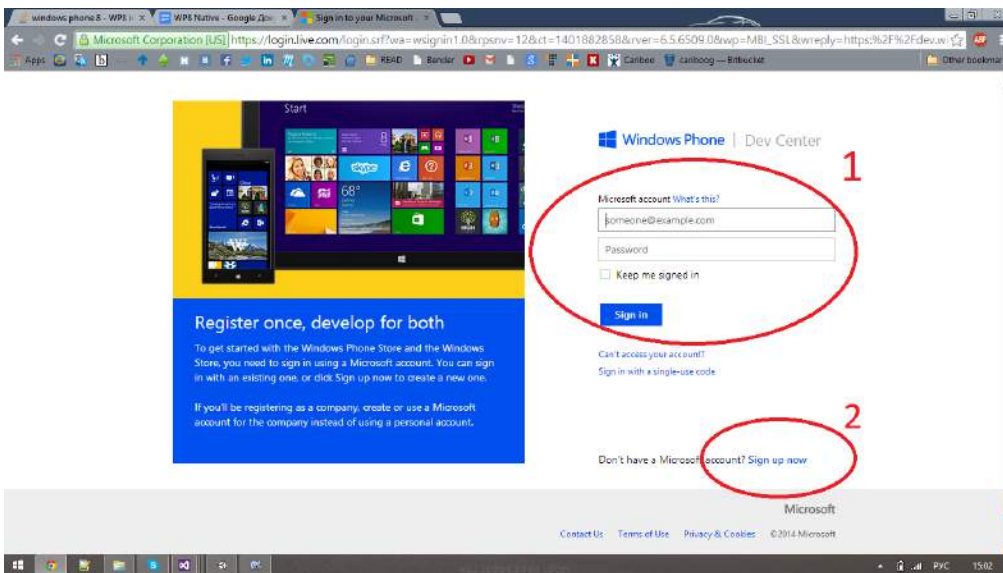
- PlayMaker Actions included
- Fully Documented
- Example scenes included

In-app Purchases

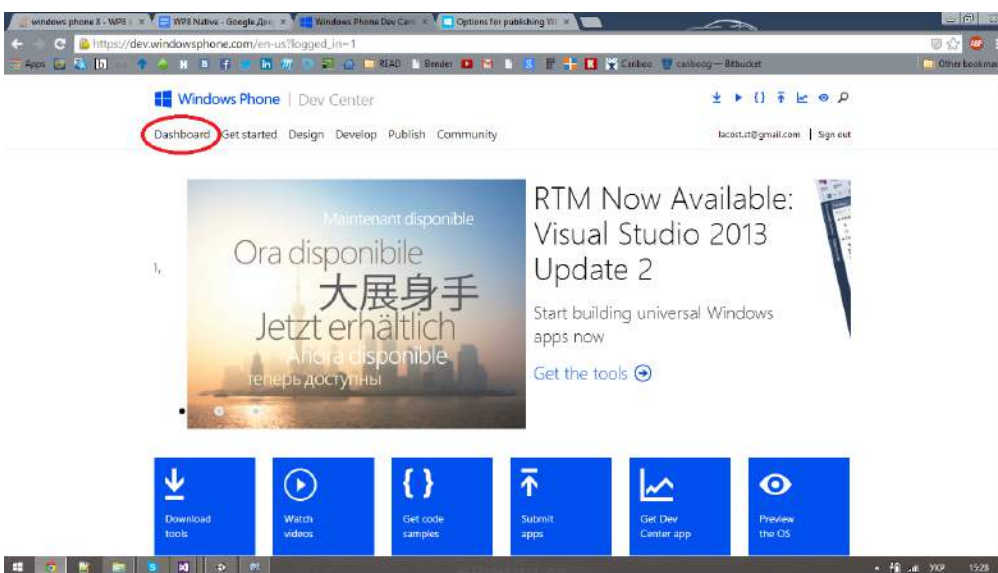
Set Up Guide:

First of all you need create [Windows Phone Developer Account](#).

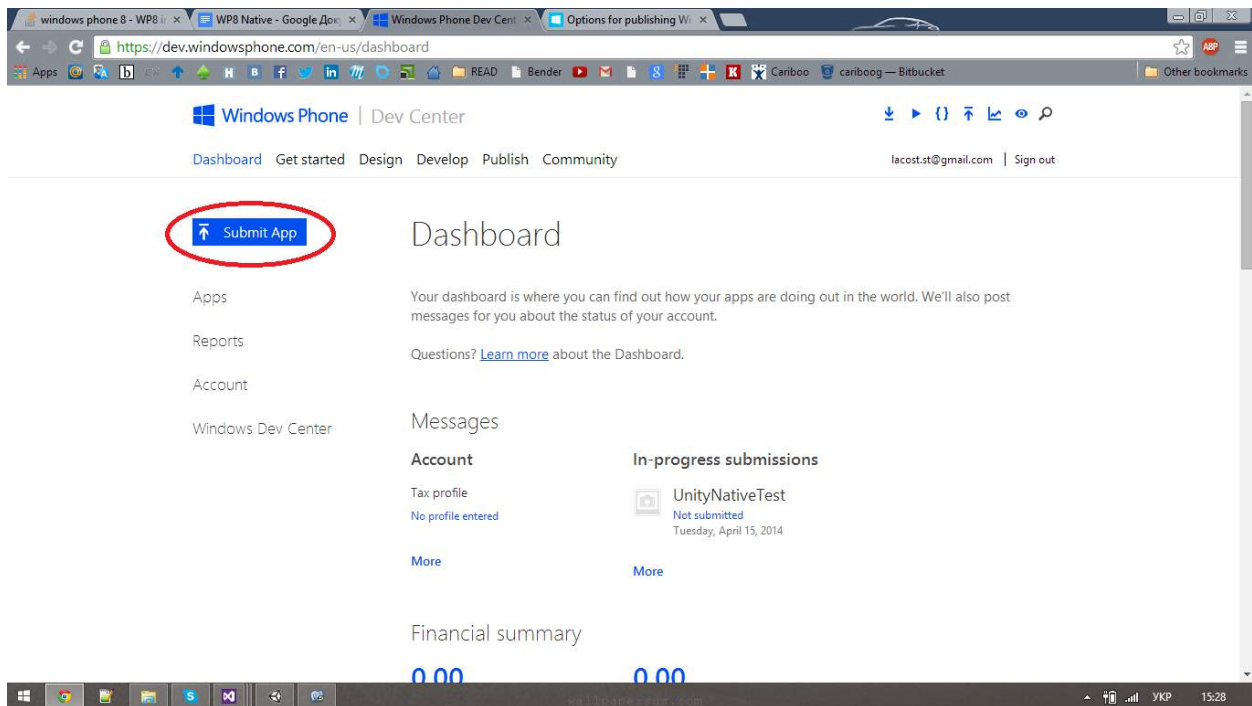
If you already have Windows Phone Developer Account fill your email and password, or click Sign up if you don't.



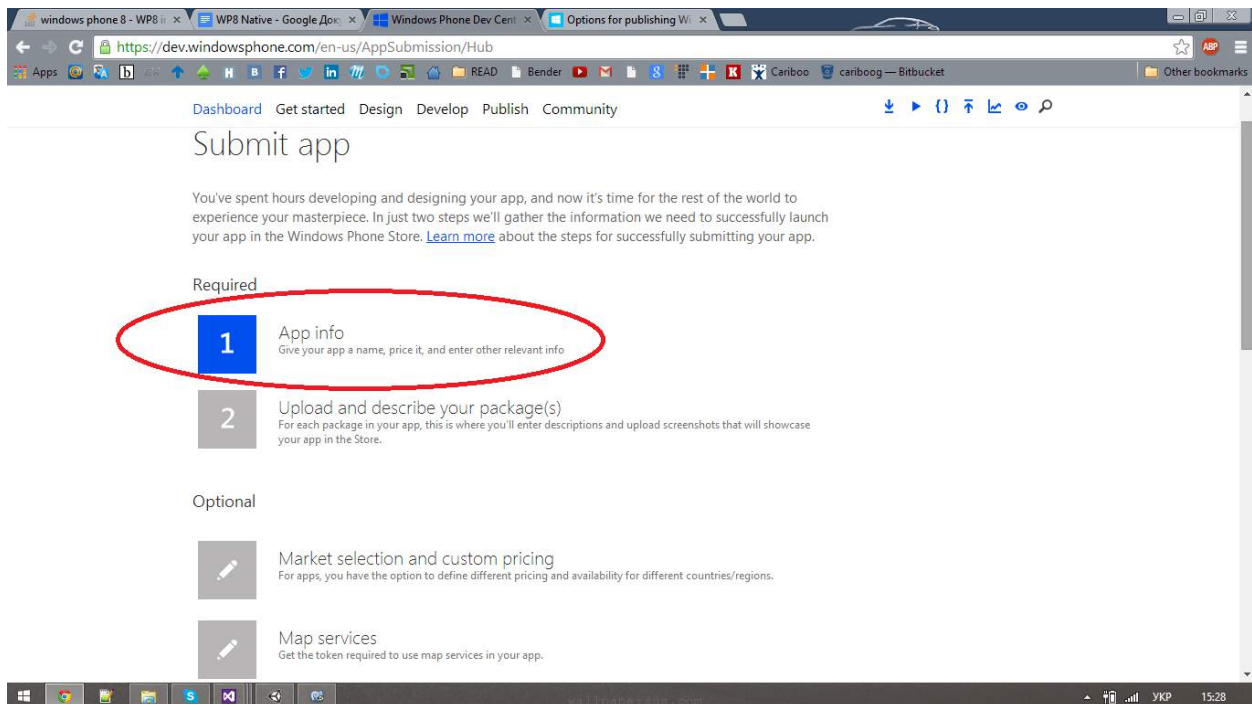
After you successfully registered your developer account and signed in, click **Dashboards** top menu item.



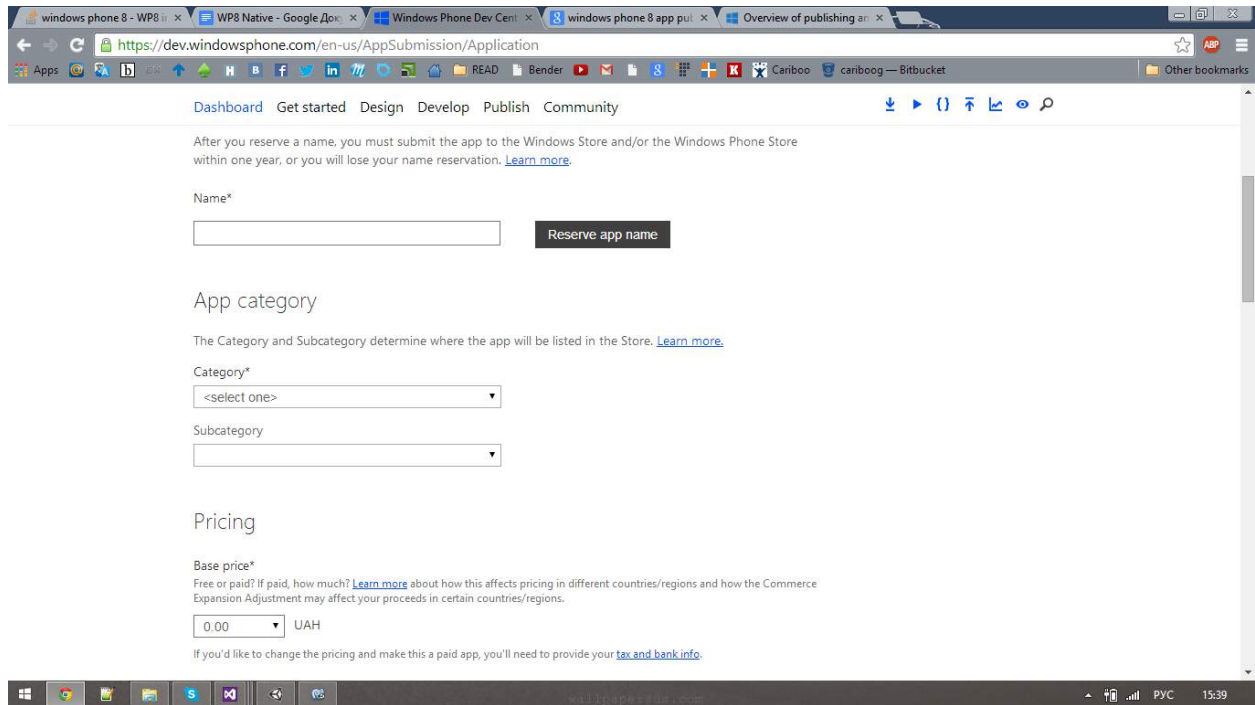
Let's create your first app. Click **Submit App** button.



Next step is to fill App Info



Fill your app name, category and price



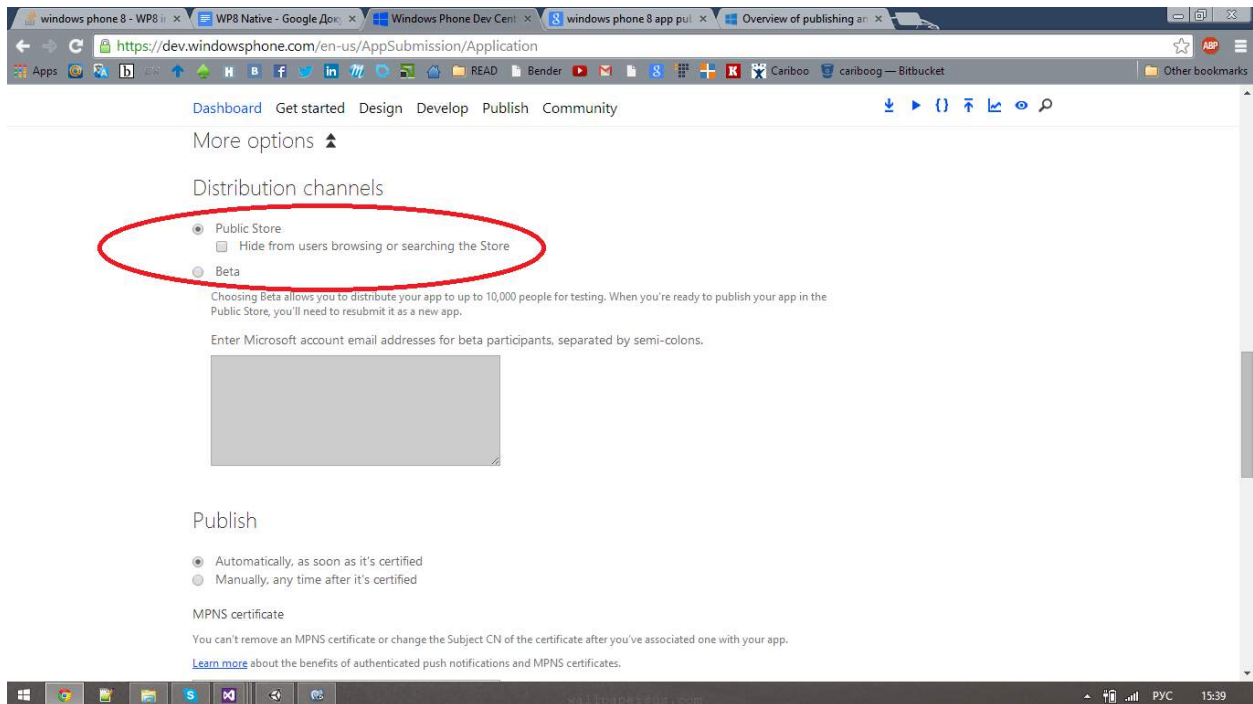
The screenshot shows the 'AppSubmission/Application' page in the Windows Phone Dev Center. The page has a navigation bar with links: Dashboard, Get started, Design, Develop, Publish, and Community. Below the navigation bar, there is a message: 'After you reserve a name, you must submit the app to the Windows Store and/or the Windows Phone Store within one year, or you will lose your name reservation. [Learn more.](#)'

The 'Name*' section contains a text input field and a 'Reserve app name' button.

The 'App category' section contains a message: 'The Category and Subcategory determine where the app will be listed in the Store. [Learn more.](#)' Below this are two dropdown menus: 'Category*' (with '<select one>' selected) and 'Subcategory'.

The 'Pricing' section contains a 'Base price*' label, a message: 'Free or paid? If paid, how much? [Learn more](#) about how this affects pricing in different countries/regions and how the Commerce Expansion Adjustment may affect your proceeds in certain countries/regions.', a text input field with '0.00' and a dropdown menu with 'UAH' selected, and a link: 'If you'd like to change the pricing and make this a paid app, you'll need to provide your [tax and bank info.](#)'

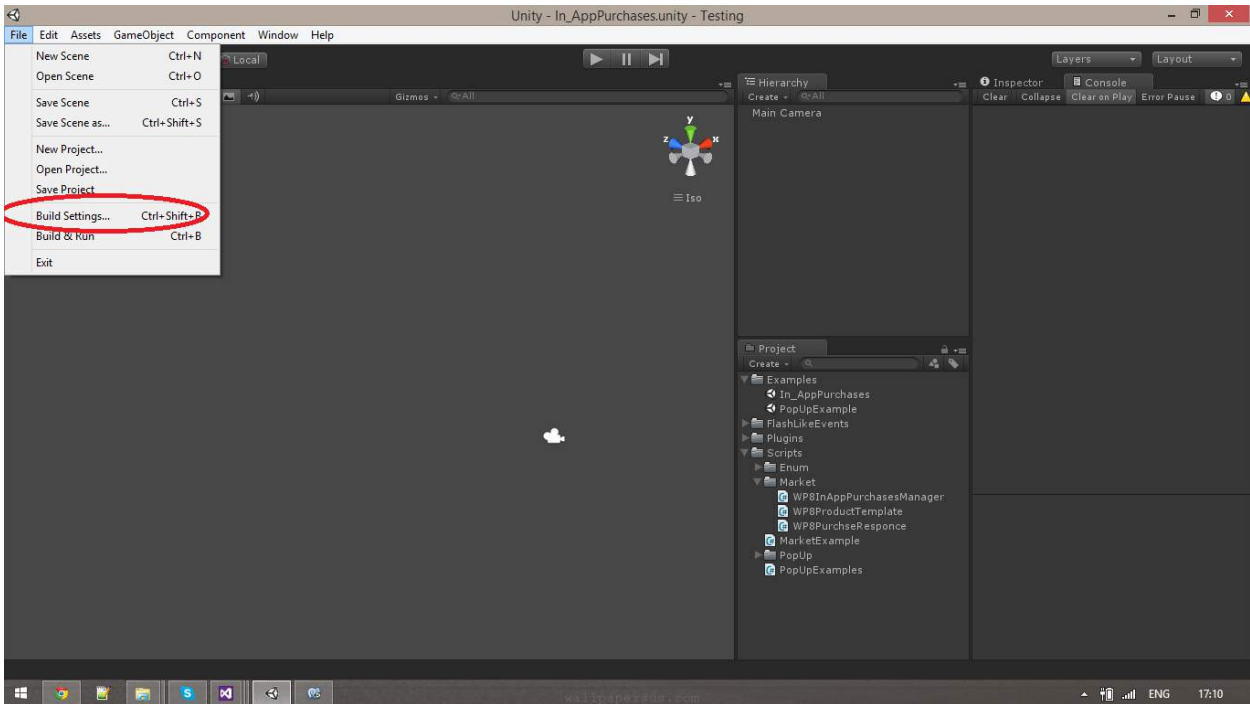
Next step you must select Distribution channel. I'll make test app, so choosed Beta.



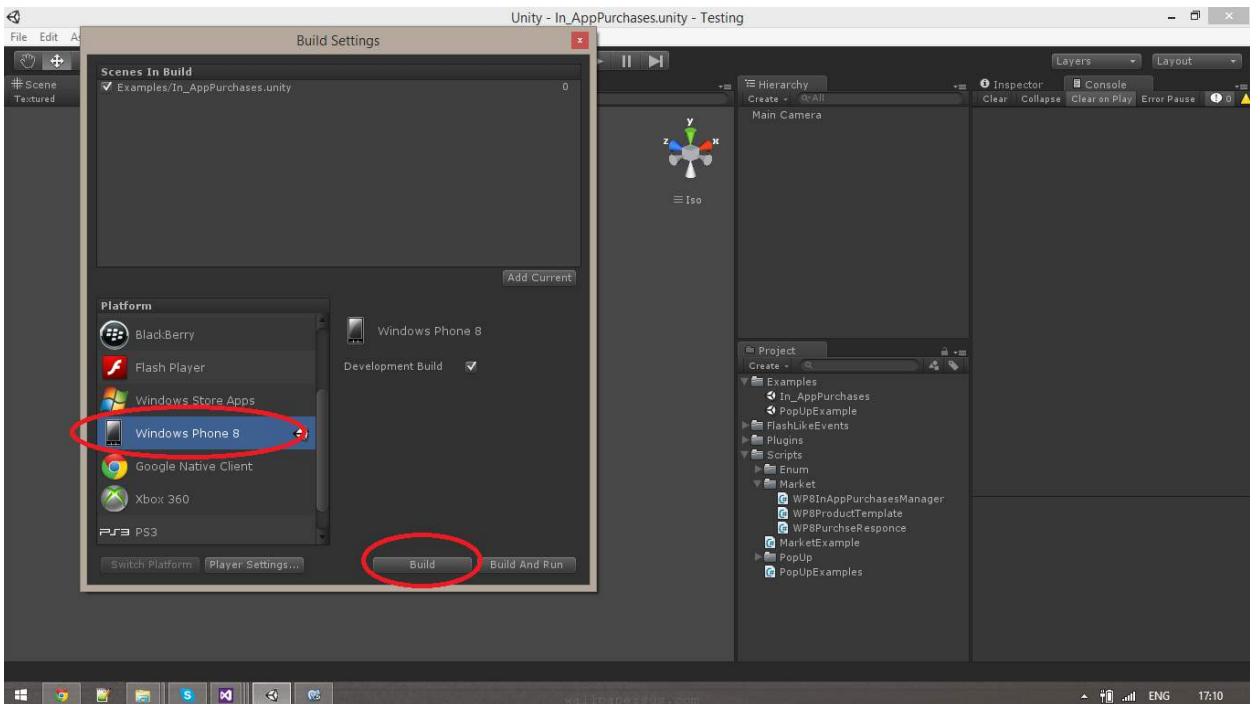
The screenshot shows the 'AppSubmission/Application' page in the Windows Phone Dev Center, specifically the 'More options' section. The 'Distribution channels' section is highlighted with a red circle. It contains two radio buttons: 'Public Store' (selected) and 'Beta'. Below the 'Public Store' radio button is a checkbox labeled 'Hide from users browsing or searching the Store'. Below the 'Beta' radio button is a message: 'Choosing Beta allows you to distribute your app to up to 10,000 people for testing. When you're ready to publish your app in the Public Store, you'll need to resubmit it as a new app.' Below this message is a text input field with a placeholder: 'Enter Microsoft account email addresses for beta participants, separated by semi-colons.'

The 'Publish' section contains two radio buttons: 'Automatically, as soon as it's certified' (selected) and 'Manually, any time after it's certified'. Below this is the 'MPNS certificate' section, which contains a message: 'You can't remove an MPNS certificate or change the Subject CN of the certificate after you've associated one with your app.' and a link: '[Learn more](#) about the benefits of authenticated push notifications and MPNS certificates.'

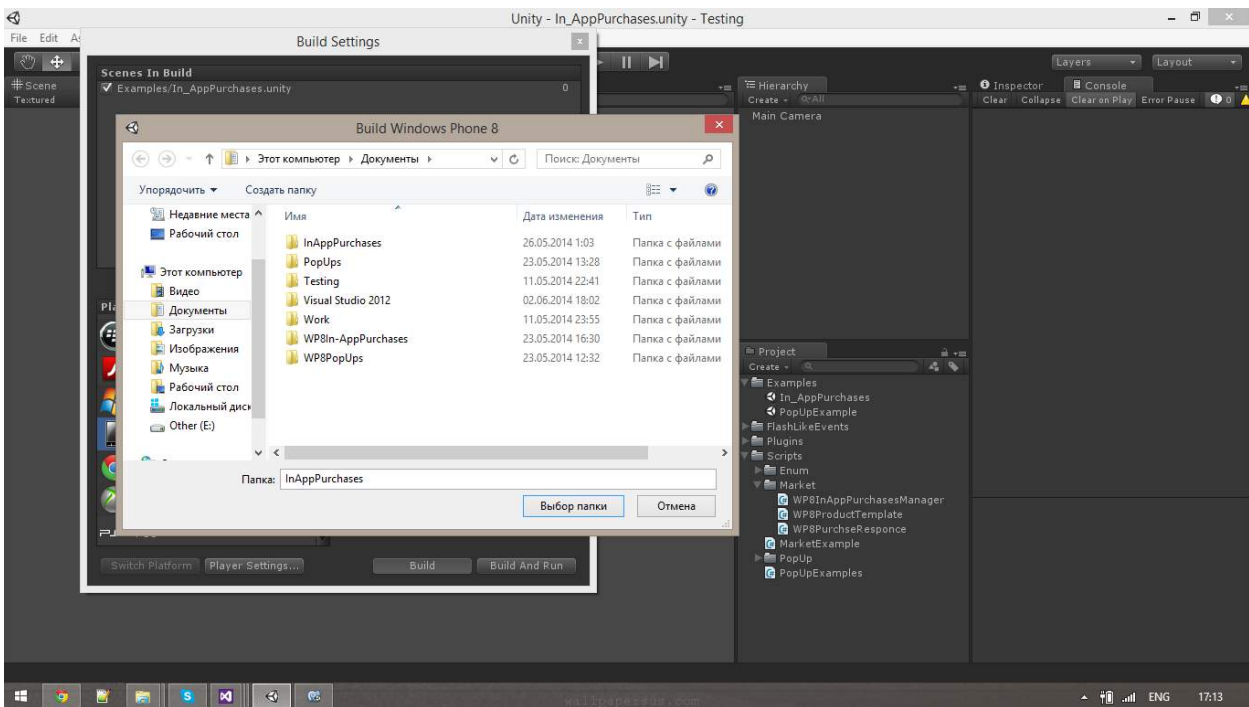
In the next step we need to upload XAP file. So let's switch to Unity.
XAP file will be generated with our project build.
Open your project and select File → Build Settings



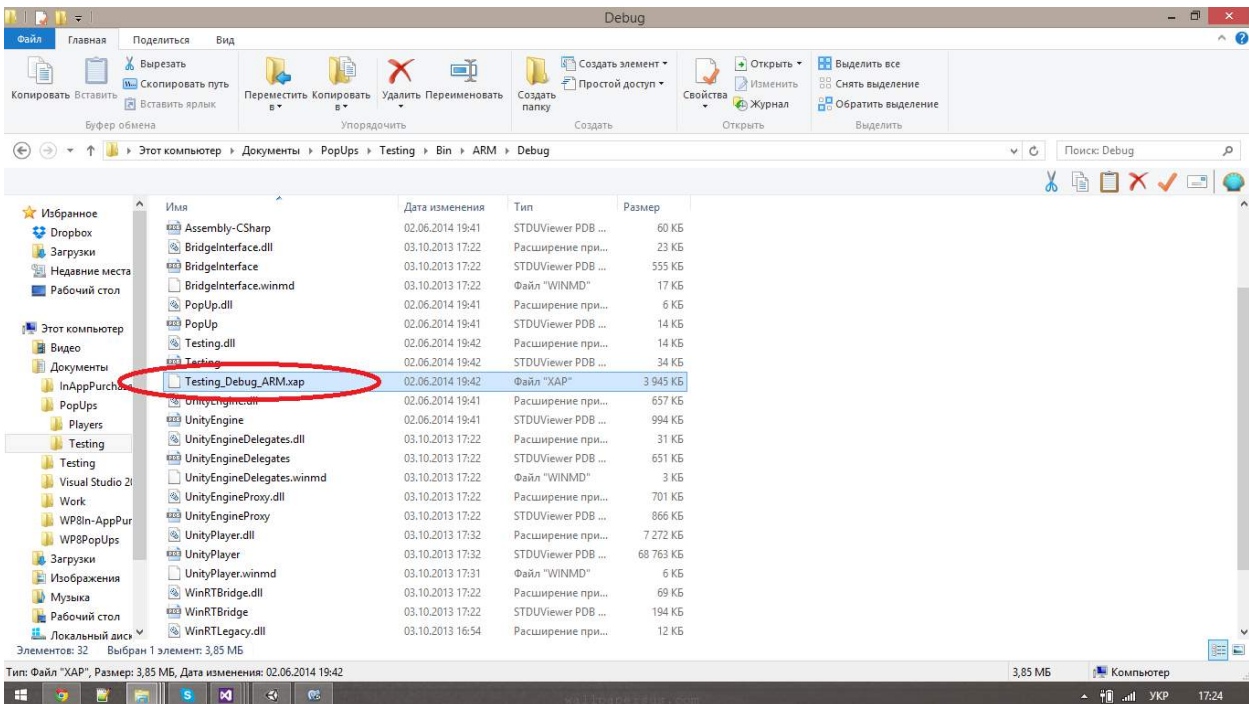
Choose Windows Phone 8, add click Build



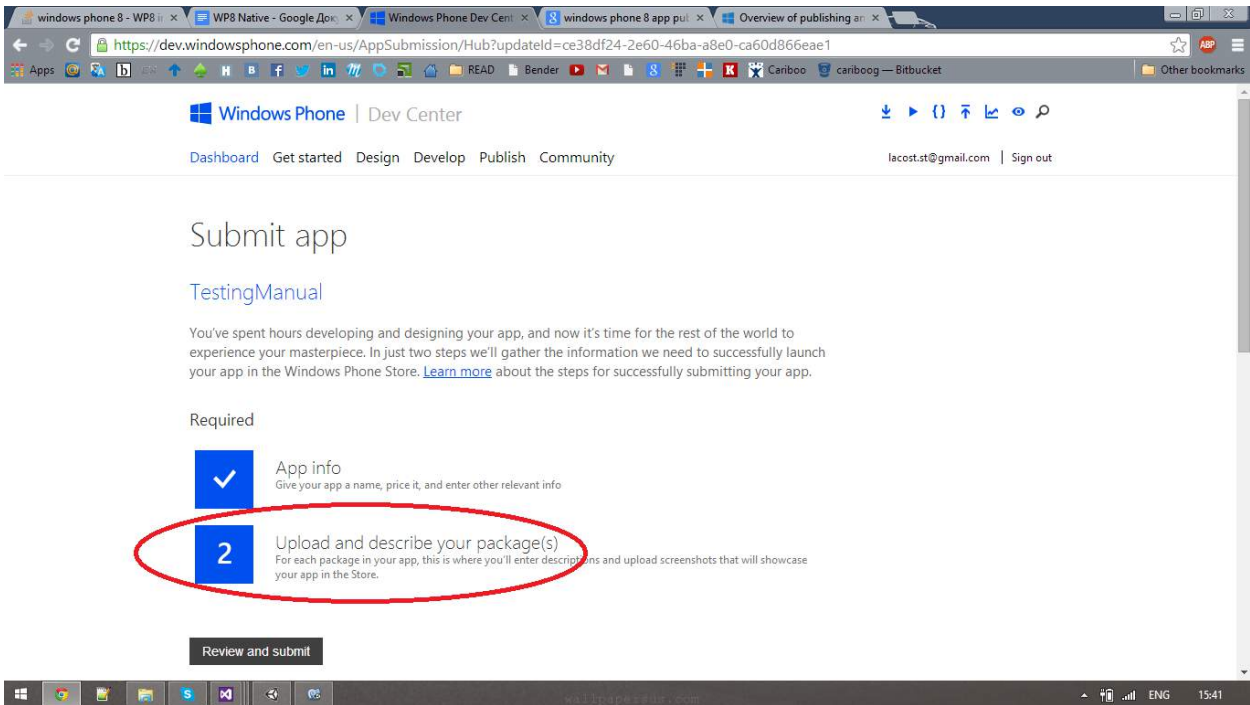
The choose destination directory.



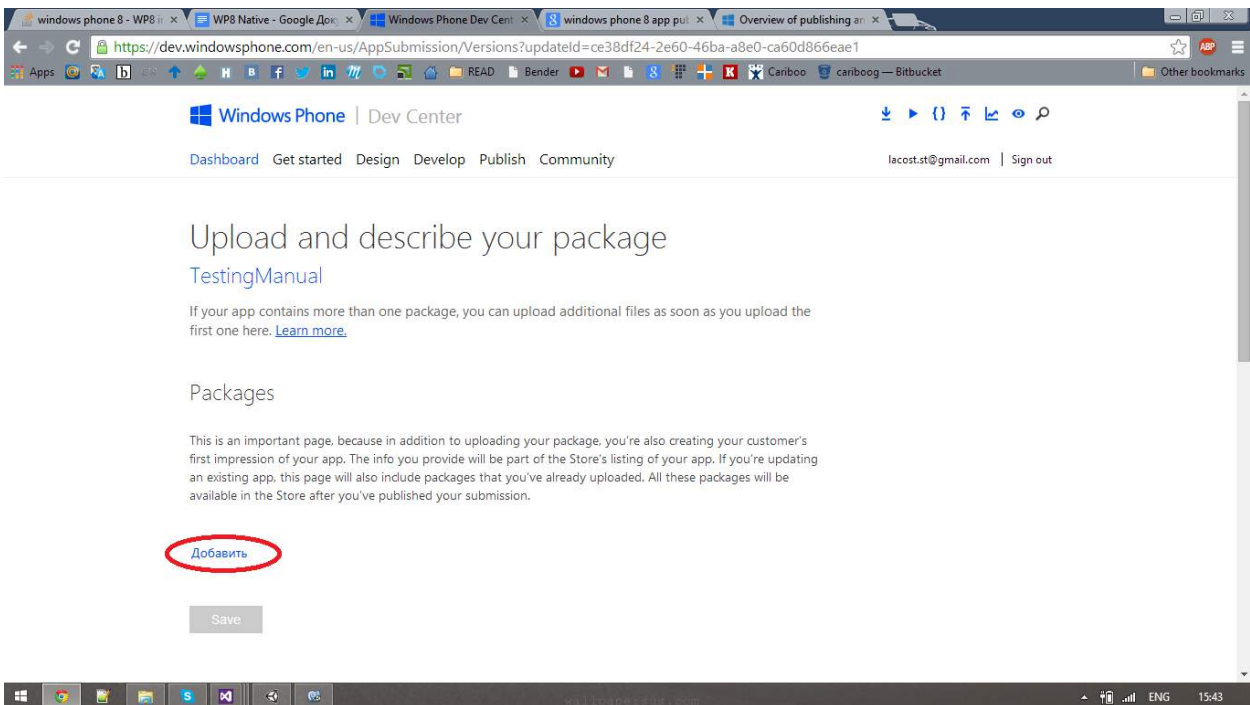
XAP file will be place at your build directory → Bin → ARM → Debug



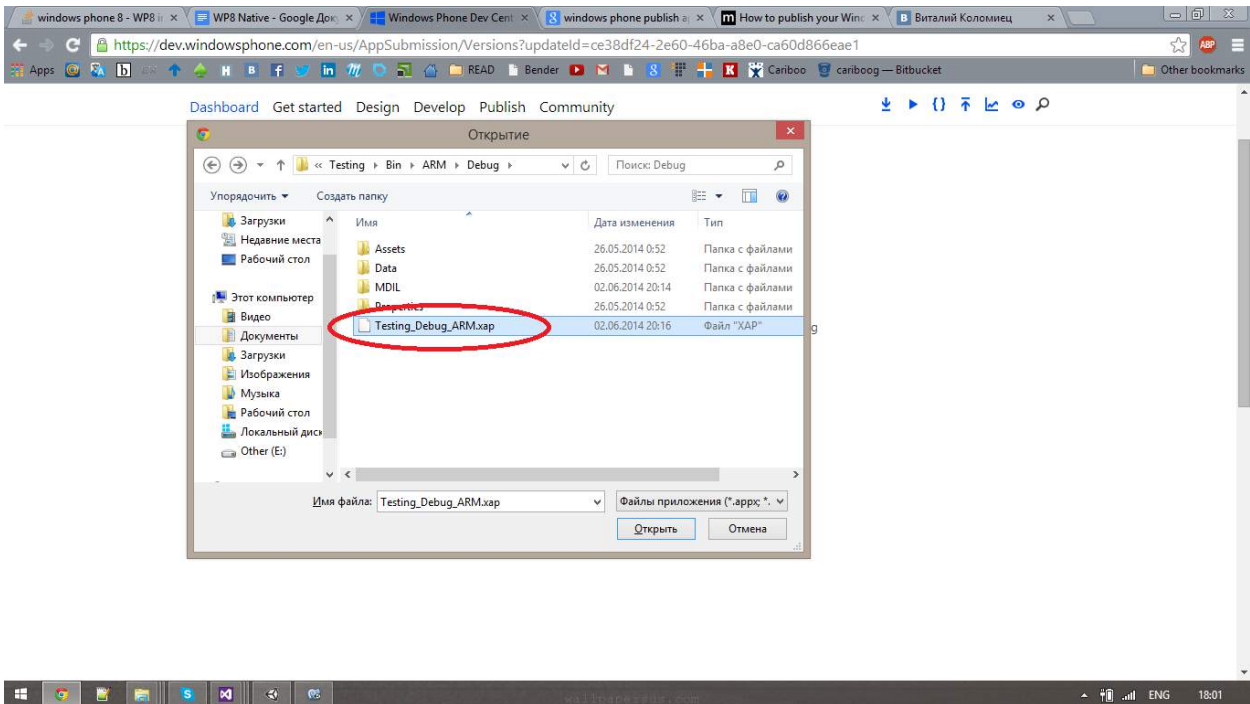
Now go back to the submission page.



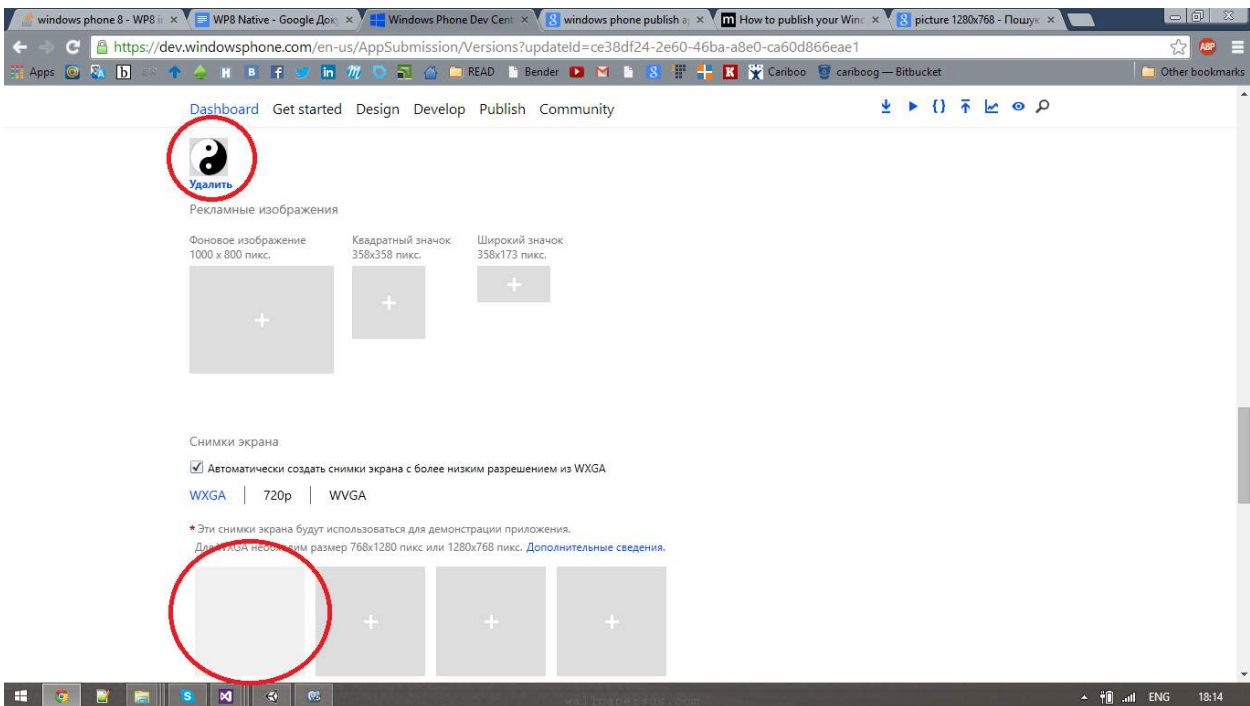
Click **Upload** and describe your package(s), then click **Add**



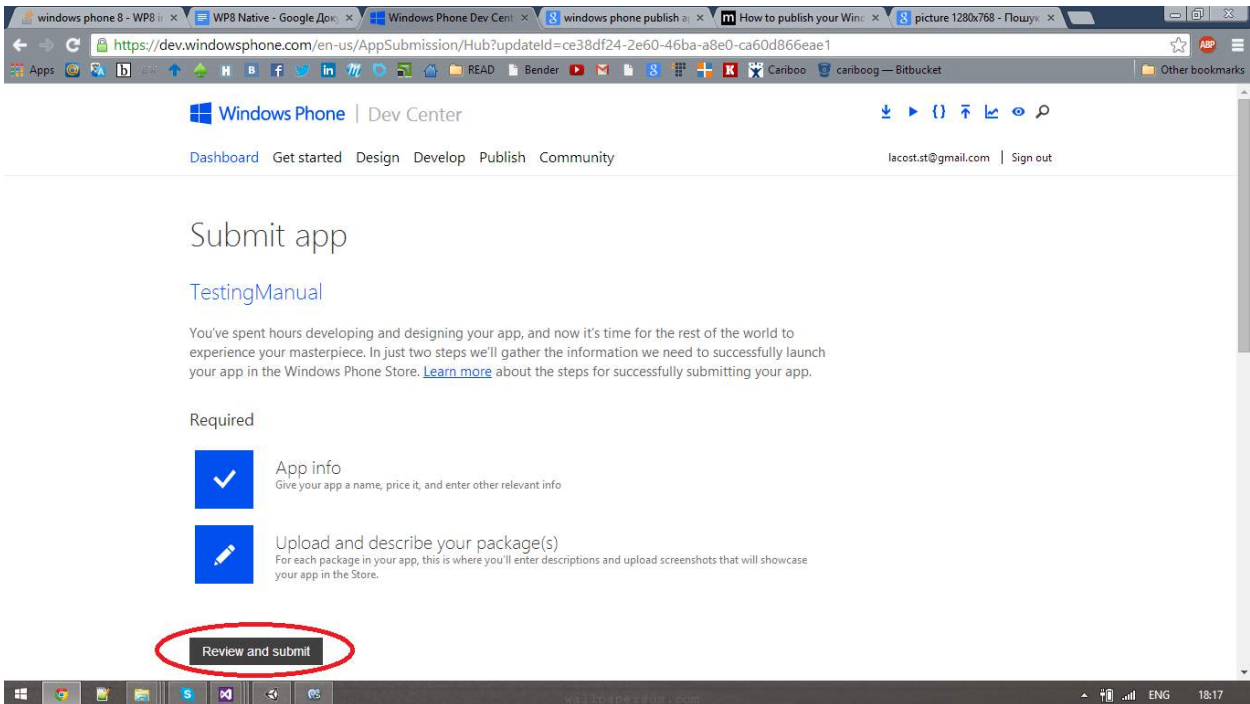
Choose XAP file in build directory



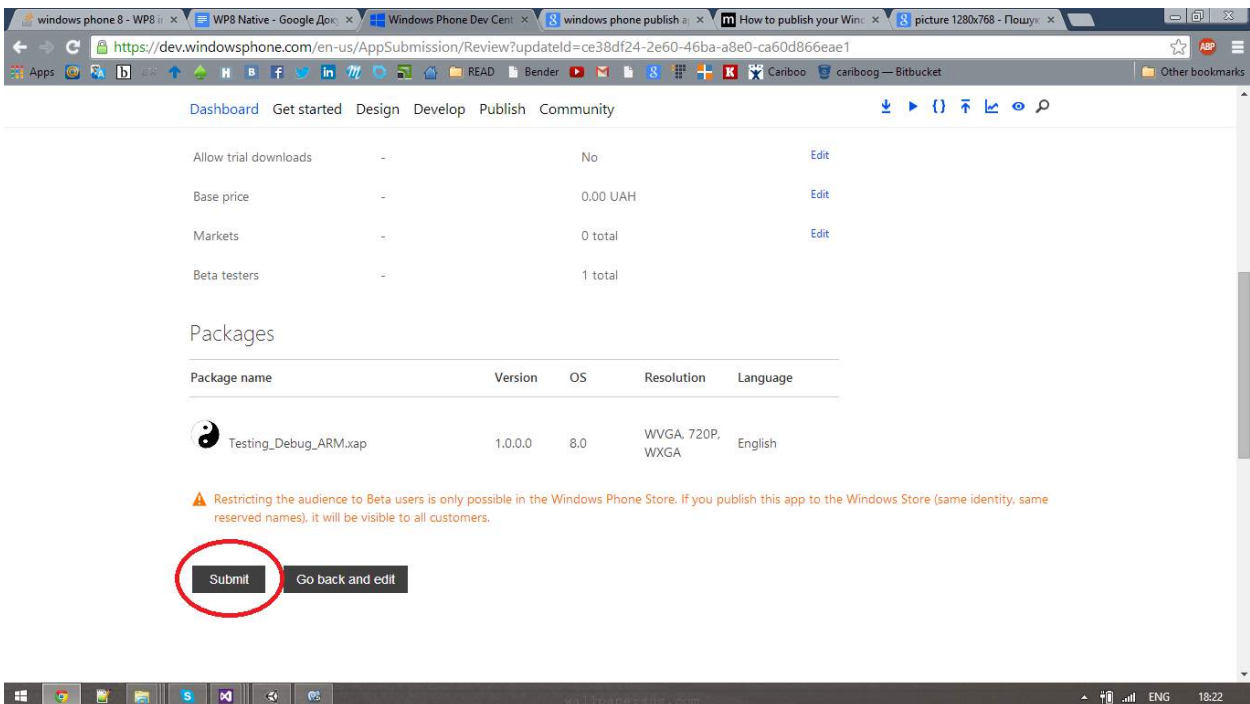
After loading you will see page with options, fill all fields with the "*" symbol and don't forget to load two .png images



Click Save and Review and submit on the next page



You will see Review Submission and if everything is allright click on **Submit** button

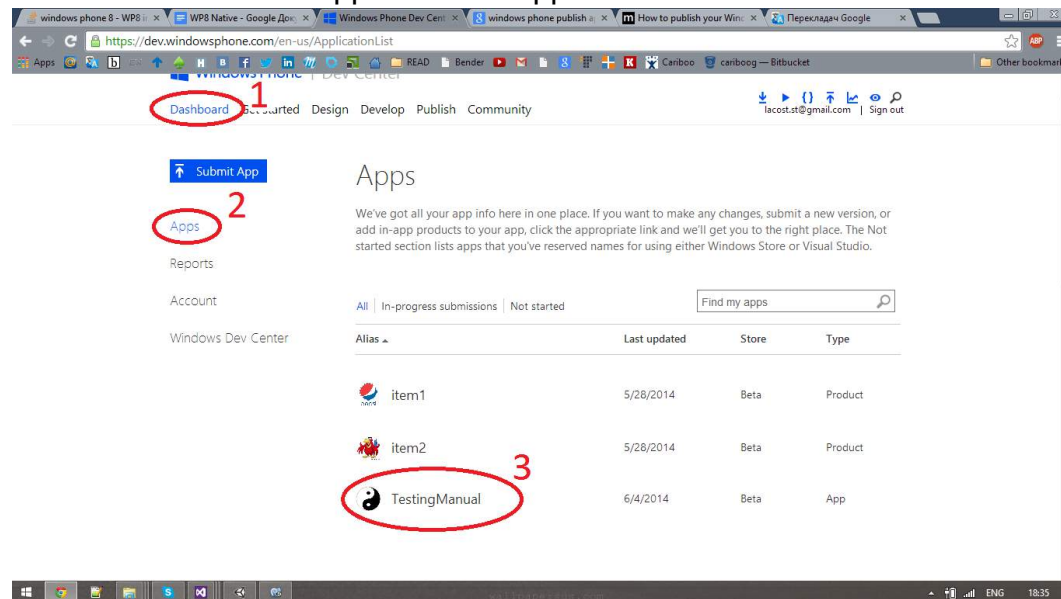


If you did everything right, you will see message like: “Your submission will go live within 2 hours. We’ll send you an email once it’s been processed.”

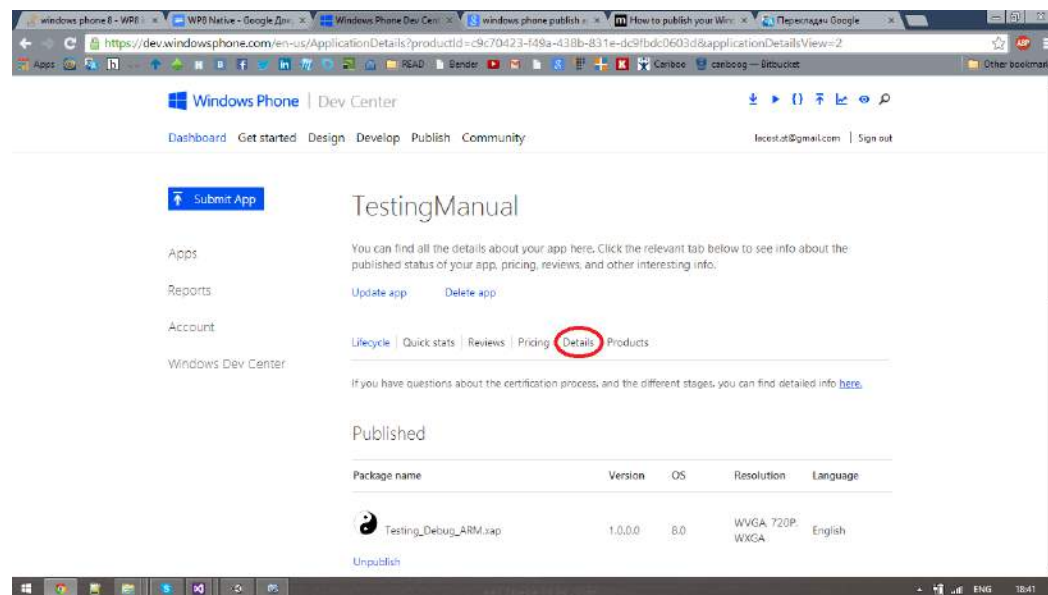
In around two hours our app will be ready.

We can set correct application ID now.

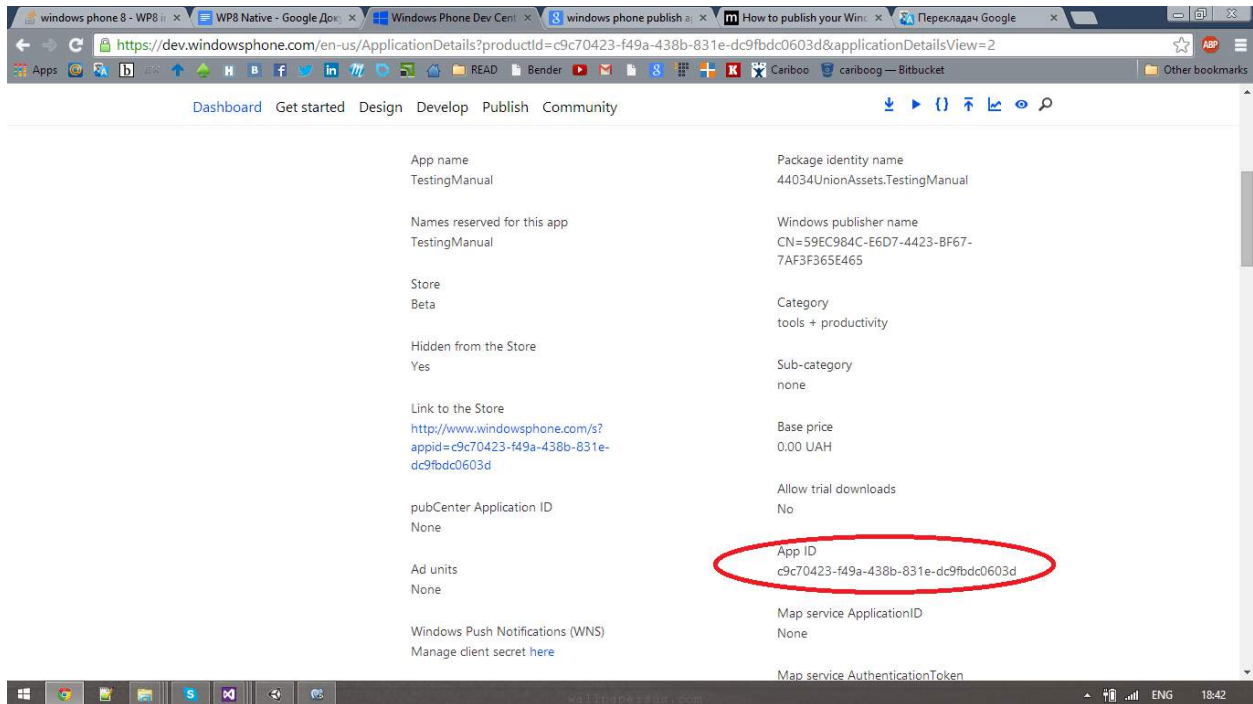
Select Dashboard → Apps → Your Application



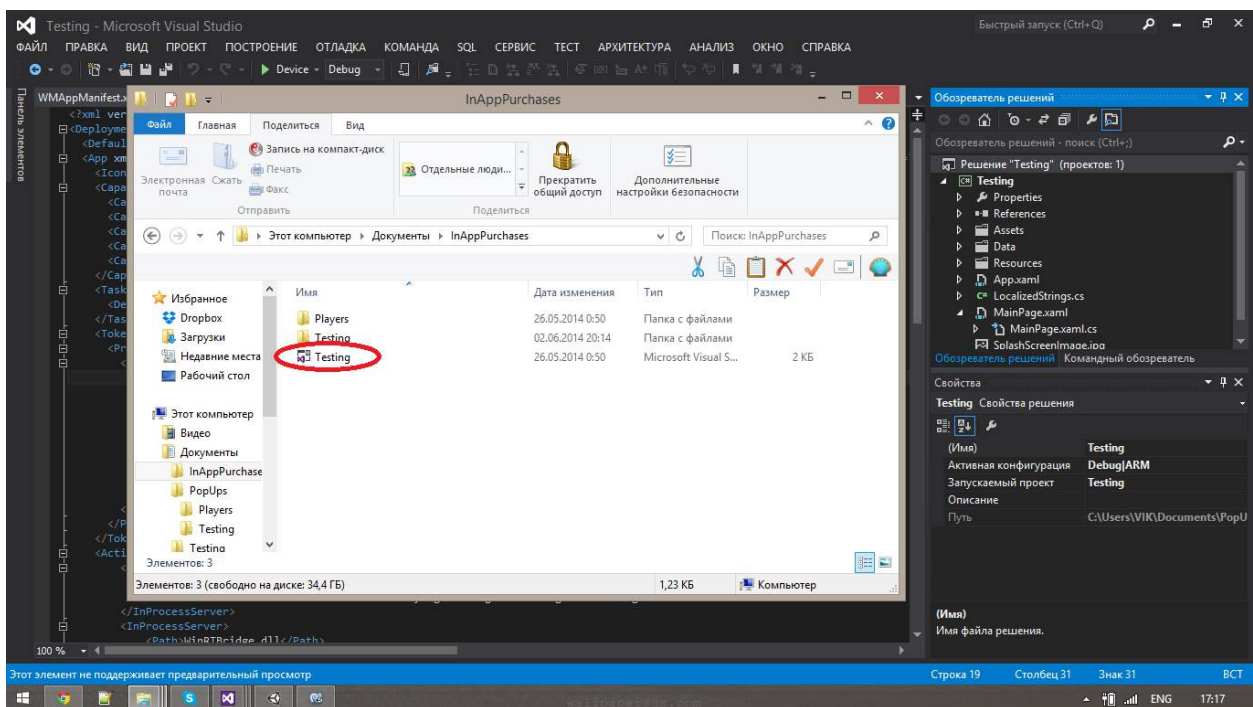
Select Details tab



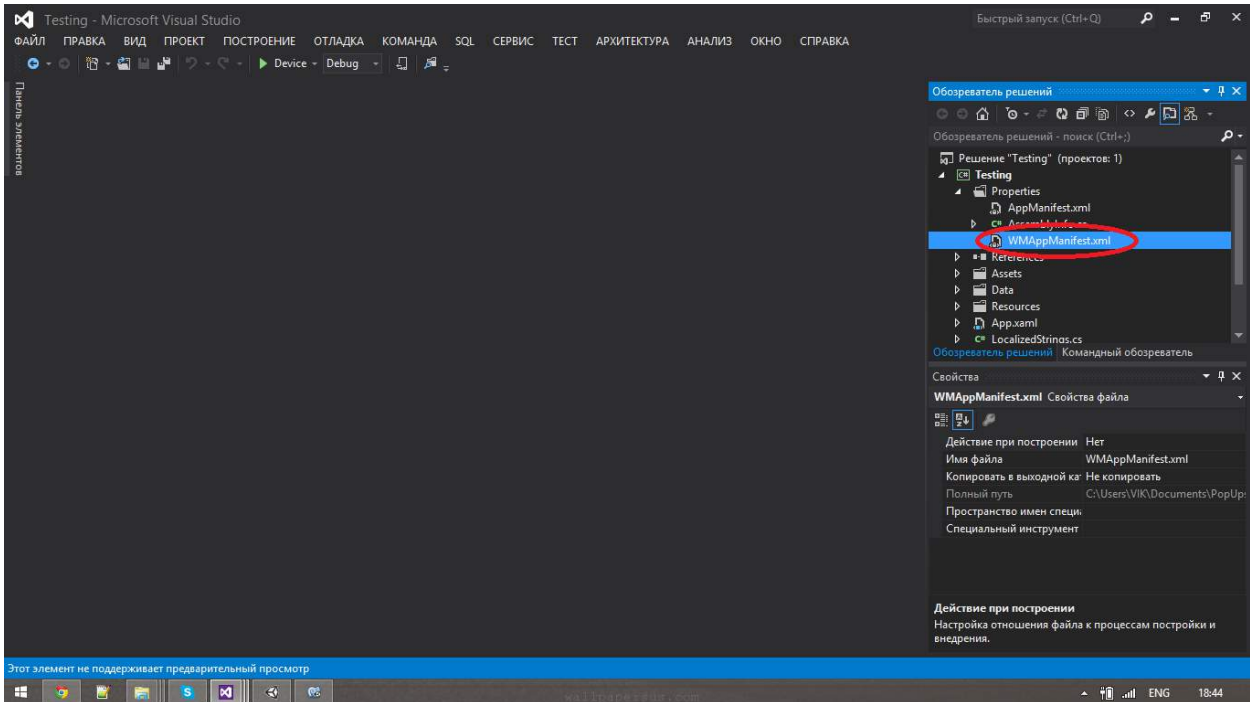
You can see your app ID here



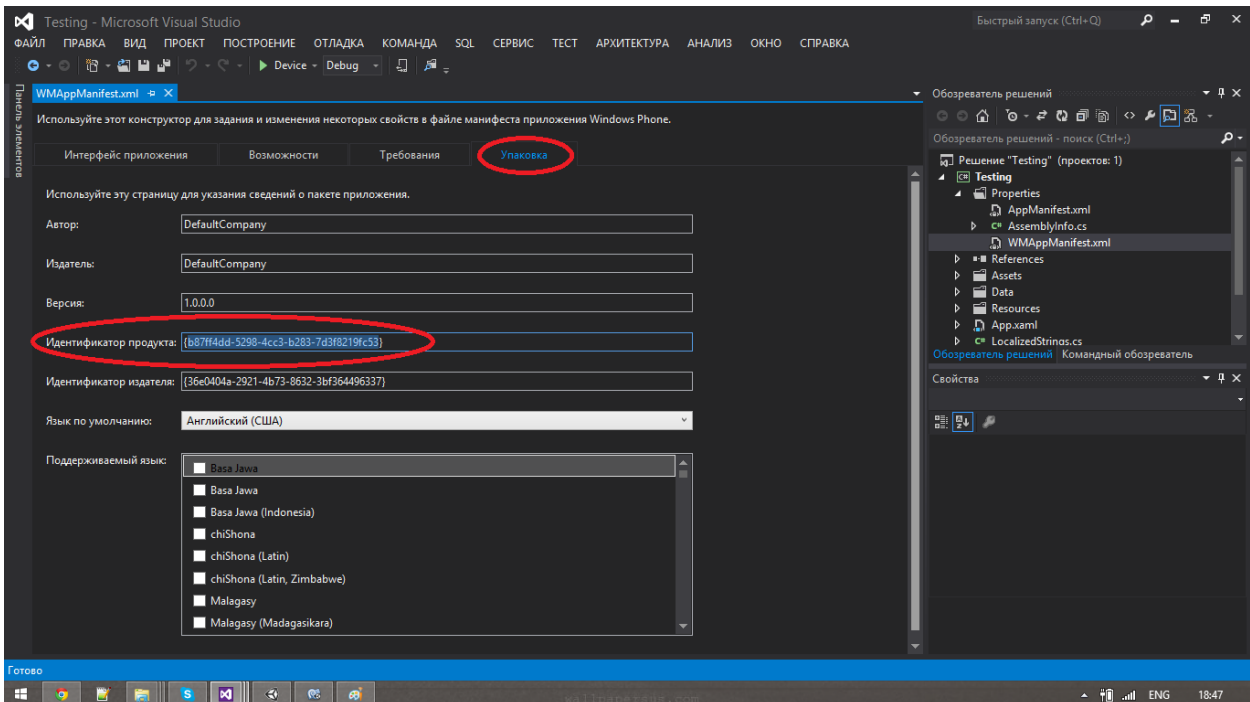
Copy that, then open build, and run Visual Studio Solution



Double click on WMAAppManifest.xml



then click package and paste your ID in field product ID between {...} and save project (ctrl+s).



Now your application is ready for IAP testing.

But we need to add purchase products first.

Click Dashboard → Apps → Your Application

The screenshot shows the Windows Phone Dev Center interface. The top navigation bar includes 'Dashboard', 'Get started', 'Design', 'Develop', 'Publish', and 'Community'. The left sidebar has 'Submit App', 'Apps', 'Reports', 'Account', and 'Windows Dev Center'. The main content area is titled 'Apps' and shows a list of applications. The 'TestingManual' app is highlighted with a red circle and a red '3'.

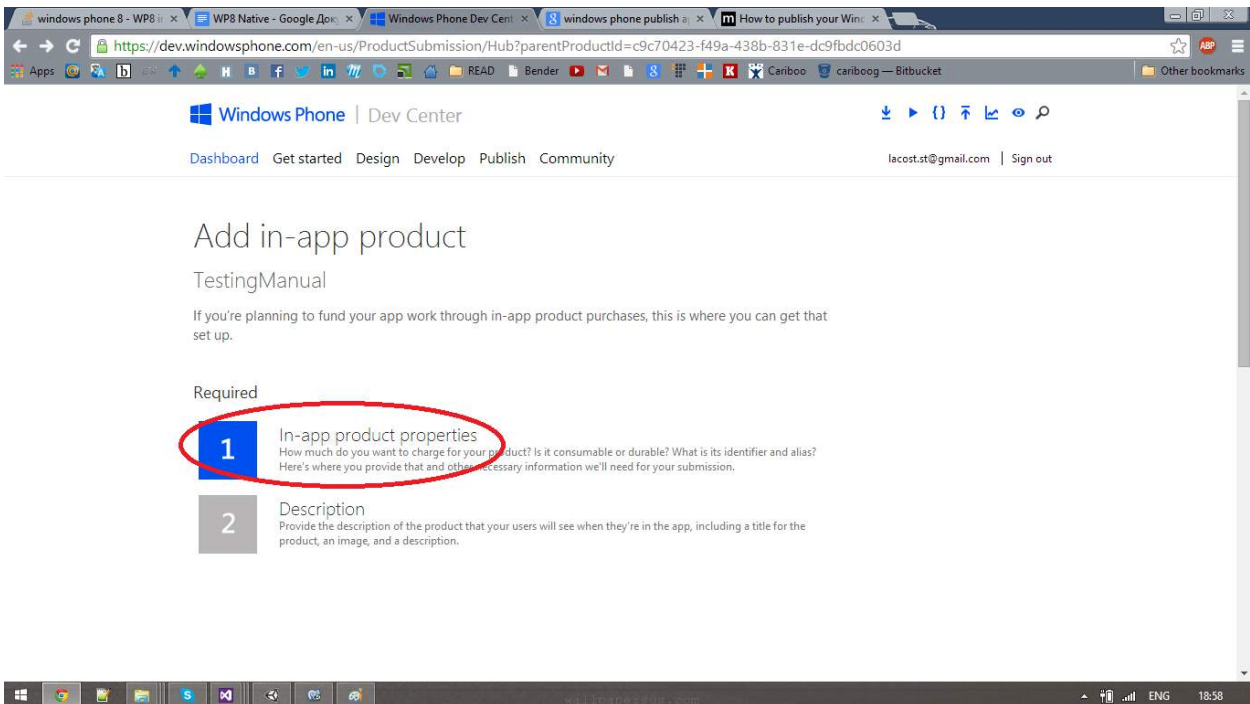
| Alias | Last updated | Store | Type |
|---------------|--------------|-------|---------|
| item1 | 5/28/2014 | Beta | Product |
| item2 | 5/28/2014 | Beta | Product |
| TestingManual | 6/4/2014 | Beta | App |

Select **Products** tab and click **Add in-app product**

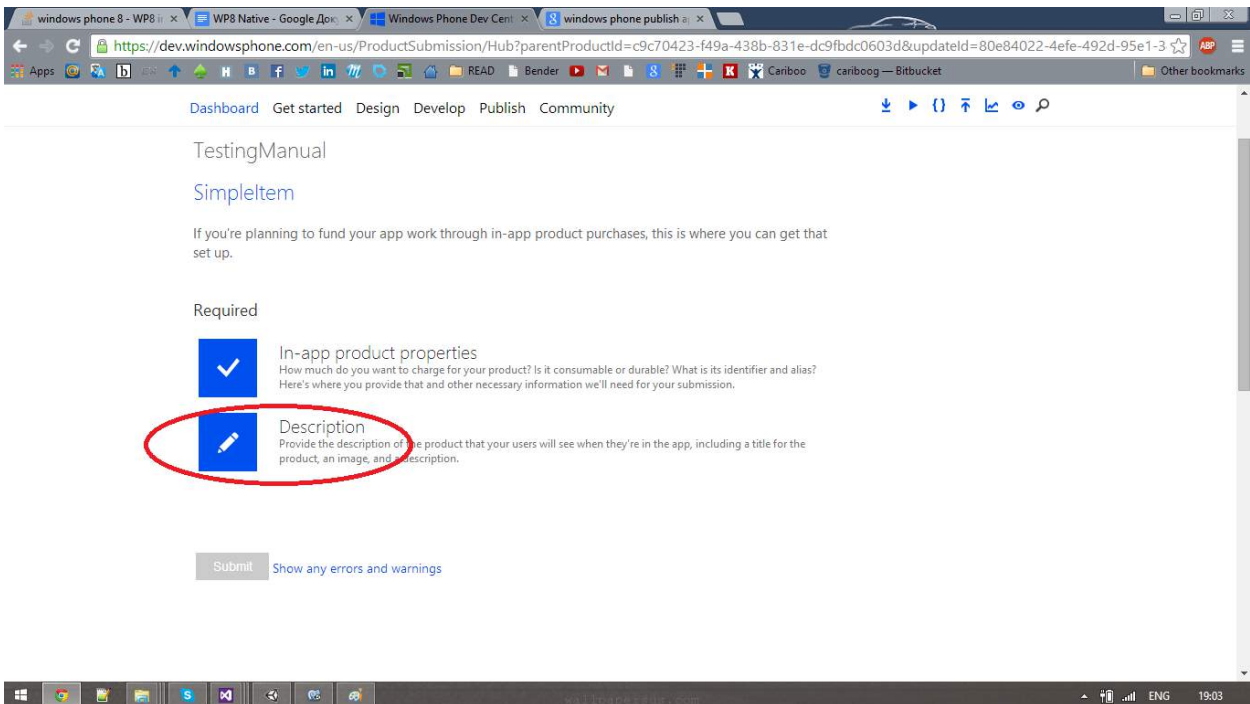
The screenshot shows the 'TestingManual' app details page. The navigation bar includes 'Lifecycle', 'Quick stats', 'Reviews', 'Pricing', 'Details', and 'Products'. The 'Products' tab is highlighted with a red circle. The main content area has a section titled 'Add in-app product' which is also circled in red.

[Add in-app product](#)

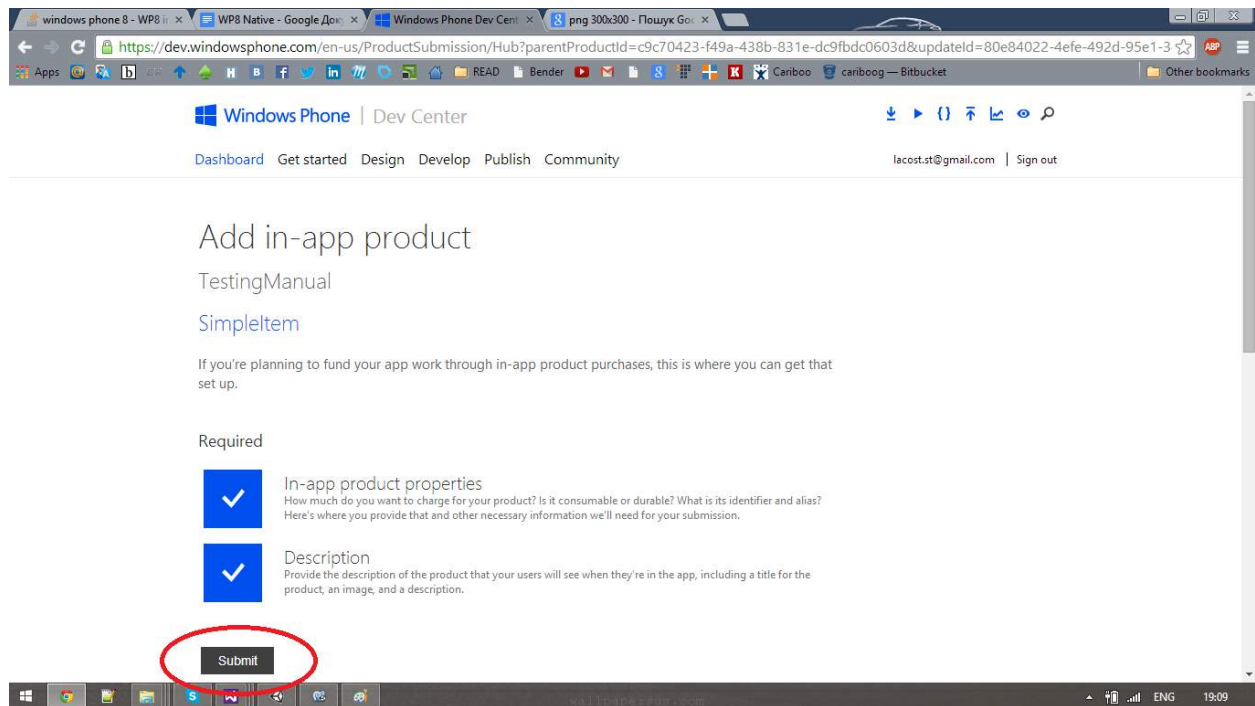
go to In-app product properties and fill all fields with * symbol



Then do the same with Description, and don't forget about .png image



Click Submit when you done.



Your products will be available in few hours.

You can now edit **WPN_BillingManagerExample** edit script. Replace placeholders with your items and you are ready for testing the example scene.

```
public const string YOUR_DURABLE_PRODUCT_ID_CONSTANT = "item2";  
public const string YOUR_CONSUMABLE_PRODUCT_ID_CONSTANT = "item1";
```

API Reference

WP8InAppPurchasesManager.

Loads store data. Will trigger INITIALIZED event when data is ready

```
public void init()
```

Initialize purchase flow by product id. Triggers PRODUCT_PURCHASE_FINISHED event when done.

```
public static void purchase(string productId)
```

Getters:

List of products

```
public List<WP8ProductTemplate> products
```

Events:

Fires when store data is successfully loaded

```
INITIALIZED
```

Fires when purchase flow finished. Event contains [WP8PurchaseResponse](#) as result

```
PRODUCT_PURCHASE_FINISHED
```

WP8ProductTemplate.

Will start product image loading sequence PRODUCT_IMAGE_LOADED event will be fired when done.

```
public void LoadProductImage()
```

Getters:

product image url

```
public string ImgURL
```

product name

```
public string Name
```

product id

```
public string ProductId
```

product localized price

```
public string Price
```

product type

```
public WP8PurchaseProductType Type
```

product description

```
public string Description
```

true if durable product was already purchased. false In all other cases.

```
public bool isPurchased
```

Loaded product texture

```
public Texture2D texture
```

Events:

Fires when product texture is loaded

PRODUCT_IMAGE_LOADED

Fires when purchase flow finished. Event contains [WP8PurchaseResponse](#) as result

PRODUCT_PURCHASE_FINISHED

WP8PurchaseResponse.

Getters:

product id

public string productId

true if product was successfully purchased.

public bool IsSuccess

purchase receipt. May be used for server side verification

public string receipt

Error message if purchase is failed

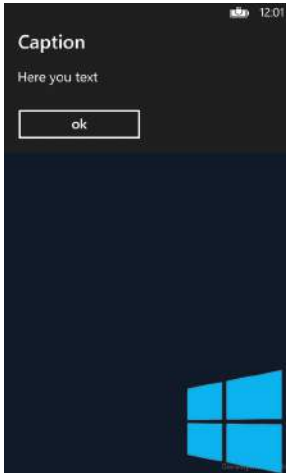
public string error

Native Pop-ups

Showing message pop-up

```
WP8Message msg = new WP8Message("Message Titile", "Message message");
```

Result for this API call is showed below:



If you need to find out when message is closed you should add listener:

```
msg.addEventListener(BaseEvent.COMPLETE, OnMessageClose);
```

OnMessageClose function will be called as soon as pop-up is closed.

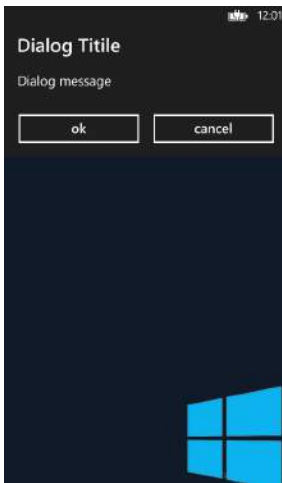
OnMessageClose function example:

```
private void OnMessageClose(CEvent e) {  
    //removing listner  
    e.dispatcher.removeEventListener(BaseEvent.COMPLETE, OnMessageClose);  
    new WP8Message("Result", "Message Closed");  
}
```

Showing dialog pop-up

```
WP8Dialog dialog = new WP8Dialog("Dialog Titile", "Dialog message");
```

Result for this API call is showed below:



If you need to find out dialog result you should add listener:

```
dialog.addEventListener(BaseEvent.COMPLETE, OnDialogClose);
```

OnDialogClose function will be called as soon as pop up is closed.

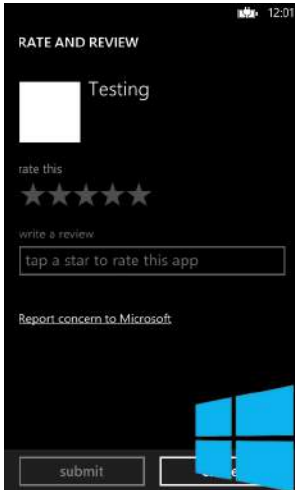
OnDialogClose function example:

```
private void OnDialogClose(CEvent e) {  
    //removing listner  
    e.dispatcher.removeEventListener(BaseEvent.COMPLETE, OnDialogClose);  
    //parsing result  
    switch((WP8DialogResult)e.data) {  
        case WP8DialogResult.YES:  
            Debug.Log ("Yes button pressed");  
            break;  
        case WP8DialogResult.NO:  
            Debug.Log ("No button pressed");  
            break;  
    }  
}
```

Showing rate pop-up

```
WP8RateUsPopUp ratePopUp = new WP8RateUsPopUp("Like this game?", "Please rate to support future updates!");
```

Dialog pop up will be created, but Yes option will redirect to the rating page as on screenshot below



If you need to find out pop-up result you should add listener:

```
ratePopUp.addEventListener(BaseEvent.COMPLETE, OnRatePopUpClose);
```

OnRatePopUpClose function will be called as soon as pop up is closed.

OnRatePopUpClose function example:

```
private void OnRatePopUpClose(CEvent e) {  
    //removing listner  
    e.dispatcher.removeEventListener(BaseEvent.COMPLETE, OnRatePopUpClose);  
    //parsing result  
    switch((WP8DialogResult)e.data) {  
        case WP8DialogResult.RATED:  
            Debug.Log ("Rate Option pickied");  
            break;  
            break;  
        case WP8DialogResult.DECLINED:  
            Debug.Log ("Declined Option pickied");  
            break;  
    }  
}
```


PlayMaker Actions

As alternative to the coding you can use Playmaker actions.

Actions can be found in zip archive under:

Assets/Extensions/WP8Native/Addons/PlayMakerActions.zip

After extracting action files, you can use native pop-ups actions. In action browser actions can be found under the **WP8 Native** tab

Here is list of actions currently available with the plugin:

Billing

- WPN_initBilling
- WPN_Purchase

PopUps

- WPN_DialogPopup
- WPN_MessagePopup
- WPN_RatePopup