Infantium Documentation

Release 1.5.0.0

Infantium

February 19, 2014

More about the project at http://www.infantium.com/

CHAPTER 1

Contents:

1.1 Android Walkthroughs

Infantium Android SDK, current version 1.5, allows Developers to contact with the Infantium API more easily.

Here you can see the walkthroughs for creating and sending e-book and game *RawData* to the API of Infantium.

1.1.1 Walkthrough: E-Book

Date February 19, 2014

Revision 1.5.0.0

Description This tutorial show how to send an e-book rawdata, detailing the different required functions with its parameters.

Introduction

Dependencies

This SDK requires the library LoopJ v1.4.3 to be included in the project for proper functioning. You can find it in the following link: LoopJ Library 1.4.3

Android Permissions

The present SDK requires the following Android Permissions in the App manifest:

- INTERNET
- ACCESS_NETWORK_STATE

Singleton Pattern

Infantium_SDK has been created using a Singleton pattern, so the only way to get an instance of the class is by calling the function: getInfantium_SDK(Context context). The SDK requires the *Context* of the Android Activity.

Example:

Infantium_SDK infantium = Infantium_SDK.getInfantium_SDK(this.getBaseContext());

The Handler

As most functions work asynchronously, an HttpHandler must be implemented. In order to simplify this task Infantium created a class called *InfantiumAsyncResponseHandler* that provides the methods that must be implemented by the Developers. Its methods are called after making a request to the API, such as creating a new player, getting logged or sending an e-book rawdata.

Here we can see an example of how to implement an *InfantiumasyncResponseHandler*:

```
InfantiumAsyncResponseHandler infantiumHandler = new InfantiumAsyncResponseHandler() {
                @Override
                public void onSuccessCloseGameplay(){
                        System.out.println("---Gameplay closed successfully---");
                }

                @Override
                public void onFailureCloseGameplay(String description){
                        System.out.println(description);
                }
};
```

In this example, if the user creates the *Gameplay* successfully, the SDK will call the *onSuccessCloseGameplay()*. If a problem comes up, it will call the *onFailureCloseGameplay(String description)*. The description is a short descriptive message that explains where or why the problem has arised.

Walkthrough

1. Configure the SDK

First of all, we have to configure the SDK with some data. This data will be, on one hand, the developer API credentials for contacting with Infantium. The other function we should call is the *setDeviceInfo()* function in order to set the current device pixels

Function:

setDeveloperCredentials(String api_user, String api_key)
setDeviceInfo(w_dev, h_dev)
setDeveloperHandler(InfantiumAsyncResponseHandler handler)

2. Set e-book ContentApp UUID

We have to set the ContentApp UUID of the e-book before creating a gameplay.

Function:

setContentAppUUID(String contentapp_uuid)

Possible responses:

- *onSuccessContentApp()*: the contentapp UUID is found in the Infantium market.
- *onFailureContentApp(String description)*: a problem occurred when trying to obtain the contentapp info from the market.

3. Set e-book Content UUID

This function will set the ebook content UUID, required before creating a new Gameplay.

Function:

setContentUUID(String ebook_content_uuid)

If the content is not correct, the error will appear when calling the *sendEbookRawdata*.

4. Get Player UUID (from the Infantium App):

This function will get the UUID of the selected player in the InfantiumApp to be used by the SDK. This requires to add a few lines in the Android Manifest of the App adapting to Infantium. The following receiver should be added to the Manifest:

```
<receiver android:name="com.infantium.android.sdk.ReceivePlayer">
    <intent-filter>
        <action android:name="com.infantium.android.sdk.ReceivePlayer"></action>
    </intent-filter>
</receiver>
```

This receiver should be added inside of the <application> tag of your Manifest. Once this is added, the call to get the Player (and this is the step 3) is:

Function:

getPlayerUUIDFromApp()

Possible responses:

- *onSuccessGetPlayerByUUID()*: Player was successfully obtained, you can now proceed to the next step.
- *onFailureGetPlayerByUUID(String description)*: A problem occurred while obtaining the player, check the description for more details.

5. Create Gameplay:

When we have set the *contentapp_uuid*, *content_uuid* and the *player_uuid* we can create a gameplay.

Function:

createGameplay()

Note: the createGameplay(String subcontent_uuid, handler) is only used to create gameplays of games.

Possible responses:

- *onSuccessCreateGameplay()*: The gameplay is created successfully.
- *onFailureCreateGameplay(String description)*: If the player is not selected, the content is not informed or there is another gameplay opened

6. Rawdata Functions:

Once the gameplay is created, we can call the rawdata functions to introduce elements or sounds. Additionally, when the ebook page is shown (the kid can see the objects in the screen), the function startPlaying() should be called. If any new elements, sounds or animations are displayed they can be added afterwards.

- Required rawdata functions:
- addElement(Element element)
- addElements(List<Element> elements)
- tapNoObjects(List<Integer> position)
- tapNoObjects(List<Integer> position, String sound_id)
- tapOnObjects(String element_id)
- tapOnObjects(String element_id, String sound_id)
- setSuccesses(int successes)
- setFailures(int failures)
- Optional rawdata functions:
- setTarget(Target target)
- setTargets(List<Target> targets)
- setEvaluate(List<String> eval)
- addSound(Sound sound)
- addSounds(List<Sound> sounds)
- addFixedAnimation(Animation animation)
- addFixedAnimations(List<Animation> animations)
- addDynamicField(DynamicField d_field)
- addDynamicFields(List<DynamicField> d_fields)
- startAnimation(String element_id, List<Integer> st_pos, String type)
- endAnimation(String element_id)
- endAnimation(String element_id, List<Integer> end_pos)
- endAnimation(String element_id, String sound_id, List<Integer> end_pos)
- startDragging(String element_id, List<Integer> position)
- finishDragging(List<Integer> position)
- finishDragging(List<Integer> position, int max_x, int max_y)

- finishDragging(List<Integer> position, String sound_id)
- finishDragging(List<Integer> position, String sound_id, int max_x, int max_y)

7. Send Ebook Rawdata:

We finally call this function when we want to send the rawdata.

Function:

sendEbookRawData(int numPage, boolean text, boolean readToMe, final InfantiumAsyncResponseHandler responseHandler)

- numPage: The number of the page in the e-book.
- text - true if the page contains text or false if not.
- readToMe - true if the book reads to the player or false if not.

Possible responses:

- *onSuccessEbookRawdata()*: The ebook rawdata is posted successfully.
- *onFailureEbookRawdata(String description)*: A problem occurred when sending the ebook rawdata.

8. Close Gameplay

Last step but not least important. If the gameplay is not closed, the SDK will not be able to create new Gameplays.

Function:

closeGameplay(InfantiumAsyncResponseHandler handler)

Possible responses:

- *onSuccessCloseGameplay()*: Gameplay closed succesfully.
- *onFailureCloseGameplay(String description)*: If the gameplay is not started or another problem occurs when closing the gameplay.

1.1.2 Walkthrough: Game

Date February 19, 2014

Version 1.5.0.0

Description This tutorial shows how to send a game rawdata, detailing the different required functions with its parameters.

Introduction

Dependencies

This SDK requires the library LoopJ v1.4.3 to be included in the project for proper functioning. You can find it in the following link: LoopJ Library 1.4.3

Android Permissions

The present SDK requires the following Android Permissions in the App manifest:

- INTERNET
- ACCESS_NETWORK_STATE

Singleton Pattern

Infantium_SDK has been created using a Singleton pattern, so the only way to get an instance of the class is by calling the function: getInfantium_SDK(Context context). The SDK requires the *Context* of the Android Activity.

Example:

Infantium_SDK infantium = Infantium_SDK.getInfantium_SDK(this.getBaseContext());

The Handler

As most functions work asynchronously, an HttpHandler must be implemented. In order to simplify this task Infantium created a class called *InfantiumAsyncResponseHandler* that provides the methods that must be implemented by the Developers. Its methods are called after making a request to the API, such as creating a new player, getting logged or sending an e-book rawdata.

Here we can see an example of how to implement an *InfantiumasyncResponseHandler*:

```
InfantiumAsyncResponseHandler infantiumHandler = new InfantiumAsyncResponseHandler() {
                @Override
                public void onSuccessCloseGameplay(){
                        System.out.println("---Gameplay closed successfully---");
                }

                @Override
                public void onFailureCloseGameplay(String description){
                        System.out.println(description);
                }
};
```

In this example, if the user creates the *Gameplay* successfully, the SDK will call the onSuccessCloseGameplay(). If a problem comes up, it will call the onFailureCloseGameplay(String description). The description is a short descriptive message that explains where or why the problem has arised.

Walkthrough

1. Configure the SDK

First of all, we have to configure the SDK with some data. This data will be, on one hand, the developer API credentials for contacting with Infantium (setDeveloperCredentials(String api_user, String api_key)). The other function we should call is the setDeviceInfo(w_dev, h_dev) function in order to set the current device pixels. Finally, if you want to receive feedback from the SDK, you will need to implement a Handler. This will be further explained in the Handler section. You can do that with the setDeveloperHandler(InfantiumAsyncResponseHandler handler) function.

```
// Configure the SDK
infantium.setDeveloperCredentials(api_user, api_key);
infantium.setDeviceInfo(w_dev, h_dev);
infantium.setDeveloperHandler(handler);
```

2. Set Game ContentApp UUID

The next step if to set the *contentapp_uuid*, which will identify the App against the server. To configure it the method setContentAppUUID(String contentapp_uuid) should be used.

```
// Set the App contentapp_uuid
infantium.setContentAppUUID(contentapp_uuid);
```

3. Create Gameplay:

When we have set the *contentapp_uuid* we can create a *Gameplay* with: createGameplay(String subcontent_uuid). The *subcontent_uuid* will be provided to you by Infantium, which will be a unique identifier for your activity.

```
// Send the previously introduced data
infantium.createGameplay(subcontent_uuid);
```

4. Rawdata Functions:

The *GamePlay* is created once everytime the kid starts a game session. Now, for every activity played during that time, a *RawData* object is sent, which will contain the information we need to analyze. This contains, among other generic stats, the elements in the screen, the actions the kid performs, and some info about the results.

When the kid enters one of the activities of the game (i.e. starts playing the game), the *RawData* is filled in three phases:

1. Register the elements in the screen.

   This is done adding the Elements in the screen (addElement(Element element)) and the Sounds if there are any (addSound(Sound sound)).

   An example element and sound could be:

   ```
   // Add an element for a dog drawing
   List<Integer> size = Arrays.asList(10, 10);
   List<Integer> pos = Arrays.asList(20, 20);
   Element dog_element = new Element(
       "dog_figure",
       size,
       "[0,255,0]",
   ```

```
    "animal",
    "dog",
    pos
);
infantium.addElement(dog_element);

// Add the sound the dog makes when it is tapped
Sound dog_sound = new Sound(
    "barking"
);
infantium.addSound(dog_sound);
```

Once registered, it is very important to point out the elements to evaluate on the screen (there may be different elements, but only a few important for the activity):

```
// Add an element for a dog drawing
List<String> evaluate = Arrays.asList("dog_figure");
infantium.setEvaluate(evaluate);
```

2. Start the timers and register the actions of the kid.

   When the kid starts interacting with the screen, we will call the startPlaying() method. This will trigger the timers inside the SDK. The SDK will automatically handle the timestamps when the kid taps the screen and the elements show up, which will allow us to get a lot of statistics about the child's development, with no effort at all on the developer side.

   For each time the kid taps on the screen, this will be registered with the tapOnObjects(String element_id) method. In this method, it must be pointed out if the interaction represents a *success*, an *error* or *none* of both. Here is an example for the previous *dog* with its sound:

   ```
   // Tapping the dog is the goal of the activity, and thus is represented a "success". When the do
   //  the "barking" sound is triggered.
   infantium.tapOnObjects("dog_figure", "success", "barking");

   // Another example, if the kid taps on the "cat_figure" element, but was not the goal of this ac
   infantium.tapOnObjects("cat_figure", "error", "error_sound");
   ```

3. Add some general info about the scores.

   When the kid has completed the activity, some conclusions about the activity are registered. This is done with the setSuccesses(int successes) and setFailures(int failures) methods.

   ```
   // Finally one "success" and one "failure"
   infantium.setSuccesses(1);
   infantium.setFailures(1);
   ```

5. Send Game Rawdata:

We finally call sendGameRawData() when we want to send the *RawData*. After sending the data, and the kid starts a new activity, the flow would go again to the 4th step! If the kid goes back to the main menu, proceed to step 6.

```
// Send the previously introduced data
infantium.sendGameRawData();
```

6. Close Gameplay

Last step but not least important: closeGameplay(). If the *GamePlay* is not closed, the SDK will not be able to create new ones.

7. Conclusions

And with this

1.2 Adding the Infantium Button

Date February 19, 2014

Revision 1.5.0.0

Description This section shows how to add the Infantium Button to your App.

1.2.1 Downloading

The button graphics may be downloaded from the following link. Here is an example of the button:

1.2.2 Positioning

The Infantium button should be placed in the main screens of the game, this is, where the user selects which games to play. It should also be included in the landing screen where the user first stops after launching the game.

1.2.3 Calling the SDK

The function of the SDK to be called is returnToInfantiumApp() with the current activity as the only parameter. This will open the Infantium App from which your App was called.