Infantium Documentation

Release 2.3.0.0

Infantium

April 24, 2014

Contents

Infantium's SDK is specially focused to adapt mobile applications to work and communicate with Infantium servers. It's purpose is to provide an intuitive and easy to use interface to communicate your App with Infantium's cognitive analysis platform.

Including our SDK in your project will provide you with a whole set of tools to register the activity in your content (be it a game, ebook, etc.), while having the privacy of the kid as our first goal. Additionally, we will provide developers with a dashboard of usage statistics, and many more features to come, like in-game cognitive adaptivity!

The SDK is programmed in native Android/Java, but is also deployed in other frameworks through a ever increasing set of different plugins. Check the *Plugins* page to check if your framework has already been adapted!

If you want to connect your content with Infantium, and haven't contacted with us yet, send us an email and we will explain you all the advantages of becoming a part of the most intelligent cognitive platform for kids in the world! You can find us at our mail address. We're waiting for you!

CHAPTER 1

Contents:

1.1 Downloading, installing and configuring the SDK

Welcome to the set-up area of the Dev Center, in the following sections we will explain in detail how to download the requirements, as well as install and configure the Infantium SDK.

1.1.1 Installation

As the SDK is delivered as a *jar* file, it is very easy to add to your project. Simply add it to your project in a folder named *libs* and reference it from the build path. Don't forget to add the *Dependencies* in the same folder where the SDK is!

If you're using a specific framework instead of developing in native Android, you may have to add the plugin instead of the SDK provided in the section before. Please refer to the *Appendix: Plugins* section for more info.

Check installation

To check the installation has worked properly, import the SDK from your App and check the version attribute of the SDK:

```
// Import the Android SDK
import com.infantium.android.sdk.InfantiumSDK;

// Inside your code get the version of the SDK
Log.i(LOG_TAG, "Checking Infantium Installation! SDK Version: " + InfantiumSDK.version);
```

Docs version: 2.3 Last update: April 24, 2014

1.1.2 Download

The Infantium SDK is provided as a *jar* container, that is, as a Java library. Contact us to get the latest release!

Dependencies

This SDK requires the *LoopJ Asynchronous HTTP Library v1.4.3* to be included in the project for proper functioning. You can download it from our servers or from LoopJ's github repo .

Docs version: 2.3 Last update: April 24, 2014

Docs version: 2.3 Last update: April 24, 2014

1.2 Getting Started

So what now? When speaking about integrating a game with Infantium many developers get a little bit confused. Integrating a game with Infantium means communicating with our SDK at different points of your Apps, feeding it with three different core sources of information, which are where we extract the data for our Cognitive Analysis:

- The *Elements* on the screen.
- The *Goals* (or objectives) that the kid must achieve.
- The actual *Interactions* that happen during game time.

To do this there are a series of methods to be called at different parts of your code. For instance, the Elements and Goals will probably be declared at the beginning of your game scenes (when the visual aspects of the current activity are declared), whereas the Interactions will very likely be captured at the Event handlers your code may implement to decide whether a kid has performed correctly an activity or not.

To make the smoothest first contact between the developers and the SDK, we have prepared two Walkthroughs which will drive you through the basics of the full integration of an App. Additionally, all the apps would have to implement also both the *Adding the Infantium Button* and the *Overriding the Android Cycle* guides.

Check out the following guides on how to integrate E-Books and Games!

1.2.1 Walkthrough: Game

Date April 24, 2014

Version 2.3

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

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

```
InfantiumSDK infantium = InfantiumSDK.getInfantiumSDK(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.

```
// Create the Gameplay with the SubContent UUID
infantium.createGameplay(subcontent_uuid);
```

4. Rawdata Functions:

The *GamePlay* is created once every time 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 and the goals to achieve, and finally the actions the kid performs.

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 Goals the kid has to complete to succeed in this game (addGoal(Goal goal)).

   An example *Element* could be:

   ```
   // Add an element for a dog
   PaintedElement dog_element = new PaintedElement("dog_figure");
   infantium.addElement(dog_element);

   // A ball
   PaintedElement ball_element = new PaintedElement("ball");
   infantium.addElement(ball_element);

   // Add a number element
   NumberElement number_three = new NumberElement(3);
   infantium.addElement(number_three);

   // Add a text element
   TextElement sentence_element = new TextElement("en-US",
       "This little puppy wants to play with the ball! Can you help him?");
   infantium.addElement(sentence_element);
   ```

   An example *Goal* could be:

   ```
   // The Goal is to move the ball to the dog
   Goal g = new Goal("drag_the_ball", "selection");
   infantium.addGoal(g);
   ```

   Please refer to the advanced guides for *Elements* and *Goals* for more detailed information.

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, relieving the developer of that task.

   For each time the kid interacts with the screen, this can be registered with the newBasicInteraction(String t, String object_type, String goal_type) method. In this method, the *t* equals to the type of the interaction, which can be *"success"*, *"failure"*, *"none"* or some others explained in the *BasicInteraction* section.

```
    // Start the timers
    infantium.startPlaying();

    // Dragging the ball to the dog is the goal of the activity,
    //  and thus it is represented a "success".
    InfantiumResponse res = infantium.newBasicInteraction("success", "ball", "drag_the_ball");

    // Another example, if the kid drags the "smartphone" element,
    //  but was not the goal of this activity.
    infantium.newBasicInteraction("error", "smartphone", "drag_the_ball");
```

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 the full cycle for sending data is complete. The integration can be enriched with many more methods and variables, but we hope this gave you an insight of the process to integrate your app with Infantium!

Now you can refer to the *Advanced Guides* section for more info.

Docs version: 2.3 Last update: April 24, 2014

1.2.2 Walkthrough: E-Book

Date April 24, 2014

Revision 2.3

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

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

```
InfantiumSDK infantium = InfantiumSDK.getInfantiumSDK(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 EBook 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. Set EBook Content UUID

The method setContentUUID(String content_uuid) will set the EBook content UUID, required before creating a new Gameplay.

```
// Set the Content's UUID
infantium.setContentUUID(content_uuid);
```

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

4. Create Gameplay:

When we have set the *contentapp_uuid* and the *content_uuid* we can create a *Gameplay* with: createGameplay().

```
// Create the Gameplay
infantium.createGameplay();
```

5. Rawdata Functions:

The *GamePlay* is created once every time the kid starts a reading session. Now, for every activity played or page turned 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 and the goals to achieve, and finally the actions the kid performs.

When the kid enters one of the activities of the ebook (i.e. starts reading the ebook), 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 Goals the kid has to complete to succeed in this game (addGoal(Goal goal)).

   An example *Element* could be:

   ```
   // Add an element for a dog
   PaintedElement dog_element = new PaintedElement("dog_figure");
   infantium.addElement(dog_element);

   // A ball
   PaintedElement ball_element = new PaintedElement("ball");
   infantium.addElement(ball_element);

   // Add a number element
   NumberElement number_three = new NumberElement(3);
   infantium.addElement(number_three);

   // Add a text element
   ```

```
TextElement sentence_element = new TextElement("en-US",
    "This little puppy wants to play with the ball! Can you help him?");
infantium.addElement(sentence_element);
```

An example *Goal* could be:

```
// The Goal is to move the ball to the dog
Goal g = new Goal("drag_the_ball", "selection");
infantium.addGoal(g);
```

Please refer to the advanced guides for *Elements* and *Goals* for more detailed information.

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, relieving the developer of that task.

   For each time the kid interacts with the screen, this can be registered with the newBasicInteraction(String t, String object_type, String goal_type) method. In this method, the *t* equals to the type of the interaction, which can be *"success"*, *"failure"*, *"none"* or some others explained in the *BasicInteraction* section.

```
// Dragging the ball to the dog is the goal of the activity,
//  and thus it is represented a "success".
InfantiumResponse res = infantium.newBasicInteraction("success", "ball", "drag_the_ball");

// Another example, if the kid drags the "smartphone" element,
//  but was not the goal of this activity.
infantium.newBasicInteraction("error", "smartphone", "drag_the_ball");
```

6. Send Ebook Rawdata:

We finally call sendEbookRawData(int numPage, boolean text, boolean readToMe) when we want to send the *Raw-Data*. After sending the data, and the kid starts a new activity/page, 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.sendEbookRawData(1, true, false);
```

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

7. 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.

8. Conclusions

And with this the full cycle for sending data is complete. The integration can be enriched with many more methods and variables, but we hope this gave you an insight of the process to integrate your ebooks with Infantium!

Now you can refer to the *Advanced Guides* section for more info.

Docs version: 2.3 Last update: April 24, 2014

Docs version: 2.3 Last update: April 24, 2014

1.3 Advanced Guides

Feel free to browse this section for learning advanced tricks when interacting with the SDK!

1.3.1 Elements

Elements are the main reference objects in the data sent to the server by the SDK. They can represent persons, animals, objects, numbers, images, text, or any other thing the kid can interact with, either actively or passively. Knowing what the kid can see on the screen at the moment of performing any action is crucial to reach conclusions about recurring patterns in the kids actions, just to name an example.

There are some parameters which are common to all kind of elements. Those are the *size*, *kinetic info* and *color*. All of those attributes are optional, but are very useful to calculate cognitive information about the kid afterwards. Here are some examples:

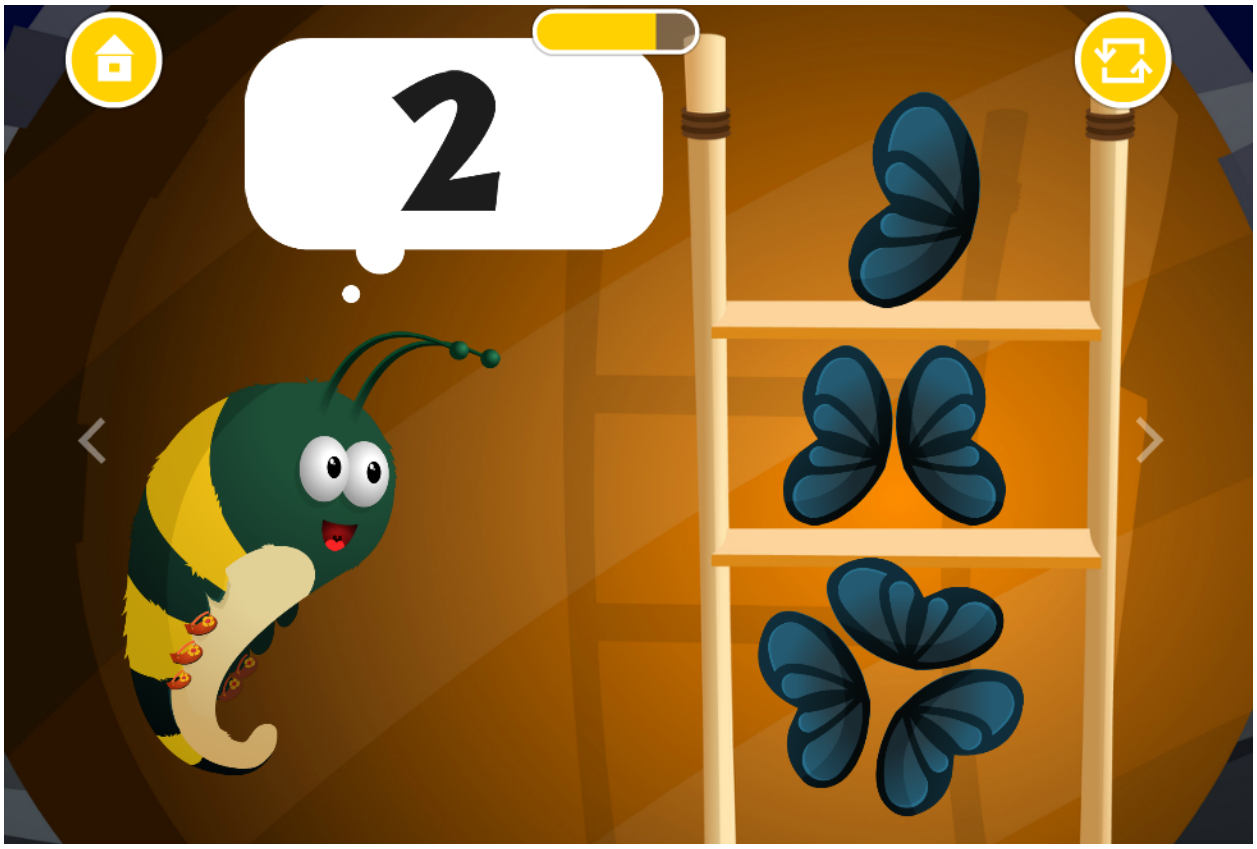
```
// Adding size to an element
Element sized_element = new Element("balloon", 200, 100);
infantium.addElement(sized_element);

// Adding movement to an element
Element moving_element = new Element("balloon");
moving_element.set_movement("circular");
infantium.addElement(moving_element);

// Adding color to an element
Element colored_element = new Element("balloon");
colored_element.setRGBColor(255, 0, 0);
infantium.addElement(colored_element);
```

Subtypes

As there are very different types of elements in the games and ebooks around the market, it is logical to have more than one kind of *Element* in the Infantium SDK. Thus, in the following sections you will find different subtypes of *Elements* which will allow you to better define your items with the least effort.

Number Elements

Number elements are those related to cardinal or ordinal numbers, be it in the form of sets of elements, or the number itself in arabic or text representation. This kind of element is always related to counting something, so it is very easy to differentiate from other kinds of elements. This could be an example of a simple *Number Element*:



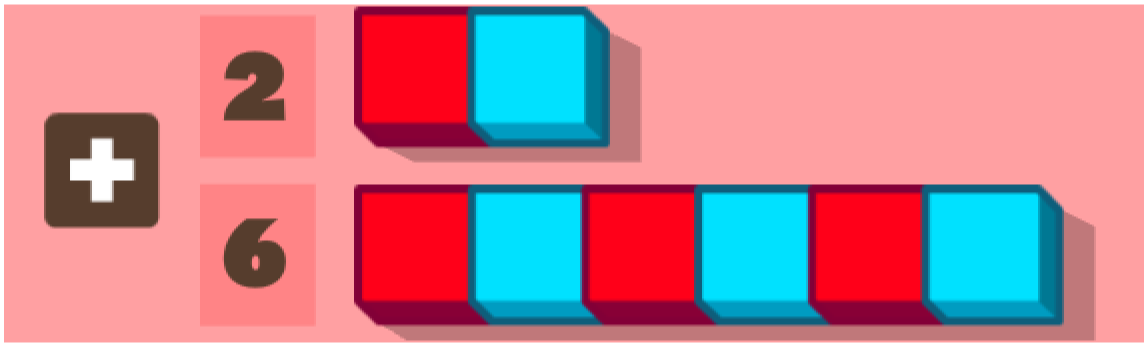
This element could be defined with a *NumberElement*. It takes an optional parameter named 'representations' explained below:

```
// Add a number element with the fewest parameters
NumberElement simple_number_element = new NumberElement("number_two", 2);
infantium.addElement(simple_number_element);

// An example of a full number with the optional parameter
//  'representations'
NumberElement full_number_element = new NumberElement("number_two", 2);
full_number_element.add_representation("arabic");
full_number_element.add_representation("set");
infantium.addElement(full_number_element);
```

The representations parameter adds information to the number element, indicating in which way the number is transmitted to the kid. The possible values can be:

- **arabic:** representation in arabic numbers like 1, 2, 3...
- **roman:** representation in roman numbers like I, II, III...
- **set:** representation as a set of elements which can be counted, like dots, or in the previous example, the wings.
- **length:** representation is also distinguishable by the length of the element. See the example below.

Representations can also have multiple values at the same time, and thus it also accepts Arrays as a parameter. This is better seen in the following example:

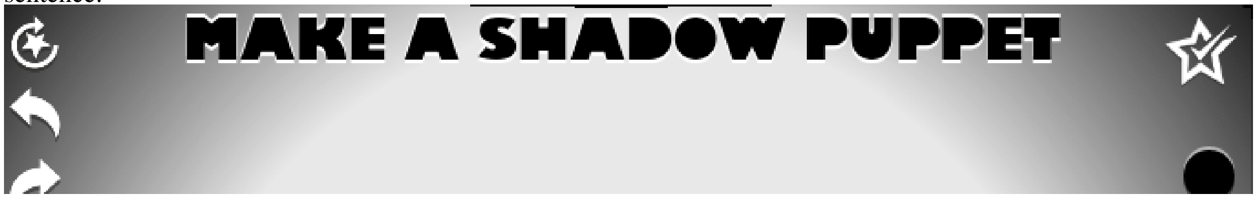


In this case, the element has two representations, one as an *arabic* number, and another one with both a *set* and a *length* representation at the same time:

```
// Full NumberElement
NumberElement full_number_element_2 = new NumberElement("number_two", 2);
full_number_element_2.add_representation("arabic");
full_number_element_2.add_representation(Arrays.asList("set", "length"));
infantium.addElement(full_number_element_2);

NumberElement full_number_element_6 = new NumberElement("number_six", 6);
full_number_element_6.add_representation("arabic");
full_number_element_6.add_representation(Arrays.asList("set", "length"));
infantium.addElement(full_number_element_6);
```

Text Elements

Text elements are very descriptive by themselves. They represent text on the screen, be it a single character or a full sentence:

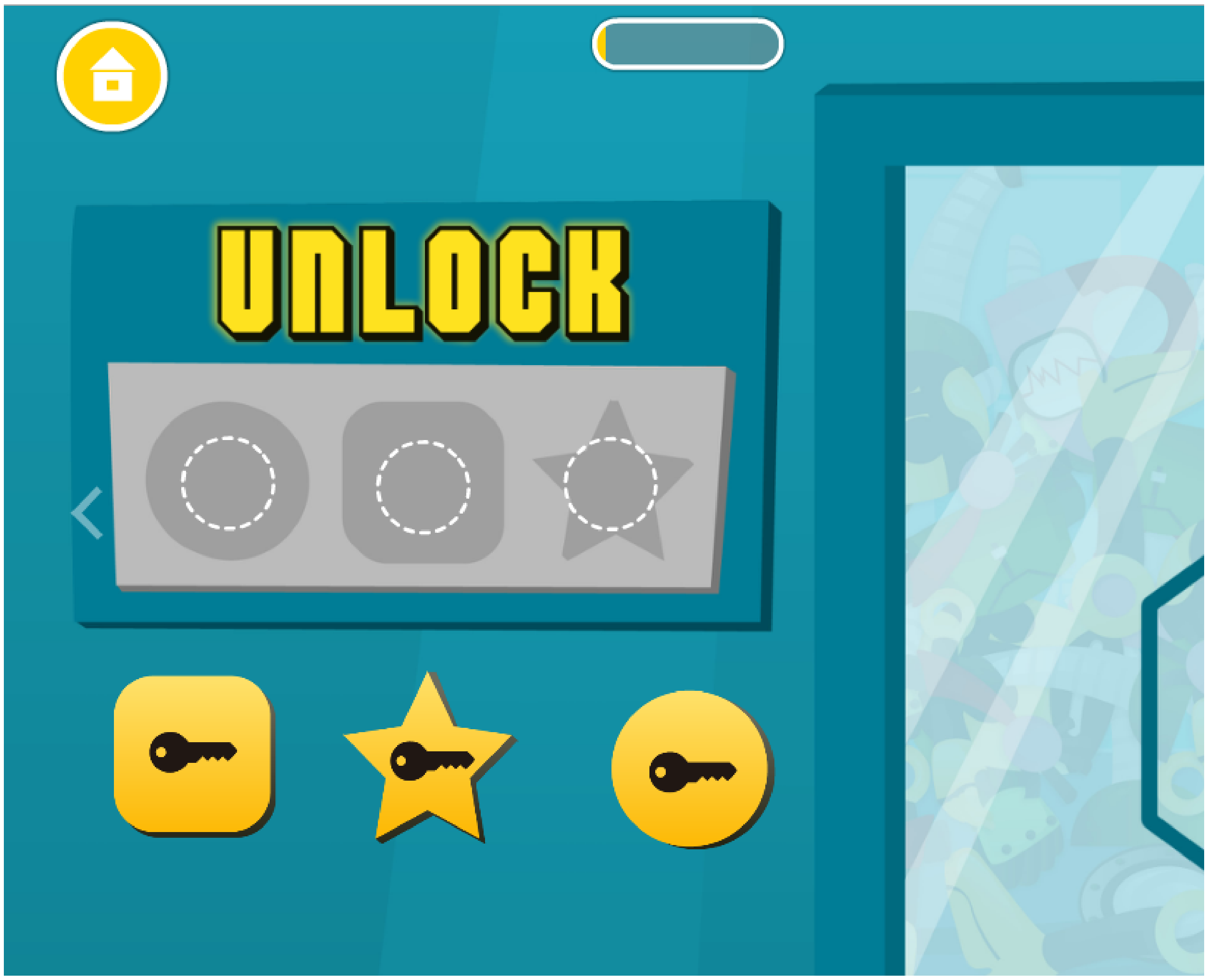


In this case, the upper text of the game would be represented as a *TextElement*:

```
// Add the sentence element
TextElement sentence_element = new TextElement("help_text", "en-US",
    "MAKE A SHADOW PUPPET");
infantium.addElement(sentence_element);

// In another case, we may want to add just one character
TextElement char_element = new TextElement("char_element", "a");
infantium.addElement(char_element);
```

Shape Elements

Shape elements are a special case of object on the screen for polygonal objects, which can fall into several categories, for instance regular or irregular polygons. There is also a special parameter for star-shaped objects.

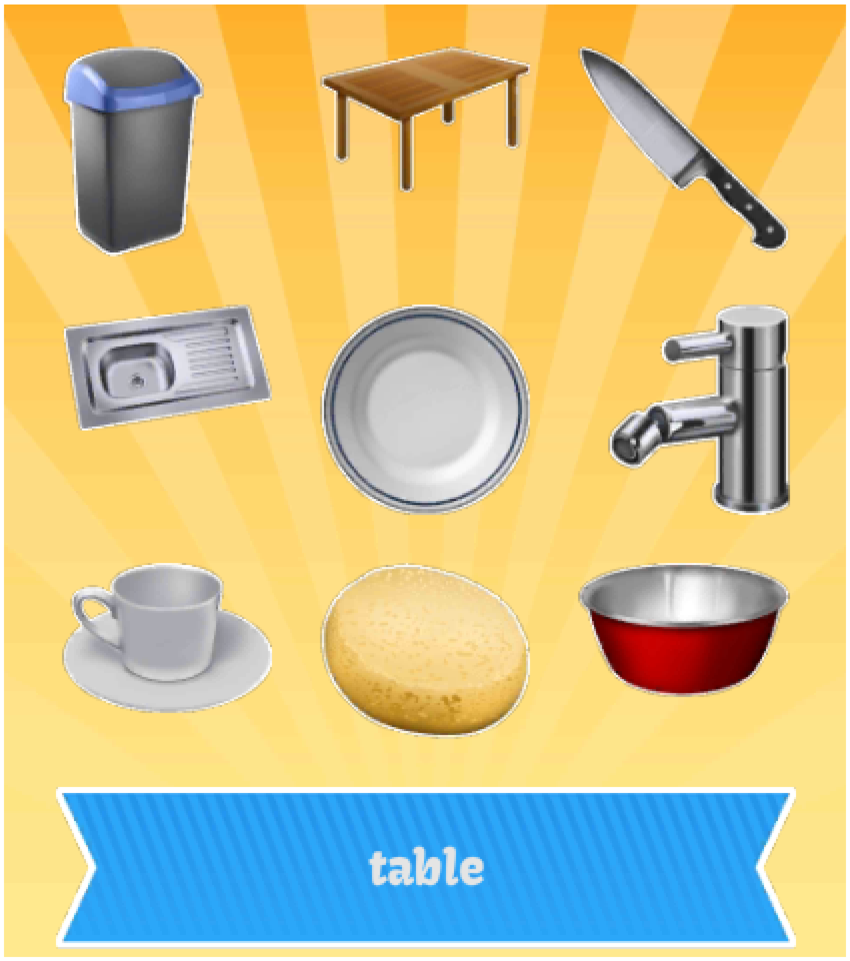


The previous elements could be defined as follows:

```
// One ShapeElement with the basic parameters
ShapeElement square_element = new ShapeElement("square_elem", "square");
infantium.addElement(square_element);

// The star element has its own parameter, which is the 'n_sides'
ShapeElement star_element = new ShapeElement("star_elem", "star", 5);
infantium.addElement(star_element);

// The final circle element
ShapeElement circle_element = new ShapeElement("circle_elem", "circle");
infantium.addElement(circle_element);
```

Picture Elements

Picture elements refer to any real world picture that appears in a game, that is. It is a very straightforward element, here you can see an example of *PictureElements*:

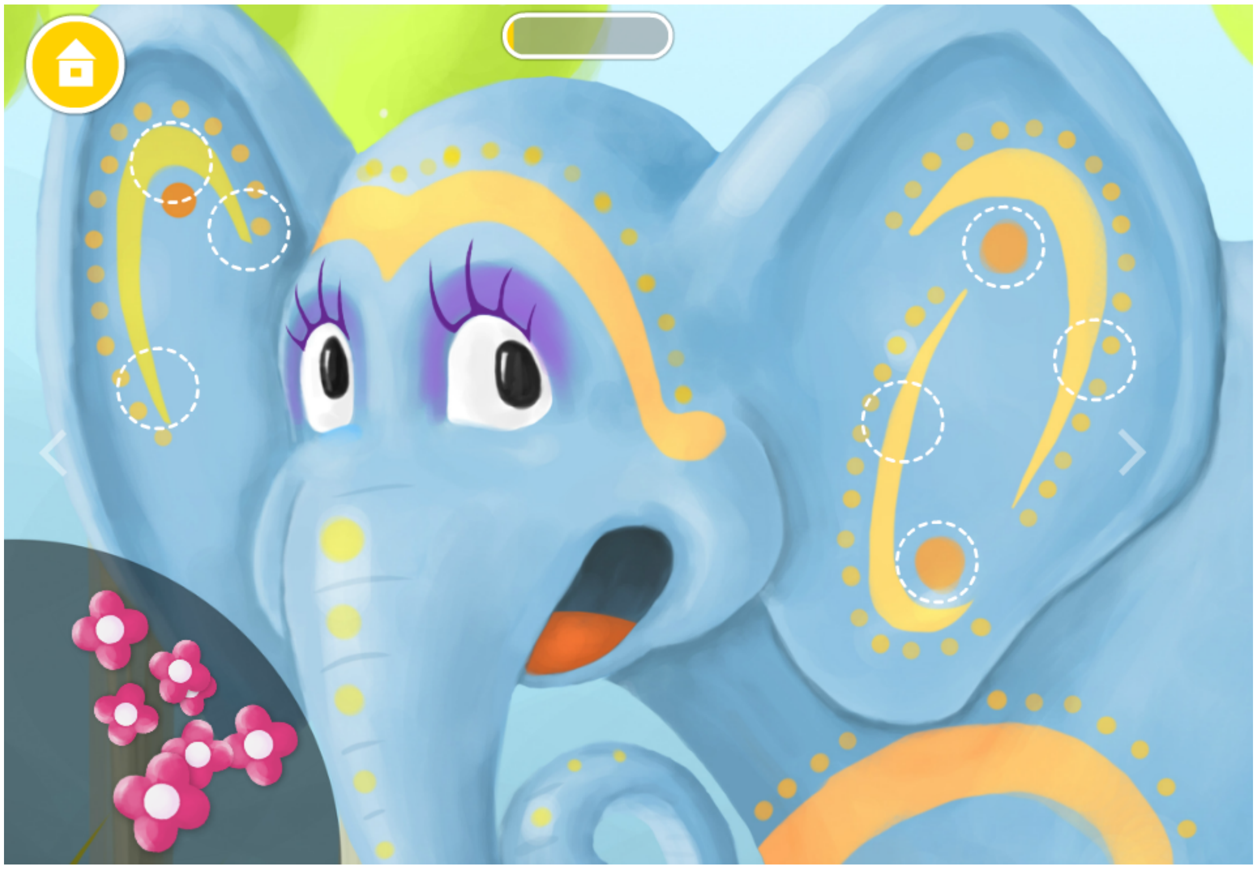


Here is an example of the code:

```
// Adding all the elements in the previous screen from an Array
for (String element : screen_elements) {
    PictureElement simple_picture = new PictureElement(element);
    infantium.addElement(simple_picture);
}

// For Picture elements it is recommended to add the size too,
//  but as most of the attributes, it is optional too
PictureElement improved_picture = new PictureElement("dog", 150, 50);
infantium.addElement(improved_picture);
```

Painted Elements

Painted elements refer to any other kind of element either drawn by hand or by any software, that actually doesn't fall in the previous definition for the *PictureElements*. An example:



In this case, all the flowers would be declared as *PaintedElements*:

```
// Adding all the elements in the previous screen from an Array
for (String element : flower_list) {
    PaintedElement simple_drawing = new PaintedElement(element);
    infantium.addElement(simple_drawing);
}

// For Painted elements it is recommended to add the size too,
//  as well as for PictureElements
PaintedElement improved_drawing = new PaintedElement("cat", 150, 50);
infantium.addElement(improved_drawing);
```

Generic Elements

For any kind of element which you may consider does not fall into any of the previous categories, a generic *Element* object is available too:

```
// Adding a generic Element with some extra information like size
//  and movement
Element random_element = new Element("happiness", 200, 100);
random_element.set_movement("circular");
infantium.addElement(random_element);
```

For more information about the Elements you can head to the Elements Javadocs.

Docs version: 2.3 Last update: April 24, 2014

1.3.2 Goals

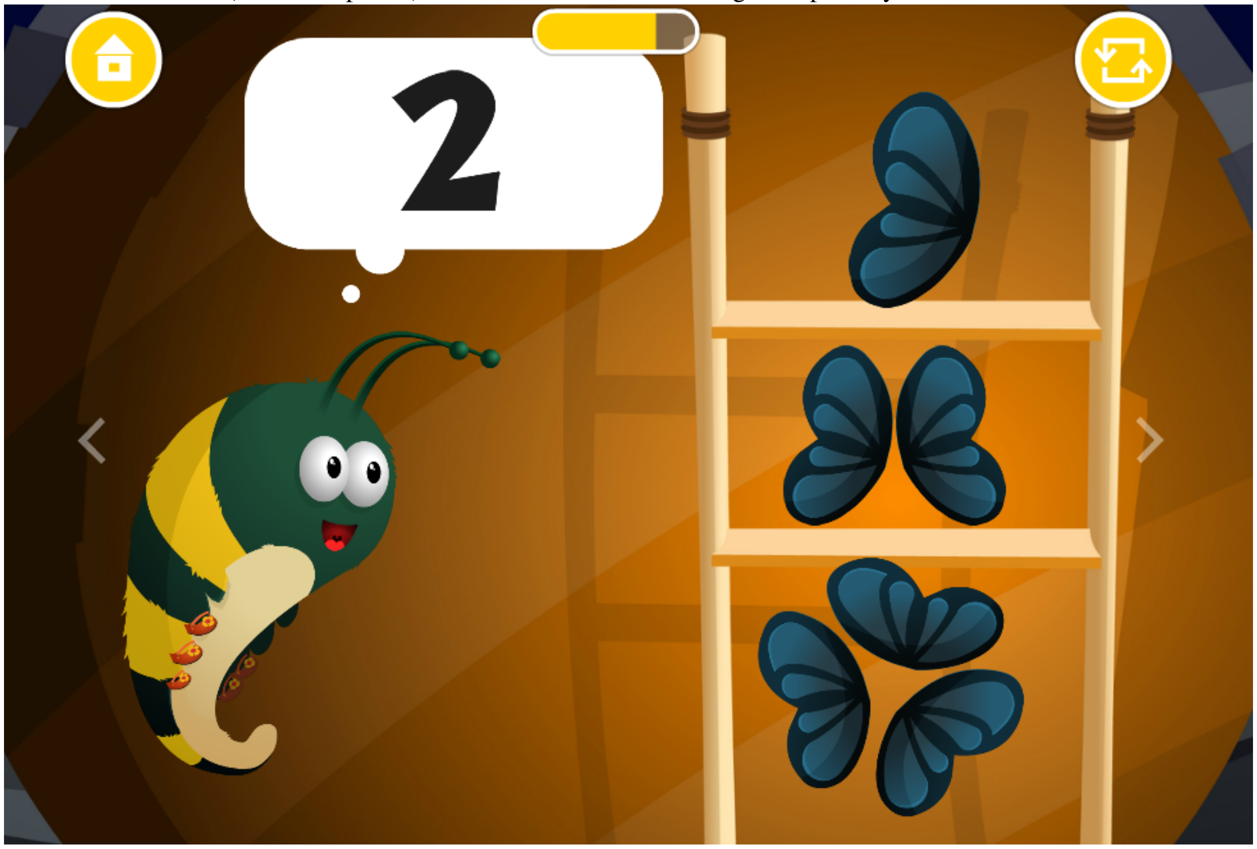
Knowing the *Elements* on the screen is crucial for our analysis, but all of it would lack a real value if we didn't know what are the micro-objectives in the current activity the kid is playing, and of course the final goal. Knowing **what the kid has to do** is as important as knowing **what the kid actually does**. Knowing what the kid has to do is what Goals are for.

Subtypes

There are several classes of *Goals*, depending on the final objective of each activity. To make it clearer in the SDK, developers can choose from different implementations of the Goal class, which are explained below.

Selection Goals

A selection goal object is used whenever the objective requires the kid to select an item from a list of possible elements, where some of them (or all except one) are incorrect. In the following example only one element is correct:



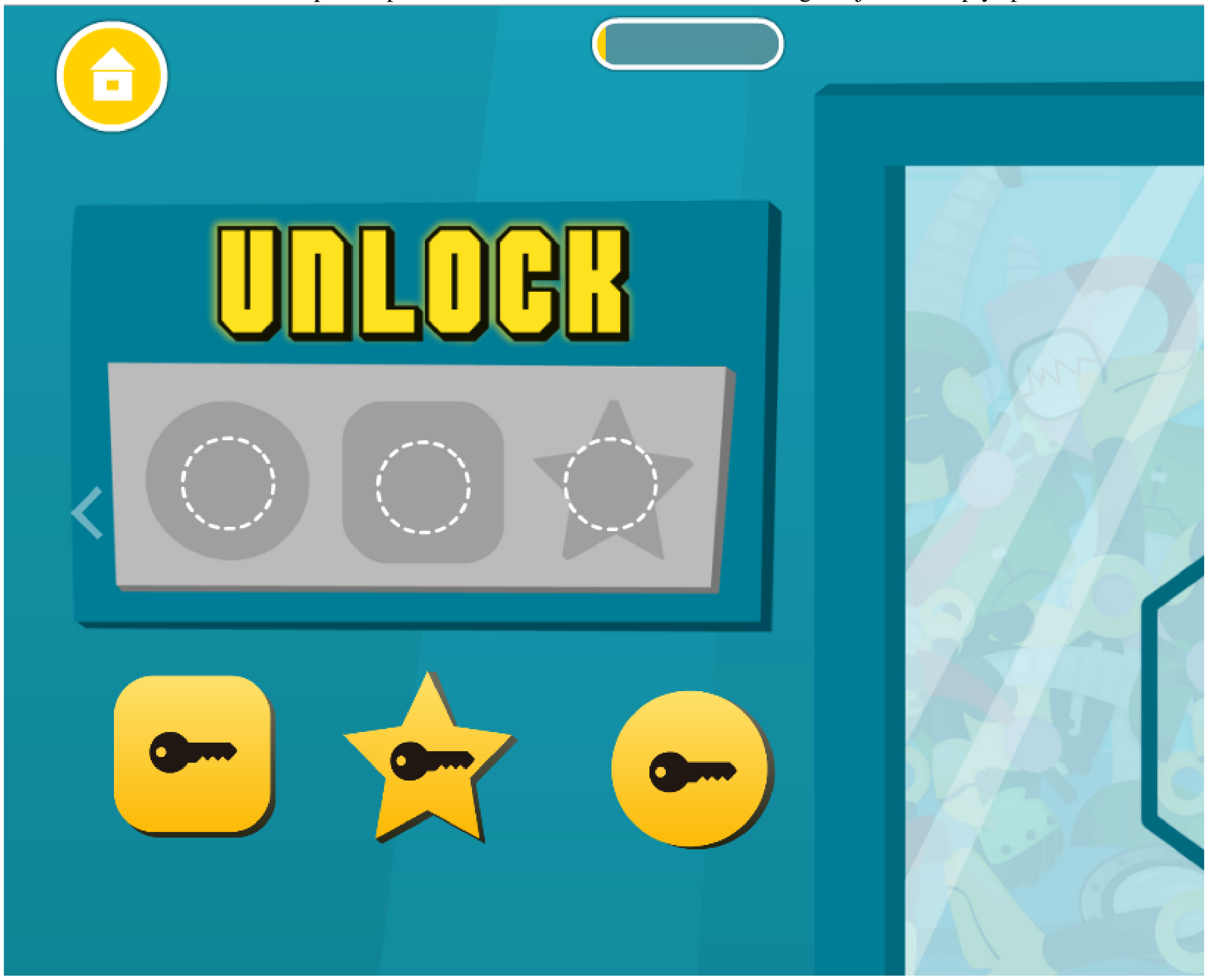
For this activity we would use the *SelectionGoal* class. All of the following are valid examples:

```
// A simple Goal with just a goal name.
//  The 'unique_solution' parameter is true by default
SelectionGoal simple_two_wings_goal = new SelectionGoal("two_wings_goal");
infantium.addGoal(simple_two_wings_goal);

// An improved SelectionGoal with some optional parameters
//  'name', 'unique_solution', and number of 'correct' and 'incorrect' choices.
SelectionGoal improved_two_wings_goal = new SelectionGoal("two_wings_goal", true);
```

```
improved_two_wings_goal.set_n_correct_choices(1);
improved_two_wings_goal.set_n_incorrect_choices(2);
infantium.addGoal(improved_two_wings_goal);

// A full SelectionGoal with all the optional parameters
//  'name', 'unique_solution', 'needed_action', 'correct' and 'incorrect' choices.
SelectionGoal full_two_wings_goal = new SelectionGoal("two_wings_goal", true, "tap");
full_two_wings_goal.set_correct_choices(Arrays.asList("two_wings_element"));
full_two_wings_goal.set_incorrect_choices(Arrays.asList("one_wing_element",
    "three_wings_element"));
infantium.addGoal(full_two_wings_goal);
```

Matching Goals

In other kind of games, the kid is required to match one element with a target, usually corresponding to elements and their shadowed silhouettes, or puzzle pieces into a shadow of the resolved image or just an empty space. For instance:

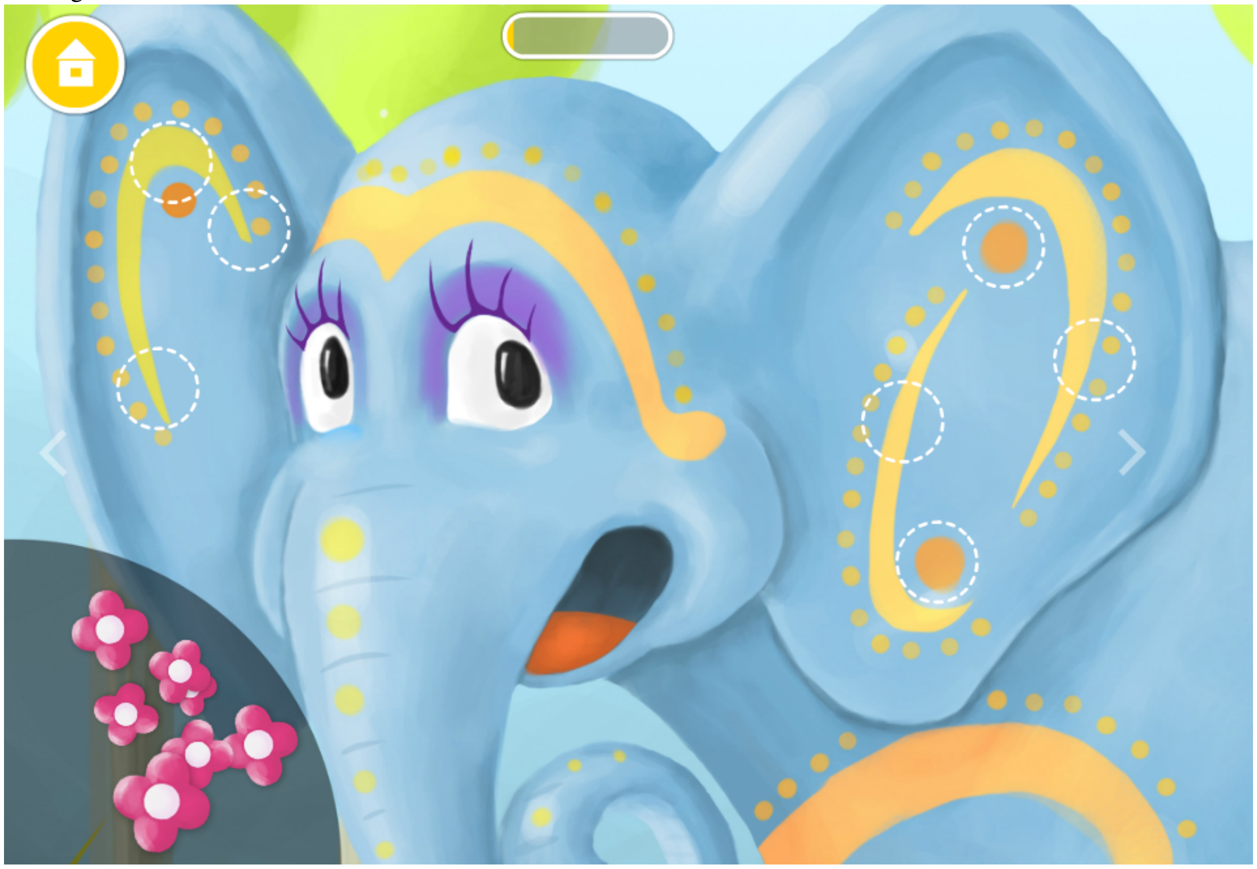


In these cases the *MatchingGoal* is the most appropriate class:

```
// A simple Goal with just a goal name and the 'matching_element' parameter
MatchingGoal simple_select_key_goal = new MatchingGoal("key_goal_1", "squared_element");
infantium.addGoal(simple_select_key_goal);
```

```
// A full MatchingGoal with the optional parameters 'target_element' and 'correspondence_type'
MatchingGoal full_select_key_goal = new MatchingGoal("key_goal_1", "squared_element",
    "squared_shadow_element");
full_select_key_goal.add_correspondence_type("shape");
full_select_key_goal.add_correspondence_type("size");
infantium.addGoal(full_select_key_goal);
```

However, there are some game activities where the kid can match different elements among them, where for a given target multiple elements fit as the correct solution, for instance the following activity, where all the flowers fit each of the targets:

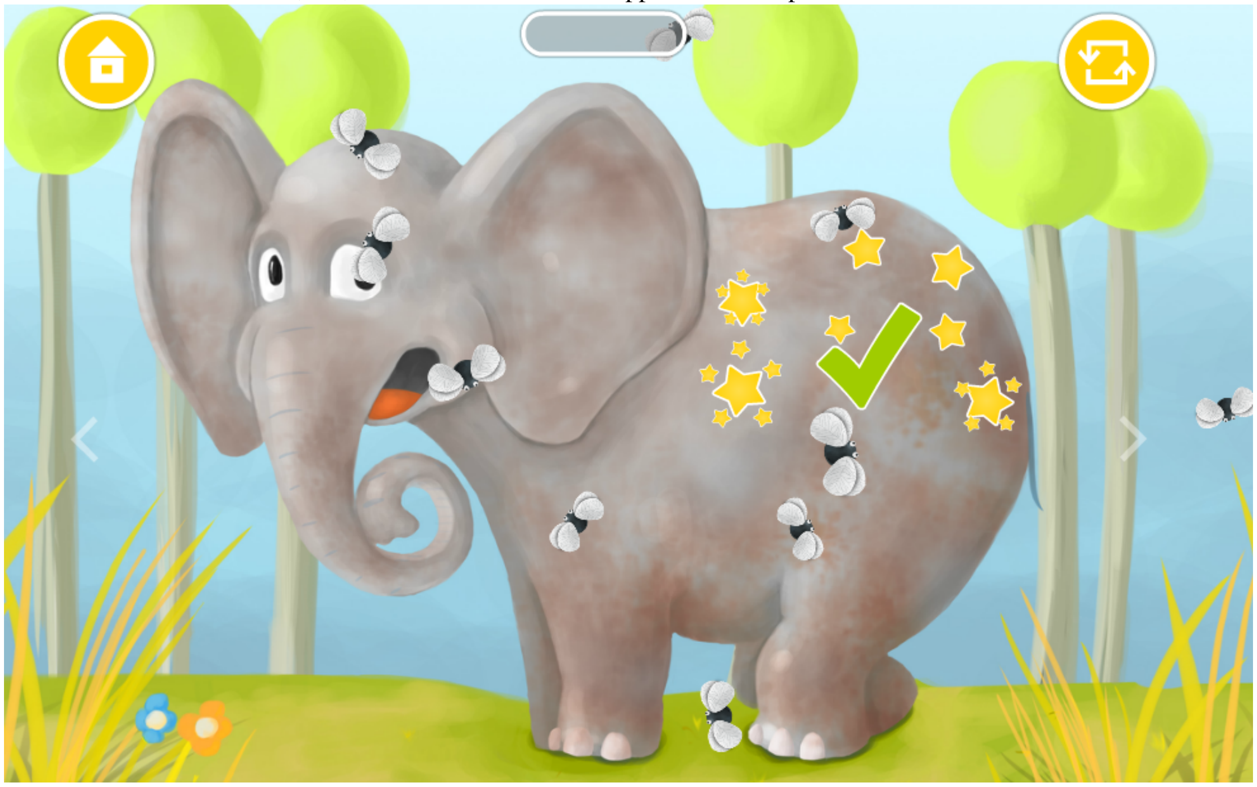


In this case you can add a list of element IDs that fit the same target, or even use the keyword "*ALL*" to mark that all the elements can fit that target. An example of the code:

```
// The same previous Goal with just a goal name and a list of 'matching_element' ids
//  as a parameter
MatchingGoal multiple_select_key_goal = new MatchingGoal("key_goal_1",
    Arrays.asList("flower_1", "flower_3", "flower_5"));
infantium.addGoal(multiple_select_key_goal);

// A full MatchingGoal with the optional parameters 'target_element' and 'correspondence_type'
//  which can accept ALL elements as correct elements
MatchingGoal full_multiple_select_key_goal = new MatchingGoal("key_goal_1", "ALL",
    "flower_spot_element_1");
full_multiple_select_key_goal.add_correspondence_type("shape");
full_multiple_select_key_goal.add_correspondence_type("size");
infantium.addGoal(full_multiple_select_key_goal);
```

Tapping Goals

A tapping goal is a kind of goal where the kid just has to tap on the elements he's seeing, without making any distinction of the nature of the element, i.e., all elements need to be tapped. An example could be:



In this case, there is an object named *TappingGoal* which adapts to this scenario:

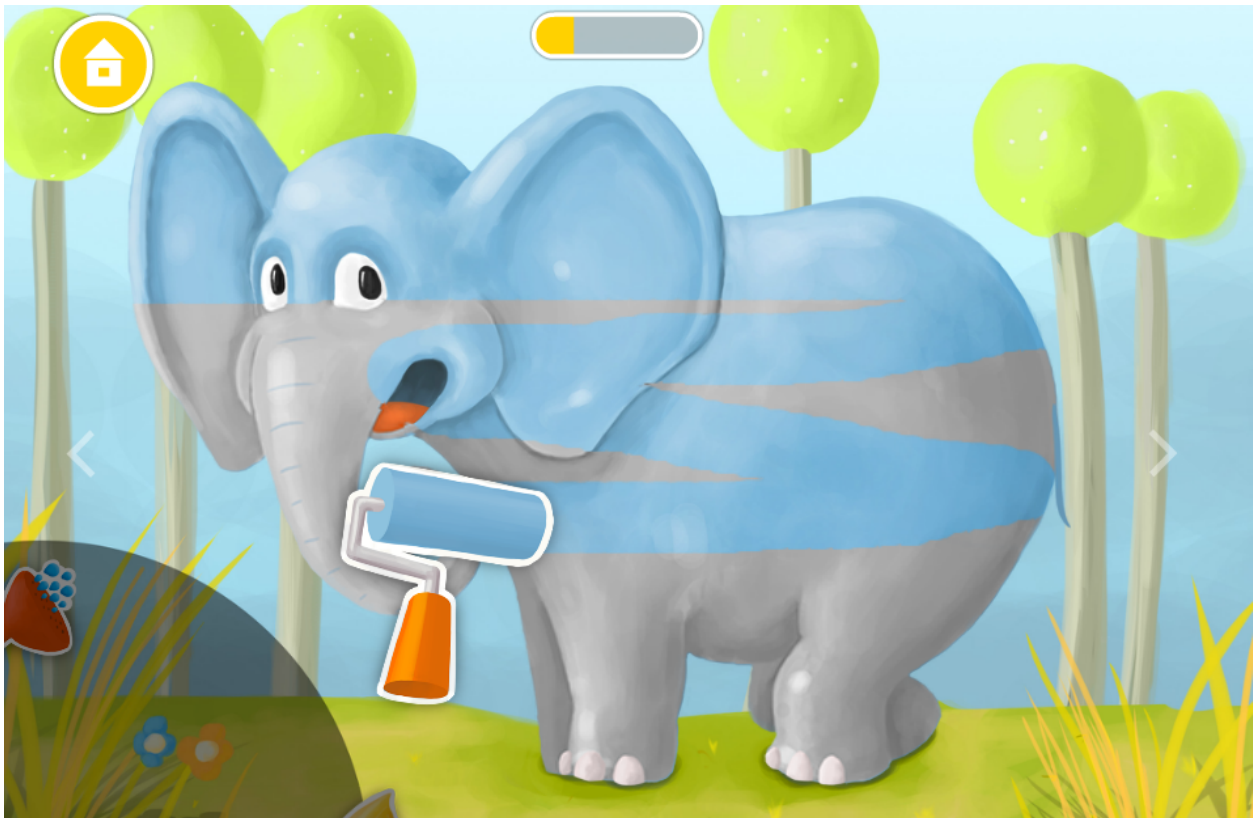
```
// A TappingGoal with the least parameters
TappingGoal simple_tapping_goal = new TappingGoal("tap_odd_flies",
    Arrays.asList("fly_1", "fly_3", "fly_5"));
infantium.addGoal(simple_tapping_goal);

// The "ALL" parameter can be used here too
TappingGoal simple_tapping_goal = new TappingGoal("tap_all_flies", "ALL");
infantium.addGoal(simple_tapping_goal);
```

If it is possible to give more information about the nature of the elements on the screen, it can be done by specifying the type of those elements with an additional *type* parameter, which can take the values: "*moving*" for dynamic elements, "*highlighted*" for elements which stand out intentionally from the rest of the elements, or "*hidden*" for partially hidden elements.

```
// A TappingGoal with the least parameters
TappingGoal moving_tapping_goal = new TappingGoal("tap_all_flies", "ALL", "moving");
infantium.addGoal(moving_tapping_goal);
```

Generic Goals

There are some kind of goals which do not fall inside any of the previous sutbypes. For those kind of objectives the generic Goal should be enough to describe the nature of the activity. For instance the following painting game:



This activity could be defined with the default *Goal* class:

```
// A Goal with the least parameters
Goal painting_goal = new Goal("paint_the_elephant");
infantium.addGoal(painting_goal);

// A full Goal with all the optional parameters:
//  'time_limit' (in milliseconds),
//  'instructions' referencing to an element id of the help text,
//  'auto_eval' which describes if the action automatically triggers
//    the evaluation process or not
Goal full_painting_goal = new Goal("paint_the_elephant", 10000, true);
full_painting_goal.set_instructions("instructions_element");
infantium.addGoal(full_painting_goal);
```

For more information about the Goals you can head to the Goal Javadocs.

Docs version: 2.3 Last update: April 24, 2014

1.3.3 Basic Interactions

As you may guess, the set of interactions of the kid with the screen is one of the cornerstones of our analysis. Being as important as it is for us, we want to make that interaction as simple as possible while keeping it meaningful for us. That is why we have tried to make it a simple interaction of the developers with the SDK. This kind of interactions is managed by the concept named *Basic Interaction*.

Basic Interactions capture what the kid is doing with the screen, regarding both the action itself, and the meaning of the action for the current activity at the app. Below you can find a simple example to start with:

```
// Dragging the ball to the dog is the goal of the activity,
//  and thus it is represented a "success".
InfantiumResponse res = infantium.newBasicInteraction("success", "ball", "drag_the_ball");

// Another example, if the kid drags the "smartphone" element,
//  but was not the goal of this activity.
infantium.newBasicInteraction("error", "smartphone", "drag_the_ball");
```

Docs version: 2.3 Last update: April 24, 2014

1.3.4 Events

In the version 2.3 of the SDK we have introduced the *Events* figure, which will allow developers to register the different types of events that may happen during the playing time of the kid with their content. The first batch of events include two different types, apart from the *generic events*, and those are *Sound Events* and *Missed Opportunities*.

Declaring and Triggering Events

The *Events* section of the SDK differentiates two steps in the events area: declaring an event once, and triggering it as many times as it happens in the game. Thus, there are two main parts in this section, Declaring Events and Triggering Events.

Declaring Events

The first step of process is to declare the *Events* that may happen during the current scene. They are declared at the same time of the *Elements* and *Goals*. There is a *Generic Event* which can be used to declare *Events* that do not fall in the specific types. The specific types are explained afterwards.

Generic Events

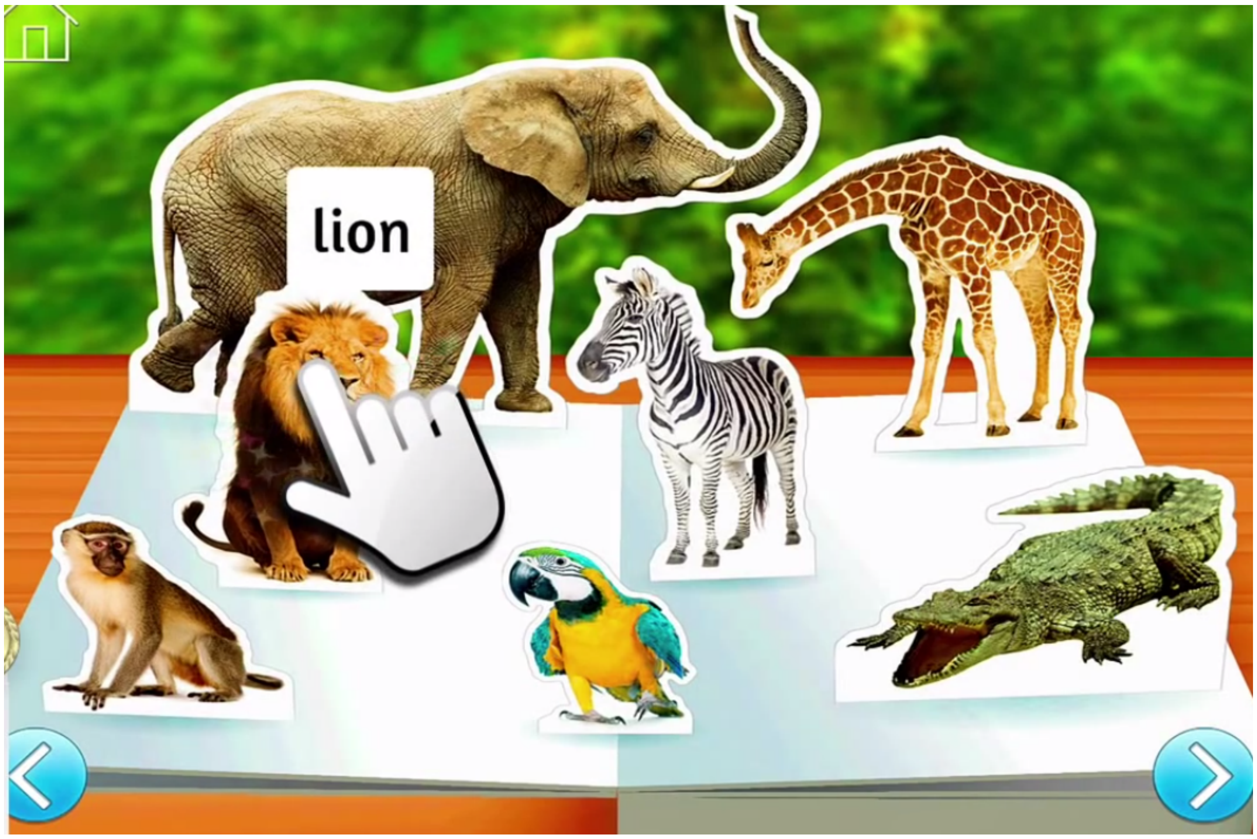
Generic events do not give much information, as they do not specify which kind of event the kid is seeing. Nevertheless, they are useful for us as they allow our analytic system to know if there is any kind of reward the kid is having after performing the different actions on the screen, and thus that is important data to take into account when analyzing the different interactions declared during play time.

Here is the example code:

```
// Add a simple generic event with its ID
Event ev = new Event("sun_starts_shining");
InfantiumResponse res = infantium.addEvent(ev);
```

Sound Events

When an event represents a sound being played at a specific time or after a specific action, this should be registered with the *SoundEvent* class. This event is very useful for us, as it may include the text read, the language of the text, the sound or song which may be sung, etc. We can know how many times a kid has been exposed to a previous concept before the current game, so we greatly encourage developers to register this kind of events!



In the previous example, every time the kid taps on one of the *Elements* on the screen, he can hear the name of the animal he is touching. In this example, one *SoundEvent* would be declared per *Element* on the screen, and it would be *Triggered* every time the kid taps on the picture (explained in the next subsection).

```
// Add a simple Sound Event
SoundEvent ev1 = new SoundEvent("lion_name_sound_basic");
InfantiumResponse res1 = infantium.addEvent(ev1);

// It is also recommended to add the text and type of Sound:
SoundEvent ev2 = new SoundEvent("lion_name_sound_w_text", "voice");
ev2.set_associated_text("Lion", "en-US");
InfantiumResponse res2 = infantium.addEvent(ev2);

// We allow also to reuse other text elements used before
// (for example instructions in the games)
TextElement help_text_element = new TextElement("help_text", "en-US",
        "MAKE A SHADOW PUPPET");
infantium.addElement(help_text_element);
SoundEvent ev3 = new SoundEvent("lion_name_sound_with_element", "voice",
        "help_text_element");
InfantiumResponse res3 = infantium.addEvent(ev3);

// Add a full Sound Event with the optional parameters
//  imprecise_sound_volume: the approximate sound level
//  duration: the milliseconds of duration of the sound
SoundEvent ev4 = new SoundEvent("full_lion_name_sound_w_text", "voice");
ev4.set_associated_text("Lion", "en-US");
ev4.set_imprecise_sound_volume(0.5);
ev4.set_duration(5000L);
InfantiumResponse res4 = infantium.addEvent(ev4);
```

Triggering Events

The second part of this section is triggering the events as soon as they occur in the current scene. This is very simple, it just requires developers to make a call to the triggerExistingEvent(String event_id) method in the event handlers of your code. Here you can find an example for triggering the events declared in the previous section:

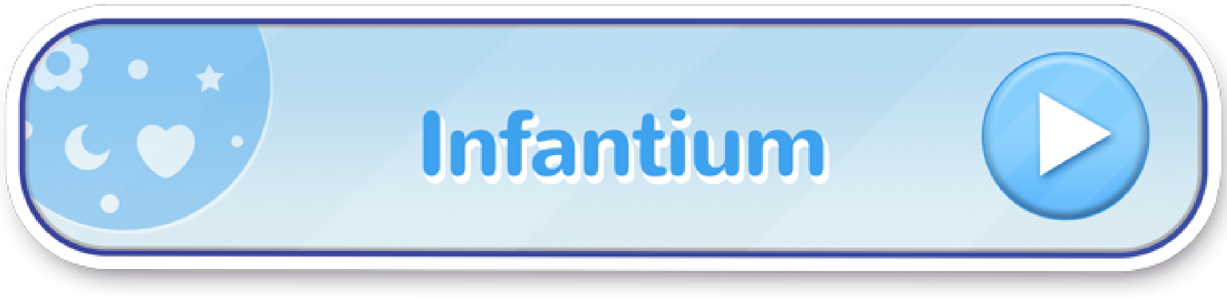
```
// Trigger the first Generic Event
InfantiumResponse res = infantium.triggerExistingEvent("sun_starts_shining");

// Trigger one of the previous Lion Sound Events.
//  triggered_by parameter indicates what triggered the event
InfantiumResponse res = infantium.triggerExistingEvent("lion_name_sound_w_text", "tap");
```

For more information about the Events you can head to the Events Javadocs. Docs version: 2.3 Last update: April 24, 2014

1.3.5 Adding the Infantium Button

Downloading

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



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.

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.

Docs version: 2.3 Last update: April 24, 2014

1.3.6 Overriding the Android Cycle

The SDK needs to execute some logic at the moment of creation and destruction of the contents adapted to Infantium, as well as at the moment of resuming and pausing those contents. To fully understand this, in case the native methods *onResume* and *onPause* are not familiar to you, you should check out the Android Activity Lifecycle page.

Once this is clear, those methods can be easily overwritten from your App by setting your own *onResume* and *onPause* which call their parent methods from your Activity. Inside those methods then it is where you should call the Infantium SDK methods that will take care of leaving your app in a consistent method. This is because many apps in the same tablet will be using the SDK, and it is important there are no receivers active from one app that could interfere with the others. All you have to do is overwrite the native methods and call the SDK functions. Here is an example of the code that would achieve this goal:

```java
@Override
protected void onResume() {
        super.onResume();
        Log.i("Your-App-Tag", "--- Resumed MainActivity ---");  // Just for example purposes
        infantium.onResumeInfantium();
}

@Override
protected void onPause() {
        super.onPause();
        Log.i("Your-App-Tag", "--- Paused MainActivity ---");  // Just for example purposes
        infantium.onPauseInfantium();
}
```

With these simple two functions of two lines each (you can avoid the comments), the app will seamlessly integrate with other apps in the same tablet with Infantium, both at creation/destruction and at resuming/pausing!

Docs version: 2.3 Last update: April 24, 2014

Docs version: 2.3 Last update: April 24, 2014

1.4 Class Reference

Infantium Android SDK, current version 2.3, 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.

1.4.1 Element

Docs version: 2.3 Last update: April 24, 2014

Docs version: 2.3 Last update: April 24, 2014

1.5 Appendix: Plugins

We are conscious that most of our partners do not program directly in native Android, and thus we have created a set of plugins that allow direct communication from different frameworks with the Native SDK. Here is the list of Frameworks currently supported. If you do not find your framework below don't get discouraged! Contact us and we will be more than happy to give you assistance to create a new plugin in any framework you might use.

1.5.1 Adobe ANE

You can find the Plugin's code in our Github repository.

Docs version: 2.3 Last update: April 24, 2014

1.5.2 Cocos2d-X

You can find the Plugin's code in our Github repository.

Docs version: 2.3 Last update: April 24, 2014

1.5.3 Corona SDK

You can find the Plugin's code in our Github repository. We also have available a sample project for Corona Enterprise. Specially, it is recommended to check the main.lua file where there are examples of all the calls available in the plugin. main.lua file link

Docs version: 2.3 Last update: April 24, 2014

1.5.4 PhoneGap

You can find the Plugin's code in our Github repository.

Docs version: 2.3 Last update: April 24, 2014

1.5.5 Unity

You can find the Plugin's code in our Github repository.

Docs version: 2.3 Last update: April 24, 2014

Docs version: 2.3 Last update: April 24, 2014

1.6 Javadocs

Here you can check the Javadocs reference of the SDK.

Docs version: 2.3 Last update: April 24, 2014

Docs version: 2.3 Last update: April 24, 2014