# Inter-App Communication between Android Apps Developed in App-Inventor and Android Studio

Lance A. Allison
Dept. of Computer Science
Winston-Salem State University
Winston-Salem, NC 27110
Phone: 1-336-7502480
lallison114@rams.wssu.edu

Mohammad Muztaba Fuad
Dept. of Computer Science
Winston-Salem State University
Winston-Salem, NC 27110
Phone: 1-336-7503325
fuadmo@wssu.edu

## ABSTRACT
Communications between mobile apps are an important aspect of mobile platforms. Android is specifically designed with inter-app communication in mind and depends on this to provide different platform specific functionalities. Android Apps can either be designed with the help of Android SDK and using IDEs such as Android Studio or by using a browser based platform called App Inventor. These two development platforms provide their own technique for inter-app communication in the same platform, however lack an established method of inter-app communication when apps are developed using the two seperate development platforms. This paper provides the missing information required for the app communications and presents the method for sending and receiving arguments between apps developed in these two platforms. The paper also outlines the significance of the result, and examines their limitations.

## CCS Concepts
**• Ubiquitous and mobile computing → Ubiquitous and mobile computing design and evaluation methods • Software creation and management→ Software development techniques**

## Keywords
Android, App Inventor, Android Studio, Mobile apps.

## 1. INTRODUCTION
Applications (apps) in Android can be developed using either MIT's App Inventor 2 (AI) [3] or by using IDEs such as Android Studio (AS) [1] with the help of Android SDK. MIT's AI is the second version of the Google's original App Inventor, which is a web browser based development environment for a simpler way to develop android apps. With little to no knowledge of programming, one can develop and deploy an android app using AI. However, this simplicity comes at a price; AI does not provide all of Android's advanced features and most apps developed in AI have to follow a specific design template. To access all of Android's features, one needs to develop apps using tools such as Android Studio, which is provided by Google as a full-fledged development environment for Android development, debugging, testing, and packaging. While communications between any two Android apps developed in either AI or AS are fairly straightforward; apps developed by these two development platform separately do not have a similar way to communicate with each other. To define the problem, suppose we have two AI apps $AI_1$ and $AI_2$, along with two AS apps $AS_1$, $AS_2$. Now also consider, $P_1$ is the process through which $AI_1$ communicates with $AI_2$ and $P_2$ is the process through which $AS_1$ communicates with $AS_2$. Then it is known that $P_1 \neq P_2$. This results in no established process, $P_x$, that will allow apps developed by AS to communicate with apps developed by AI. This paper will present this process $P_x$, which will allow apps developed in two different platforms to communicate with each other programmatically. Obvious questions arise, as to why we need to generate this process given that AI apps do not have advanced features. Since AI provides faster lead time in developing apps than AS, having a backend system developed by AS on which AI apps can run, would give developers a faster way of developing new app ecosystems. One such situation is presented below.

Since 2013, Winston-Salem State University has begun implementing a Mobile Response System (MRS) [4]-[6]. MRS is a mobile learning environment that enhances class room engagement by creating a responsive environment where students solve problems in an interactive way that communicates solutions directly and immediately to their instructor. This is performed through Android powered mobile devices, which have interactive activities in which students learn principles of their subject with immediate feedback. This mobile platform is used by the faculty to develop their own interactive activities to be used in the classroom. However, currently, such interactive activities must be developed by AS to be used with MRS. One way to alleviate this problem is to allow the MRS to communicate with apps developed by both AI and AS. That way, interactive apps developed by faculty from other disciplines using AI can utilize MRS to deploy their activities in the class.

## 2. DEVISED METHOD
Sending a String from an AS app to an AI app requires an intent, with the preset extra key of "*APP_INVENTOR_START*" which the AI application will recognize as the extra value. The challenge is to find the qualified name of the AI app that we like to initiate. Normally for AS apps, the name of the Java package concatenated with the class name of the app activity is the qualified name to locate an app in the device. However, there were no such identifiable naming convention for AI apps, as no one can see the source code. With the help of the Android Debug Bridge [2], we identified that AI apps has the following qualified naming convention for their apps:

*appinventor.ai_Google-User-Name.AI-App-Screen_name*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*Mobilesoft '16*, May 16–17, 2016, Austin, TX, USA.
ACM 978-1-4503-4178-3/16/05.
DOI: http://dx.doi.org/10.1145/2897073.2897117

With the help of the above information, the following block of code shows the AS intent for launching an AI app and sending a string.

```
Intent launch = getPackageManager().getLaunchIntentForPackage
                ("appinventor.ai_GoogleUSERname.AIbacktoAS");
launch.putExtra ("APP_INVENTOR_START",ARG_TO_PASS);
startActivity(launch);
```

When the AI application initializes the screen, the app will set the on screen text label (Label1) to the passed string value followed by the activity starter blocks required to return the string back to the original AS application. This is shown in Figure 1.

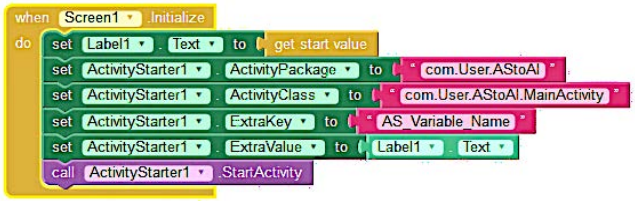


**Figure 1. AI blocks to communicate with AS app.**

After the AI application launches its intent to return the string, the AS application will receive the original string as shown below:

```
String variable = getIntent().getStringExtra ("AS_Variable_Name");
```

For the reverse operation to be performed, the same string was sent from an AI application to an AS app and then sent back to the original AI app. This required much of the same blocks that are depicted in figure 1. The random string was set to the label and included in the activity starter's extra value. Figure 2 shows the AI intent blocks used to send the string.

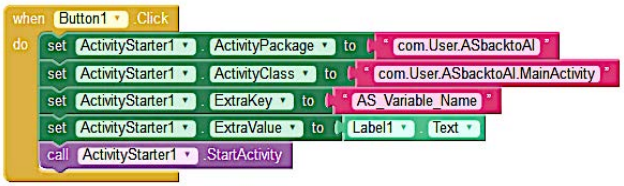


**Figure 2. AI application intent for AS application.**

The AS application then receives the string, followed by the launch intent for the original AI app. Again, the extra key must be set to "*APP_INVENTOR_START*" being that AI applications will only initialize values with this extra key. The block of code below demonstrates the code used.

```
String variable = getIntent().getStringExtra ("AS_Variable_Name");
Intent launch = getPackageManager().getLaunchIntentForPackage
                ("appinventor.ai_GoogleUSERname.AItoAS");
launch.putExtra("APP_INVENTOR_START",variable);
startActivity(launch);
```

Finally, the original AI application would receive and initialize the random string using the blocks as shown in Figure 3.

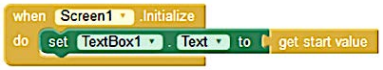


**Figure 3. AI blocks required to receive starting arguments.**

## 3. RESULTS

To investigate the cross platform communications, there were two applications developed per platform. A randomly generated 100-character string made up of both upper and lower case letters was sent back and forth between two applications to measure the length of time taken. This allowed us to check how to initiate apps, how to pass values between them, how to handle Android lifecycle events, and to determine whether there was any difference in performance during execution.

Four sets of trials (AI to AI, AI to AS, AS to AS and AS to AI) were performed measuring the time taken, with the AI to AI and AS to AS trails used as the control variables. Each set of trials had 15 runs and then the times were averaged and shown in Table 1.

**Table 1. Elapsed time during the string's transfer trails.**

| Transfer paths | Avg. Time (ms) | Stand. Dev. |
|---|---|---|
| $AI_1$ to $AI_2$ back to $AI_1$ | 189.8 | 7.3 |
| $AI_1$ to $AS_2$ back to $AI_1$ | 159.4 | 7.1 |
| $AS_1$ to $AS_2$ back to $AS_1$ | 154.6 | 6.8 |
| $AS_1$ to $AI_2$ back to $AS_1$ | 269.4 | 10.4 |

In situations where AI apps were at the receiving end, transfer times were significantly longer. Alternatively, when AS applications were on the receiving end transfer times were much lower. AI communications may require more time to process because of its use of the Kawa compiler. When AI apps are compiled their code blocks are converted to byte code by Kawa compiler and then interpreted to be executable. AS apps do not require this third party program to convert the code, resulting in a more uniform application with faster response times.

The application size has also been recorded for comparison and AI apps were larger than AS apps even though the lines/blocks of code required were similar. There are also some limitations to the methods outlined above. Firstly, when receiving strings with AI apps, there is only the option to initialize one string value. This means that you cannot initialize multiple variables; AI apps will only initialize the first received value. Secondly, AI apps can only receive a string to its main activity. There is no ability to use an intent filter to initiate a specific activity, which limits AI app's communications greatly compared to AS apps.

## 4. CONCLUSION AND FUTURE WORK

This paper presents a technique to provide Android inter-app communications when apps are developed in two different development platforms: Android Studio and App Inventor. Performance data was provided and methods were compared to find any limitations. In the future, the presented technique will be used to extend MRS to other disciplines and will shed more light into how larger and more complex apps will work together when they are designed by two different Android development platforms.

## 5. ACKNOWLEDGMENTS

We would like to thank Winston-Salem State University's undergraduate research grant and NSF grant #1332531 for supporting this work.

## 6. REFERENCES

[1] Android Studio, http://developer.android.com/tools/studio/index.html.

[2] Android Debug Bridge, 2015, http://developer.android.com/tools/help/adb.html.

[3] App Inventor, http://appinventor.mit.edu.

[4] Fuad M. M. and Deb. D., 2014, Design and Development of a Mobile Classroom Response System for Interactive Problem Solving, In *Proceedings of 26th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Vancouver, Canada, July 1-3, 49-52.

[5] Fuad, M.M., Deb, D. and Etim, J., 2014, An Evidence Based Learning and Teaching Strategy for Computer Science Classrooms and Its Extension into a Mobile Classroom Response System. In *Proceedings of 14th IEEE Advanced Learning Technologies Conference (ICALT)*, July 7-10, Athens, Greece, 149-153. DOI:10.1109/ICALT.2014.52

[6] Mobile Response System, http://compsci.wssu.edu/MRS.