Kotlin Mobile Application for PC Control

Jacob T. Schwartz
Department of Computer Science
University of Dayton
Dayton, Ohio USA
schwartzj1@udayton.edu

ABSTRACT

We present a PC remote Bluetooth control application implemented in the Kotlin programming language for Android. Using the Android Bluetooth library, the mobile application communicates with a local server running on a client PC to interact with it's operating system and perform actions such as volume control, custom keystrokes, turning off or rebooting the device, and more. The client server runs on Kotlin, utilizing the languages ability to interoperate with the Java programming language to utilize Javax's Bluecove library to communicate with the client machine's Bluetooth adapter. This is the only option to write a Bluetooth server on any desktop operating system, as the only native Kotlin library for Bluetooth communication, at this time, works exclusively on the Android operating system; Android, nor its fork to create Internet of Things devices (Android of Things) are capable of running alongside major operating systems.

KEYWORDS

Android, Bluetooth, Java interoperability.

ACM Reference Format:
Jacob T. Schwartz. 2019. Kotlin Mobile Application for PC Control. In *Proceedings of CPS 452-01: Emerging Languages.* ACM, New York, NY, USA, 3 pages.

1 INTRODUCTION

Kotlin is a powerful general-purpose language built to be a successor to the Java programming language, and its most powerful ability is as a language to write mobile applications for Android. Because it was written with the intention of taking over for Java, it is able to interoperate with one hundred percent of code writtn in Java. This ability allows us to use all the power and ability of Java libraries while writing in a cleaner more concise language. Not only this, but Kotlin also implements features to resolve many of Java's largest problems, with the biggest being NullPointerExceptions which are resolved through Kotlin's Null Safety features. Altogether, this put Kotlin in a prime position to be used for Android app development.

2 BLUETOOTH SERVER

As the intention of this presentation is an application that allows for the control of a Windows PC through an Android application over Bluetooth, there are two separate applications built to make this possible: a Android application, and a Windows Bluetooth server. To best explain the workings of the project, we will begin with the Windows server and how it handles input.

2.1 Server Startup and Connection

Because Kotlin lacks a native library to utilize Bluetooth on Windows, we must take advantage of Kotlin's interoperability with the Java programming language and utilize the Bluecove library. With this in place we may start up a Bluetooth server, making sure to use a random UUID and looking for established Bluetooth UUID that is hard-coded into the Android application. Once this server is open and ready to accept connections from the application on connected devices.

2.2 Input Handling

Once the Android application is connected to the Bluetooth server, it is ready to receive messages and handle them accordingly. The type of commands that can be sent from the application are broken up into four categories: Access, Keystrokes, Power, and Volume. Each category is wrapped up in it's own Singleton to aid in modularization and help keep the code easier to read. Each Singleton also has a corresponding coroutine channel, this is so that we may launch the Singletons in their own coroutine in order to keep the main thread able to accept new input immediately after, rather than being held up by the operations executed as a result of the action.

2.2.1 Access Commands. The Access Commands are the most limited, though easiest to implement. With the lock and logout features in Windows being offered as commands for the Windows command prompt, all that must be done is run the corresponding commands using a Kotlin ProcessBuilder, which is also wrapped in another Singleton so that other command groupings may utilize it as well.

2.2.2 Keystroke Commands. Keystrokes are run on the server far differently than any other command category. With no command line interface available to handle this sort of action, we must again resort to interoperating with Java and putting to use the Robot library. This library offers the unique ability to simulate key presses and releases. Given the difficulty of ordering the press and release function calls to ensure the proper keystroke is executed, the initial prototype has the keystrokes hard-coded into the application as described in Table 1. Though in the future, this option could be offered through the use of file storage and a stack-based operation to handle the ordering.

CPS 452-01: Emerging Languages, Fall 2019, University of Dayton, Ohio 45469–0232 USA

Table 1: Hard-Coded Keystroke Operations

Keystroke Choice	Keystroke Performed
Alpha	*<alt + c>*
Beta	*<space>*
Gamma	*<ctrl + alt + n>*
Delta	*<ctrl + shift + c>*

2.2.3 Power Commands. All Power commands are run the same way as the Access grouping, simply putting to use the native command line operations for the given features. It is worth noting that although the Android application UI prototype only offers a Shutdown option, the Windows server component does have the built-in functionality for Restart and Hibernate as well.

2.2.4 Volume Commands. Although the Volume commands are executed via commands in a ProcessBuilder, similarly to Access and Power, the commands this Singleton runs are far different than the aforementioned groups. Given that Windows does not offer native command line operations for actions on the device's volume, and the fact that Kotlin running on the JVM limits it's ability to handle such low level device operations, we were forced to apply a unique solution. This was solved with the SetVol tool. This tool is an executable that is written in C++ (giving it the ability to affect Windows volume), does not need to be installed to run, and is licensed under CC BY-ND 4.0 which allows for free and unbridled use and redistribution.

3 ANDROID APPLICATION

With the explanation of how the server handles the messages it receives, we may now move on to covering how these messages are sent from the remote device. The Android application is made up of only two separate screens or "Activities" as they are known in Android. The main screen features the expected user interface with a button for each action which can be performed, as well as a Settings icon to allow you to choose the device which the user would like to control.

3.1 Settings Activity

Upon selecting the Settings, the application opens a new Intent, essentially it is a breadcrumb for the application to build up a stack of its destinations so that it may find its way back to previous Activities. As we will need to share information between the Main and Settings activity, we must also provide the Intent with a unique result code. Results are stored in a database local to the app, with the key being the result code provided. This will allow the application to access the information in any Activity as long as it has the corresponding result code.

When the Settings Activity opens, it will initially check to ensure that the device has Bluetooth enabled, if it is not the application makes a call to the Android API to prompt the user to enable this feature. With Bluetooth enabled, the application pulls a list of all stored paired Bluetooth devices the Android device has access to, displayed as their "friendly" name as this is easier for the user and requires no additional information be requested as Android handles this natively [3]. Once one is selected, the Bluetooth address is stored with the result code.

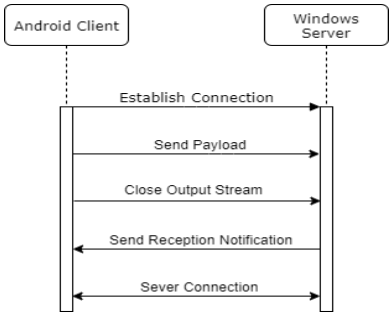

Figure 1: A visual representation of how the client and server communicate over Bluetooth.

3.2 Main Activity

With a Bluetooth device selected, once the user returns to the Main Activity, the application will attempt to establish a connection to the device. [2] If a connection cannot be established to the selected device, the user will be informed by an error prompt and they will be allowed to choose a new connection. Provided the application is able to make a connection to the selected device, the screen will show the connected device's name and the user can now proceed to use the application for its intended use.

Each icon (aside from the Settings) has it's own specific payload to be sent to the server, outlining it's intended command category as well as it's specific action to be performed.

Sending the message from the application to the server is among the most confusing to get working properly, as Android does not handle a Bluetooth socket output stream the same as a normal output stream, or at least the API for it is quite buggy if the mannerisms are not intentional. To start, calling flush on the output stream will not do as it claims and seems to be do nothing when called, and as a result, each payload must be terminated with a newline to force the buffer to be flushed, which means the payload must then be trimmed on the server upon reception.

4 CONCLUSION

This project was a an onslaught of bugs, not only from the project itself, but from within Android and Bluetooth as well. Many idiosyncrasies had to be overcome in order to bring this project to a working state. In the end, it became clear just how different Bluetooth is from other forms of wireless communication, specifically Wi-Fi. When developing for Wi-Fi on an endpoint device, we take for granted how much the Internet infrastructure handles for us, but with Bluetooth being near-field and no devices, such as routers, in between the client and server, all of the work lands on the developer to ensure the two endpoints can not only establish a connection, but send and receive data as well.

In the end, the ultimate power of the Kotlin programming language is to thank for bringing this project together. Without some of it's ingrained, unique features, most notably but not limited to Java interoperability, this project could not have come to fruition. It is no surprise this language has been named the official language of Android in such a short time, it is easy to see how much cleaner the code is, how much easier it is to access data, and how many fewer lines are required to complete the same task [1].

REFERENCES

[1] R. Coppola, L. Ardito, and M. Torchiano. 2019. Characterizing the Transition to Kotlin of Android Apps: A Study on F-Droid, Play Store, and GitHub. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics (WAMA '19),*. ACM Press, New York, NY, 8–14.

[2] G. Costantino, F. Martinelli, P. Santi, and D. Amoruso. 2012. An Implementation of Secure Two-Party Computation for Smartphones with Application to Privacy-Preserving Interest-Cast. In *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM Press, New York, NY, 447–450.

[3] T. Johnson and P. Seeling. 2012. Localization using Bluetooth device names. In *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*. ACM Press, New York, NY, 247–248.