**Remote Monitoring of Android Devices Using the Inca Framework**

Fabian Lema

**Abstract**

In a large WSN, controlling the state of health of individual sensors' services is extremely challenging. The objective of this project is to perform remote monitoring of Android-powered devices using the Inca Grid Monitoring Framework. The project was divided into two parts, the first part aimed to the establishment of communication between an Android service and a SL4A script. The purpose, to check if it was possible to easily get a service's state information using simple scripts. The second part consisted in implementing Inca in the Android platform. Inca is written in Perl and though it uses many architecture independent Perl modules, some modules needed to be cross-compiled with the arm-linux-eabi-gcc toolchain. The Inca Reporter Manager installed in the phone communicated and performed a test with an Inca Server which demonstrated that it is possible to run Inca in Android-powered devices. However, the settings used to run Inca only last until the device reboots.

**Introduction**

A Wireless Sensor Network (WSN) acts as getaway between the physical and the digital world as it consists of autonomous sensors used to monitor physical and environmental conditions of certain area. Currently, the best candidates for autonomous sensors are smartphones running on Android OS due to their low cost, versatility, and high connectivity (WiFi, Bluetooth and cellular networks). However, a WSN made up of such complex resource-constrained platforms presents challenges when verifying its correct functioning. Every remote sensor runs several different services at any given time (data acquisition, storage, processing, reception, transmission, etc.) which makes the checking for performance issues, or failures of individual services in clusters of remote sensors a very difficult task. The purpose of the project is to adapt the Inca Grid Monitoring Framework (http://inca.sdsc.edu) to run in mobile devices running Android OS. Inca, developed at the San Diego Super Computer Center, is a

software which purpose is to detect infrastructure issues by executing periodic, automated, user-level testing of Grid software and Services. Adapting the Inca framework to Android devices allows the remote monitoring of the health of the deployed hardware platform (smartphone, and sensors) as well as critical services in an automated and scalable fashion.

**Methods**

The project was divided into two parts. The first part consisted in learning about the platform, understanding how services work on Android, how to implement them and how to get information from them. The second part was the implementation of the Inca Framework on the phone.

**Materials**

Implementing the Inca Grid Monitoring Framework on an Android device required many different tools. The main being the smartphone, Samsung Nexus Galaxy. The first part of the project required application development, in order to achieve this the programming languages Java and Python were used on the Integrated Development Environment, Eclipse. Python was used on Android through the Scripting Layer for Android (SL4A). Development in Android requires the use of the Android SDK, and ADT Eclipse plug-in. The communication with the phone via command line was possible thanks to the Android Debugging Bridge (adb).

The second part of the project required compilation tools, as Inca is a fully developed software and the objective was to run it in the phone, a slightly different platform. To achieve this, the adb tool was heavily relied on, also the Android NDK which allows the compilation of programs written in languages different than Java to run on Android-powered devices. The Nexus Root Toolkit was used to root the phone. The X-plore app installed as it that allows the exploration of directories in a rooted phone including the root directory and it also allows to set and change permissions. Perldroid and several Perl modules required by Inca were installed in the phone. The project was conducted initially on Windows

but later moved to Linux due to the greater versatility that Bash shell has in comparison to command prompt.

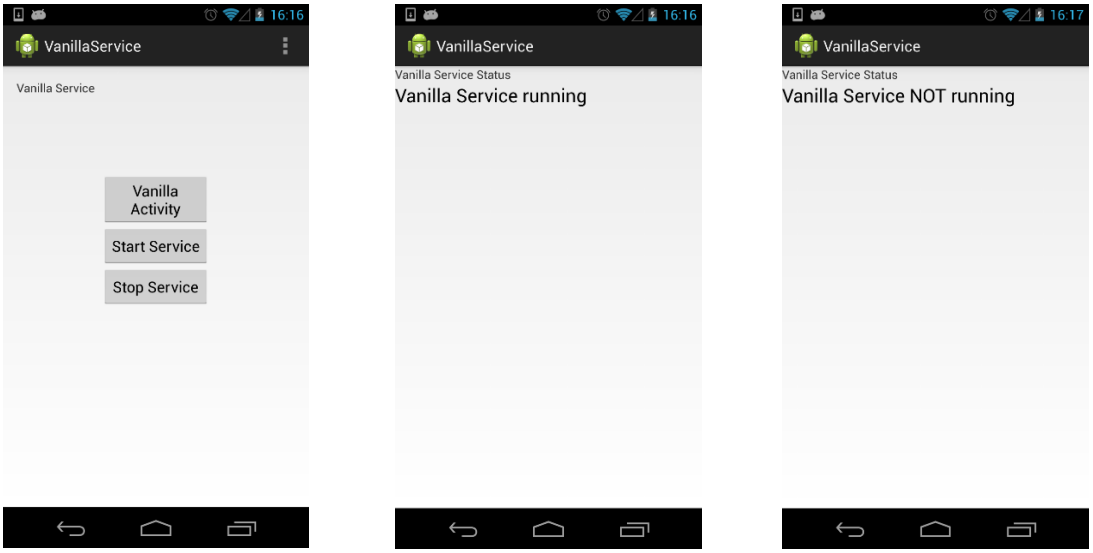
**Procedures**

The main goal of the first part of the project was to check if a Python script could communicate and retrieve information from a Service running on the phone. To achieve this purpose, an application (Vanilla app) was developed in Java using Eclipse and the Android SDK, the app sole purpose was to run a Service that would send information of the amount of time it had been running. With the help of videos on YouTube and several resources available online the completion of the service was achieved. A service is a process that runs in the background without having any interaction with the user. An example of a service is the Twitter feed in the Twitter app, once the user starts the application what is displayed in the feed is the activity of the people whom the user follows. After some time, the feed refreshes itself and shows new activity. This auto-refreshing feature is achieved through a service that pulls information from the Twitter website periodically, note that the user does not have to press the refresh button.

In the process of developing this simple Vanilla app, differences between Services, Activities and Intents were learned. Activities are the pages that the user interacts with, Intents as their name says show the intentions of the user, descriptions of operations to be performed. The Vanilla app has two activities, the first activity is the main menu, it presents three buttons, the first button launches a new activity that allow the user to check whether the vanilla service is running, the second button starts the Vanilla service if the service was not already running, and the third button kills the service. The service itself uses a sticky Broadcast to send information. Broadcasts send information from within the app to the whole environment, but they can only be caught by a Broadcast receiver that is listening for a particular transmission. Sticky broadcasts differ from broadcasts in that their intents remain in the system after being handled by a broadcast receiver, broadcasts' intents disappear in comparison (techtopia).

To use sticky Broadcasts following code has to be added to the Android Manifest

```
<uses-permission android:name="android.permission.BROADCAST_STICKY"/>
```




*Vanilla Service app, (left) main Activity, (middle) Second Activity when Service running, (Left) Second Activity when Service not running*

The application can be tested in a virtual phone in Eclipse, this is possible thanks to the ADT plug-in and the Android SDK. To install the application to the phone, first plug in the phone to a computer, go to Settings/ Developer Options/ and check USB debugging. Using adb in command prompt or bash shell type:

```
$adb install /LOCATION_OF_PROJECT/VanillaService.apk
```

The application is now available in the phone. The development of Android applications in Java is rather long, to accelerate and simplify this process SL4A was used to create a Python script that would get the information that the Vanilla app transmits. “Scripting Layer for Android (SL4A) brings scripting languages to Android by allowing you to edit and execute scripts and interactive interpreters directly on the Android device” (code.google). SL4A can be found in the Google Play store, once installed, add the Python and Perl interpreters and then install them.

Inca has several parts, Reporter Manager, Reporter Agent, Depot, Suites and Reports. In a nutshell, a user sends a suit (test) to the Inca Agent, the Inca Reporter Manager will receive this suite and will perform the test specified by such suit. The results of the test will be saved on a XML file known as the Report. The Reporter Manager then sends the Report to the Depot were the Report can be accessed from the web.

At this instance in the project, the only that we focused on are the Inca-Report modules which are available for Perl and Python which provide a standard to organize the information generated from the tests. Shava Smallen, the head programmer of Inca provided me with the Inca-Reporter Python and Perl modules. SL4A allows the usage of external modules on its scripts which facilitates the development of applications. The first major challenge came when trying to install the Python modules in the phone. From the provided file, the attempts to install the modules in the phone using the provided setup.py where completely unfruitful. Out of ideas, I installed the Inca-Reporter modules in my computer, the installer created a directory named /inca which contained Python binaries (.PYC) and the modules (.PY). This folder was copied to the phone with the help of adb.

```
$adb push /LOCATION/inca /sdcard/sl4a/scripts
```

My mentor, Sameer Tilak noticed that this approach could be wrong since the Inca Reporter binaries were compiled for an architecture different than Android, but in fact this approach worked and would keep on working later in the project.

For easiness the Python Eclipse Plug-in was installed following the guidelines found in the book *Pro Android Python with SL4A* by Paul Ferril. From this source many of the APIs were learned and after investing some time trying small blocks of code the communication between the Python script and the Java service was achieved. The main problem was listening for the Broadcast, SL4A does not possess an explicit Broadcast Receiver and the description of the API are somewhat vague. The SL4A function that is analogous to a Broadcast receiver is eventRegisterForBroadcast().

*Communication between Service and Script*

```
String str2 = "Transmiting from MyServices";
Intent sendIntent = new Intent();
sendIntent.setAction("com.example.helloworld.Broadcast");
sendIntent.putExtra(Intent.EXTRA_TEXT, str2);
sendBroadcast(sendIntent);
```

*Sticky Broadcast*

```
ACTION = "com.example.helloworld.Broadcast"
e=droid.eventRegisterForBroadcast(ACTION, False)
```

*Broadcast Receiver*

To use the Inca-Reporter modules import them in the header of the script, for the testing purposes SimpleUnitReporter was used.

```
from inca.SimpleUnitReporter import SimpleUnitReporter
reporter = SimpleUnitReporter(
  name= 'IRSimpleTest',
  version= 0.0,
  description='This is a test reporter',
  url='http://inca.sdsc.edu/releases/2.5/repdocs/pyhton.html',
  unit_name='checkServiceRunning'
)
```

The reporter object then can use all of the methods of the SimpleUnitReporter class. eventWait returns a JSON object, the idea was to put the extract the information from the broadcasted intent, and put it in the log of the reporter. However, the event rt is a JSON object which has three fields, name, data and time. The field ‘data’ looks like another JSON object, but is treated like a string, which makes impossible the extraction of the time the service has been up.

*Getting information from the broadcast and storing it in the Reporter log*

```
rt= droid.eventWait(1).result
if rt == None:
    reporter.unitFailure("Service not running")
    droid.log("Reporter failure")
    reporter.log('info','Service Failure')

else:
    print rt.values()
    reporter.unitSuccess()
    reporter.log("info","Reporter Success")
    e = rt['data']
    reporter.log('info', rt['data'])
    reporter.log('info', e[24])
```

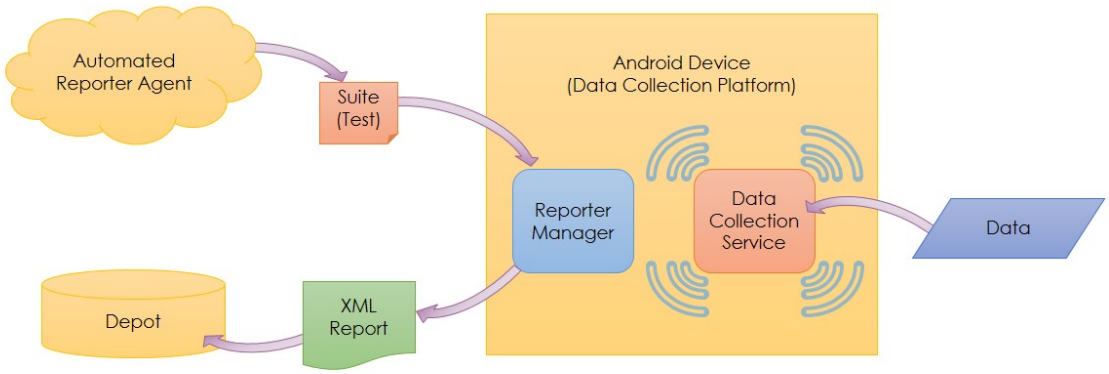
*Wait for the event to occur, if it does write to the Reporter log "Success" and rt['data'] to the log, rt.data.action generates an error.*

```
<info>
  <gmt>2013-07-17T07:28:06Z</gmt>
  <message>Reporter Success</message>
</info>
<info>
  <gmt>2013-07-17T07:28:06Z</gmt>
  <message>{"android.intent.extra.TEXT":2541849,"action":"com.vanillaservice.Broadcast"}</message>
</info>
<info>
  <gmt>2013-07-17T07:28:06Z</gmt>
  <message>E</message>
</info>
```

*Output, e has the value of rt.data, when e[24] is added to the log, the 24$^{th}$ letter of the rt.data is printed in message. Note that 2541849 is the Service's up time in seconds.*

Although precisely extracting the information broadcasted by the Vanilla Service was not successful, it is proven that it is possible to catch a transmission from a Service running on the phone and get some information easily using SL4A APIs.

The second part of the project was the actual implementation of the Inca Grid Monitoring Framework on the Android device.




1. The Reporter Agent sends a Test to the Report Manager.
2. The Report Manager then listens to a broadcast from the Data Collection Service to get information about the state of the Service.
3. The Report Manager creates an XML Report with the results of the Test and stores it in the Depot.

The installation of Inca was attempted several times, by following the instruction found in inca.sdsc.edu. Both standard ways of software installation in bash and Perl presented issues. In both instances the installation of Inca uses 'make' which as ubiquitous as is in Linux, is missing in Android. Using adb was too constrained as by default the root directory is hidden from the user. To overcome this issue the phone was rooted using the Nexus Root Toolkit. Once the device was rooted, BusyBox was installed. This package installs a number of applets which enhance the operability of the device from bash shell. cat, chmod, clear, cp, crond, diff, echo, grep, gunzip, gzip, and ls are among the standard applets installed with BusyBox. Android limits what directories the user has access to, punctually the user has full access to /sdcard all other directories are constrained in one way or another. The app X-plore was installed to browse the phone, later one this app would be of tremendous help as it can override file permissions.

A semi fully functional shell was then available, however this did not help as it was still lacking of the make command. At that point, the approach of copying modules sounded very good again, especially given that Inca is fully written in Perl, as long as the modules that Inca uses are located in the phone, Inca should work. The project was moved changed from platforms from Windows to Linux as Linux bash

shell is much more user friendly than command prompt. Moreover, I thought that by installing Inca on Linux the risk of compilation issues with Android would lessen since Android is based on a Linux Kernel. Inca can be downloaded for free from its website, the Inca Reporter Manager was installed in a computer using the standard Perl installation

```
$perl Makefile.PL
$make
$make install
```

All the generated directories were then copied to the phone to /sdcard/ReporterManager. After talking to Shava Smallen, she told me she would create server and told me install the following Perl modules:

XML::Namespace XML-NamespaceSupport-1.09 PERLSTANDARD
XML::SAX XML-SAX-0.14 PERLSTANDARD
XML::Simple XML-Simple-2.14 PERLSTANDARD
Schedule::Cron Schedule-Cron-0.97 PERLSTANDARD
Module::Build Module-Build-0.2609 PERLNON
Date::Manip Date-Manip-5.54 PERLSTANDARD
Params::Validate Params-Validate-0.91 PERLSTANDARD
URI URI-1.35 PERLSTANDARD
Sub::Uplevel Sub-Uplevel-0.09 PERLSTANDARD
Test::Exception Test-Exception-0.20 PERLSTANDARD
BSD::Resource BSD-Resource-1.24 PERLSTANDARD
Time::ParseDate Time-modules-2003.1126 PERLSTANDARD
Log::Log4perl Log-Log4perl-0.51 PERLSTANDARD
Log::Dispatch Log-Dispatch-2.10 PERLNON
Log::Dispatch::File::Stamped Log-Dispatch-File-Stamped-0.06 PERLSTANDARD
Net::Domain libnet-1.19 PERLSTANDARD
Curses Curses-1.24 PERLSTANDARD
Term::ReadKey TermReadKey-2.30 PERLSTANDARD
Test::Pod Test-Pod-1.26 PERLSTANDARD
Curses::UI Curses-UI-0.9607 PERLSTANDARD

With the exception of BSD::Resource and Params::Validate all the modules are architecture independent, all them were placed in /sdcard/com.googlecode.perlforandroid/extras/perl/site_perl BSD::Resource and Params::Validate needed C in order to compile so after reading on the subject the decision to cross-compile Perl 5.16 was made. The SL4A project was running Perl 5.10, with the hopes that a more recent version of Perl would come with more standard modules, following the instructions found on how to compile Perl 5.16 for Android found in the Perldroid website (code.google.com/p/perldroid/) arm-perl-5.16 was successfully pushed to the phone.

The expectations were not met, however, with the help of Hoang Nguyen, my host mentor, David Abramson, PhD. Student, I was able to cross-compile the module Params::Validate. Hoang Nguyen noticed that cross-compiling Perl 5.16 made use of the Android NDK, so after finding an online resource about compiling open source libraries with Android NDK (blog.jimjh.com), and script was made to change the default location the C compiler. Perl modules at installation by default look into the Perl and C compilers of the host machine to compile the module. This is an efficient approach as the modules are guaranteed to work on the machine. The script re-writes the location of the C compiler (where Build and Build.PL should look for the Perl and C compilers) redirecting them to the compilers found in the Android NDK, this way the modules compiled with the new Build and Build.PL will use the arm-linux-androideabi toolchain, and work in the phone.

*build.sh Script*

```
#!/bin/sh
# FourierTest/build.sh
# Compiles fftw3 for Android
# Make sure you have NDK_ROOT defined in .bashrc or .bash_profile

NDK_ROOT=$HOME/android-ndk
export PATH="$NDK_ROOT/toolchains/arm-linux-androideabi-
4.4.3/prebuilt/linux-x86/bin/:$PATH"
export SYS_ROOT="$NDK_ROOT/platforms/android-14/arch-arm/"
export CC="arm-linux-androideabi-gcc --sysroot=$SYS_ROOT"
alias cc="arm-linux-androideabi-gcc --sysroot=$SYS_ROOT"
export LD="arm-linux-androideabi-ld"
alias ld="arm-linux-androideabi-ld"
export AR="arm-linux-androideabi-ar"
alias ar="arm-linux-androideabi-ar"
export RANLIB="arm-linux-androideabi-ranlib"
alias ranlib="arm-linux-androideabi-ranlib"
export STRIP="arm-linux-androideabi-strip"
alias strip="arm-linux-androideabi-strip"
# ./configure --host=arm-eabi --build=i386-apple-darwin10.8.0 LIBS="-
lc -lgcc"

# make
# make install
export PERL_BIN=/home/fabian/hostperl-5.16.0/bin/
${PERL_BIN}/perl Build.PL
${PERL_BIN}/perl Build
exit
```

The compilation of Params::Validate was achieved and once this modules were stored in the phone. The following steps are required for Inca to work.

From a console and a computer that has the Android Development Kit type:

```
$adb shell
```

You should now see:

```
shell@android:/ $
```

The device has to be rooted, type 'su' to enter bash shell as Super User

```
root@android:/ #
```

The easiest way to realize the change is by seeing the dollar symbol ($) change to pound (#).

Make sure the folder /data/local/gcc has the following files and directories:

```
root@android:/ # ls /data/local/gcc/
R/ libexec/
arm-elf-linux-androideabi/ lost+found/
bin/ share/
include/ tmp/
lib/
```

Perl and its components are located in /sdcard/perl/ and contains:

```
Inca/ bin/ lib/
```

The best location to store to the Perl components is to save them in /sdcard because this part of the memory does not go back to its original state on reboot. Note that files in this directory cannot be executables, therefore commands like chmod 777 <file> have no effect.

To add Perl type:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/sdcard/perl/lib/arm-linux-androideabi/CORE/
```

which is the address of libperl.so

The Perl modules that are used by the Inca Reporter Manager are located in

```
/sdcard/com.googlecode.perlforandroid/extras/perl/site_perl
```

Add this to the PERL5LIB variable

Type `export PERL5LIB=/sdcard/perl/lib:$PERL5LIB` to add the Perl standard modules

Now typing perl -version in shell should give the following output

```
root@android:/ # perl -version

This is perl 5, version 16, subversion 0 (v5.16.0) built for arm-
linux-androideabi

Copyright 1987-2012, Larry Wall

Perl may be copied only under the terms of either the Artistic License
or the GNU General Public License, which may be found in the Perl 5
source kit.

Complete documentation for Perl, including FAQ lists, should be found
on this system using "man perl" or "perldoc perl". If you have access
to the Internet, point your browser at http://www.perl.org/, the Perl
Home Page.
```

Perl is ready now, let's proceed to set up the Inca Framework. The Inca Reporter Manager can be found

at /sdcard/ReporterManager. Inside this directory you will find the following directories:

```
root@android:/sdcard/ReporterManager # ls
Inca-ReporterManager-10.12447
bin
build.log
buildRM.sh
etc
lib
man
sbin
```

Remember that the files stored in /sdcard cannot be executables, which is why using a tool like the app

X-plore, manually copy the contents of /sdcard/ReporterManager/bin to /system/bin and make them

executable

```
root@android:/ # ls /system/bin/ -la|grep inca
-rwxrwxrwx root root 19396 2013-08-22 17:50 inca-approve-changes
-rwxrwxrwx root root 4896 2013-08-22 17:50 inca-null-reporter
-rwxrwxrwx root root 5598 2013-08-22 17:50 inca-ping-client
```

```
-rwxrwxrwx root root 13284 2013-08-22 17:50 inca-run-now
```

Do the same with the files in /sdcard/ReporterManager/sbin to /sbin

```
-rwxrwxrwx root root 46394 2013-08-27 12:15 inca
-rwxrwxrwx root root 11550 2013-08-27 12:15 reporter-instance-manager
-rwxrwxrwx root root 19975 2013-08-27 12:15 reporter-manager
```

and the files in /sdcard/ReporterManager/etc to /etc

```
drwx------ root root 2013-08-22 17:53 common
-rwxrwxrwx root root 591 2013-08-22 17:54 log4j.properties
-rwxrwxrwx root root 0 2013-08-22 17:53 rmkey.pem
-rwxrwxrwx root root 0 2013-08-22 17:53 rmreq.pem
```

Create the directory /bin in /root, note this folder will be erased every time the phone reboots.

In shell type the following command:

```
$mount -o bind /system/bin/ /bin
```

This will create a link between the /system/bin/ and /bin, and it is necessary as some of the Inca

Reporter Manager modules and other Perl modules import /bin/sh

Mount usr/bin/ with bin and create a /var directory in /sdcard, this directory will be used to save locally

the logs generated by Inca.

Shava Smallen provided me with a test which was slightly modified:

```
perl sbin/reporter-manager -d inca://vm-
ops020.alamo.futuregrid.org:6324 -e /bin/inca-null-reporter -r
/sdcard/var/reporter-packages -R /sbin/reporter-instance-manager -v
var -w 1 -i android-test1 -L DEBUG -P false -a inca://vm-
ops020.alamo.futuregrid.org:6323 -l /sdcard/var/rm.log
```

**Results**

The Inca Reporter Manager was successfully installed in the phone, logs were saved in the phone and

then sent to the web where they can be accessed:

http://vm-ops020.alamo.futuregrid.org:8080/inca/HTML/rest/android/androids

The test was simply asking for the version of Perl installed in the phone. The accomplishment of such trivial task demonstrates that it is possible to implement Inca on an Android-powered device. In this case the Inca Reporter Manager in the smartphone was able to connect to a server and send information about its state, in this case the Perl version it has.

**Discussion**

It is very important to realize of the biggest constraint of the Android platform when intending to execute programs, which is that upon reboot all the variables and folders created in the device under the /root directory are erased. In other words, the /root directory resets to its original state. The process of setting up the environment to run Inca is non-trivial and it only lasts until next reboot. More testing is required to see to what extent Inca can be used to perform remote monitoring of the phone. Moreover, regular Inca commands do not always work and Perl has to be explicitly invoked when running Perl executables and the user has to figure out how to use the commands e.g. going to the folder where the executable is and executing it directly.

**Acknowledgements**

I want to thanks the Director of PRIME, Gabriele Wienhausen, PRAGMA Chair, Peter Arzberger, and the PRIME Program Manager, Teri Simas, for giving me the opportunity of participating in research abroad and supporting and guiding me through every step of the process of getting to Australia. I also want to express my gratitude to my mentor in the University of California, San Diego, PhD. Sameer Tilak, and my host mentor at the University of Queensland, PhD. David Abramson, for giving me the chance to work with them and providing me real-world experience. Last but not least, I want to thanks the Inca Project lead, Shava Smallen, for helping me throughout every part of the process of implementing Inca on Android, and Hoang Nguyen for taking the time to help me in every instance I was stalled.

**Sources**

*Android Broadcast Intents and Broadcast Receivers*, web:

http://www.techotopia.com/index.php/Android_Broadcast_Intents_and_Broadcast_Receivers#Sticky_Broadcast_Intents

*Android File System Hierarchy*, web: http://anantshri.info/andro/file_system.html

*Android Scripting*, web: http://code.google.com/p/android-scripting/

*Compiling open source libraries with Android NDK: Part 2*, web: http://blog.jimjh.com/compiling-open-source-libraries-with-android-ndk-part-2.html

*Inca*, web: http://inca.sdsc.edu/

*Perldroid*, web: http://code.google.com/p/perldroid/wiki/Compiling

Ferril, Paul. *Pro Android Python with SL4A*, New York: Apress, 2011. Print.

*SL4A API Help,* web: http://www.mithril.com.au/android/doc/