*Display of Scientific Image Sources With Mobile Devices*

**Ryan Dens**

Allendale Columbia School

Rochester, New York

Advisor: Douglas Jacobs-Perkins

**Laboratory for Laser Energetics**

University of Rochester

Rochester, New York

December 2014

## 1. Abstract

Software infrastructure is being developed for mobile devices to enable scientists and technicians to view live images captured by cameras in the experimental areas of the Laboratory for Laser Energetics (LLE). Mobile display devices are desirable because users can see the images wherever wireless connectivity is available. However, they present significant software design challenges for efficient bandwidth management, visual presentation and ease of use. A rudimentary application was developed for Android devices. The application can display static images with captions and extract and graph line intensities from the image. This application provides the foundation to acquire and display live images and provide basic image analysis tools. The availability of live image display on mobile devices would allow the result of any repair or adjustment to be viewed nearly instantaneously.

## 2. Introduction

Currently, there is no secure way for LLE scientists and technicians to access live image and video feeds quickly and efficiently from a mobile device. LLE is developing many of its image acquisition software applications, such as the SI-800 scientific-grade camera service, as "services" that share data using ZeroC's Internet Communications Engine (ICE). ICE is available for many operating systems, including common mobile devices, and allows cross-platform communication. This enabling technology is being pursued as the basis for real-time image display on mobile devices at LLE.

Through this research, an Android application was developed. The application, with an appropriate username and password, accesses images from the OMEGA Shot Images and

Reports, based on a Uniform Resource Locator (URL). With an appropriate URL, username, and password, the application can load a static image, generate relevant graphs and add meta-data, such as image and graph captions. This application is the foundation for a comprehensive application for the selection and display of live image sources on mobile devices.

Sections 3 and 4 discuss the choice of platform for the application development, followed by an examination of the engineering of the application in section 5. Sections 5.1 and 5.2 specifically relate to how common issues, such as data transfer within the application and asynchronous image downloading, were resolved.

## 3. Determination of the Optimum Platform for the Application

A variety of decisions had to be made in order to determine how the application was to be developed. Considerations included taking advantage of the benefits of smartphones while minimizing the effect of their disadvantages, determining which programming language and operating system were best suited to the goals of the project, and how to develop the application in the decided manner.

### 3.1 Advantages and Disadvantages of Mobile Devices

The premise of the project was to enable display of scientific image sources on mobile devices. The advantages and disadvantages of mobile devices compared to a traditional personal computer had to be considered in order to determine the most effective manner to present the greatest amount of information possible with an intuitive interface. Compared to traditional personal computers, mobile devices often have much smaller screens, slower hardware, and

limited bandwidth over wireless internet networks. But, mobile devices also have touch screens and many built in sensors, such as accelerometers, that personal computers do not have that could be used to create a more intuitive interface for the user. Additionally, developing an application for mobile devices is useful because the devices can be taken anywhere due to their small size and long battery life.

This in particular offers the greatest advantage to technicians in the target bays of OMEGA and OMEGA EP. The laser and target bays are clean room work areas without many computers. Since not all available computers can access images from all cameras, live image display capabilities on mobile devices would be useful for performing tasks such as alignment operations. Additionally, it would also offer faster and mobile access for scientists running experiments at the LLE.

**3.2 Determination of a Suitable Programming Language**

Three different programming languages were considered when determining what language the application was to be initially developed in: HTML5, Java, and Objective-C. Java and Objective-C can be used to develop native applications for Android and iOS operating systems, respectively. HTML5 is a markup language that can be used to develop a web-based application.

First, the advantages and disadvantages of a web-based application compared to a native application were considered. Web-based applications offer a broader range of supported devices and higher inherent security, whereas native applications offer faster rendering of the application,

smoother response to touch gestures, and direct access to the mobile device's built-in hardware resources and sensors. Both of the advantages of web-based applications simply reduce the amount of work required for the final product, since the application could be re-developed in additional languages to support more devices and, with the proper tools, the security advantages of a web-based application over a native application would be minimal or nonexistent. Therefore, it was determined that a native application could offer a better user experience than a web-based application. Of the two types of programming languages considered for developing a native application, an Android application programmed in Java was chosen due to the developers' familiarity with Java and the cost associated with developing Objective-C applications.

**4. Application Development tools**

Since LLE does not have an in-house expert on mobile device programming, software tools for the development of Android applications were first explored. First, the Eclipse Integrated Development Environment (IDE) was installed along with Android Developer Tools (ADT) and the Android Software Development Kit (SDK). Some additional libraries were required to be installed in order to support the Android SDK on a 64-Bit operating system, as described on the Android developer webpage. Since the project's completion, the Android Studio has become the official IDE for Android application development. While this was tested on an LLE laptop, seemingly without any issues, Eclipse was used because, at the time, it was the IDE used by a vast majority of Android developers. However, for those involved in extending this

project at LLE, it is recommended that the existing project be transferred from Eclipse to Android Studio in order to stay current with respect to the Android SDK.

The first task undertaken was to understand how basic Android applications are developed. Several applications were developed, such as a to-do list, in order to explore some aspects of the design philosophy of mobile applications. Due to the limited processing resources available on mobile devices, the Android operating system is relatively strict, when compared to personal computers, with regards to which applications are allocated resources. If an application is using too many resources, it will crash. Or, if the application has not been accessed for a significant amount of time, the application will stop.

The foundation of an Android application is its activities, which make up the user interface. In general, an Android activity is specified by a Java class and an XML file. The Java class specifies how the activity reacts to the user input, while the XML file specifies the layout of that screen of the application, including buttons, text, sliders, and most other static features of the application. Multiple activities can make up an application. For example, in the case of the application developed for LLE, there are four: Main, Image, Graph, and Edit Image.

**5. Application Activities**

Figure 1 shows the Main Activity, which consists of three Edit Text boxes, declared in the XML file, and a button. The first Edit Text allows the user to specify the image source with a URL. If that URL requires a secure login to access the image, the user can input his or her appropriate username and password.


**Figure 1. Screenshot of the Main Activity of the Android application.**
The activity consists of three Text Views, three Text Edits, and one button.

Figure 2 shows the Image Activity. While the view itself is relatively simplistic, the bulk of the system resources are consumed by this activity because, if required, high resolution images can be downloaded from the internet to the device. Due to their resolution, the image files downloaded are large and take up the majority of the processor time. The activity consists of a Web View, which displays the image, and a Text View, which holds the caption overlayed onto the image.

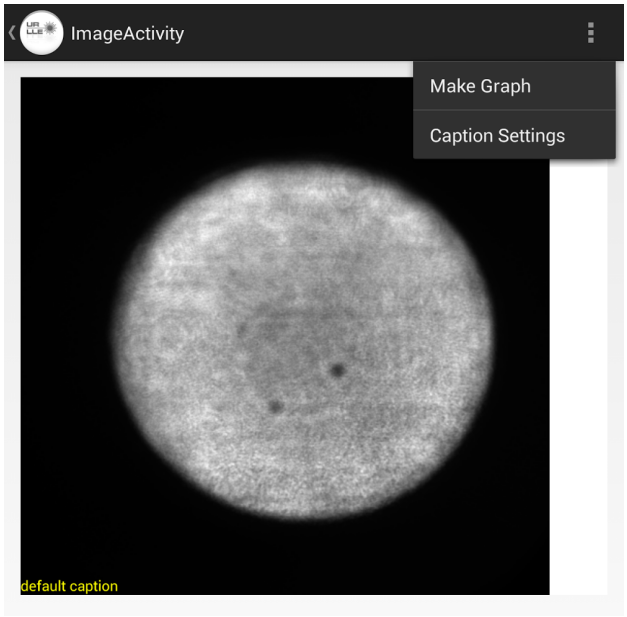


**Figure 2. Screenshot of the Image Activity from the Android application.**
The activity consists of a Web View and a Text View. The settings menu has options to start the Edit Image Activity and the Graph Activity,

Figure 3 shows the Edit Image Activity. Here, the user has the option to change the URL that points to the image source, overlay a caption onto the image, and position the caption. Since the user has already logged in, the option to input a new username and password is not provided. The new URL and caption are input using Edit Text boxes, as the URL, username, and password were specified before. However, in order to change the position of the caption over the image, the user must utilize the three Seek Bars, again provided by the Android SDK. One Seek bar specifies the width of the text box that contains the caption. If the user wished to split the caption into multiple rows, or to make it one single row, he or she must increase or decrease the width of the caption. The remaining two seek bars set the distance from the left and bottom sides of the screen. The indentation is based on a percent of pixels in a given column or row. For instance, changing the left margin Seek Bar's position to the exactly half ensures that the first letter of the

caption will be in the middle of the screen. This ensures compatibility across many devices, from small smart phones to large tablets.

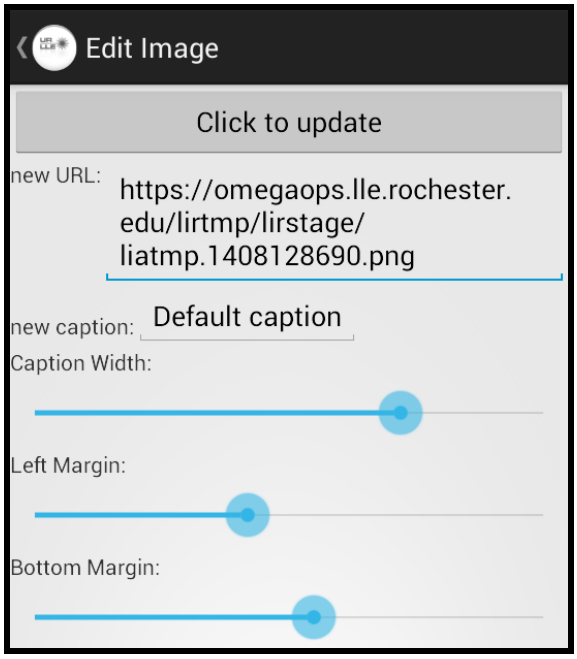


**Figure 3. Screenshot of the Edit Image Activity from the Android application**
The activity consists of one button, five Text View boxes, two Edit Text boxes, and three Seek Bars.

Figure 4 shows the Graph Activity. The graph is a chart of pixel intensity for the image specified by the URL. The Graph Activity is instantiated from the settings menu in Image Activity from an Alert Dialog. Users can set which column or row they wish to analyze in this dialog. The title of the graph can also be set from a secondary Alert Dialog. In order for the graph to be generated, the image displayed in the Web View of Image Activity is saved as a BitMap. The bitmap, column or row specifications, and graph title are then pushed to the Graph Activity. The specified row or column of the image can be easily analyzed by analyzing the two-dimensional array of the BitMap. In order to display the data on a graph, the Graph View library was implemented. The Graph View library is an open source library for plotting graphs

on Android devices. A line graph was created, and the data obtained from the BitMap was transferred to the Graph View and displayed.

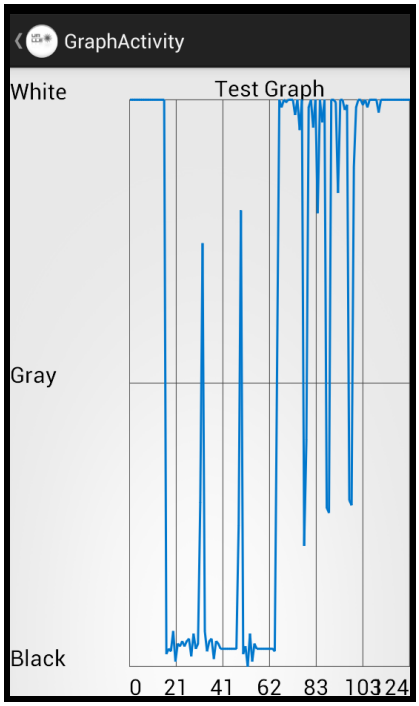


**Figure 4. Screenshot of Graph View Activity from the Android application**

### 5.1 Starting Activities and Transferring Data

Developers must be particular in the way they allow the user to travel from one activity to the next in order to not lose data or crash the application. The best manner to travel from one activity to another is using the "Intent" class, provided by the Android SDK. A basic diagram of how an intent is primarily used is shown in Figure 5. Here, the Intent instantiated in Activity 1 is being used to start and transfer information to either Activity 2 or Activity 3. As stated by the class documentation for the Intent class, "An intent is an abstract description of an operation to be performed". From the Main Activity, all other activities were instantiated using the functions of the Intent class.

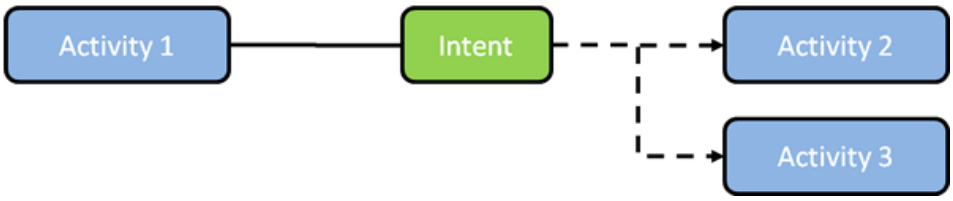


**Figure 5. Diagram Showing an Intent Starting Multiple Activities**

By following the instantiation of the Image Activity from the Main Activity, the capabilities of the Intent class can be better understood. First, upon the creation of Main Activity, a new instance of an Intent is created. The constructor used takes two variables: the activity, specified by the class from which the Intent is being created, and the activity, again specified by the class, that the Intent will start. By calling the function "startActivity", the current activity is safely and efficiently left while the new activity is created. If the developer were to simply attempt to change the view of the application, referring to the XML file, the application would crash. This occurs because the XML file specifying the layout of an activity is tied to the Java class that specifies how the activity functions. When only the view is changed, the Java class remains open and the application crashes.

The Intent class is not only able to safely and efficiently start new activities. It can also carry along user input from the previous activity to the new activity. This is done by calling the function "putExtra". At the most basic level, the function takes two arguments, represented as strings. One string is used as a key, while the other holds the data that the developer wishes to transfer. The key must be globally unique with respect to the project to ensure that the right data is transferred. All keys were instantiated as public, static, and final. This method must be called before the "startActivity" method in order to ensure that all data is transferred. There are several

other variations of the “putExtra” function that take booleans, integers, and even arrays. The function that passes arrays is used in the instantiation of the Graph Activity, but in the case of the instantiation of the Image Activity, only the function passing strings is utilized.

Again, examining the case of the instantiation of the Image Activity, once the function startActivity is called, the Main Activity is closed and the Image Activity is opened. From Main Activity, three values were passed to the Image Activity using the function “putExtra”: the URL, the username, and the password. However, these will not automatically be transferred, and therefore be able to be directly referred to, in the Image Activity class. Upon the creation of Image Activity, the function “getIntent” is called. This function takes no arguments, but simply searches for an Intent that was sent to it. The intent is found upon the creation of Image Activity because an Intent was required to create it. The function returns the Intent that was sent to it. From that, the intent can now be referred to in the new activity. Likewise, the new activity reads data by calling the function “getStringExtra” with a key. For instance, if one were to call the function with the key value that corresponds to the key for the URL, the string representing the URL would be returned.

**5.2 Proper Image Download on Android Devices**

While the views and actions of an Android application are divided by activities, the execution of a task is divided into threads. Tasks can be executed synchronously or asynchronously. Synchronous tasks are executed one at a time, in sequence. When the processor has executed one task, it moves to the next. These tasks are executed on the same thread, which is generally referred to as the user interface thread. This thread handles all the input from the user

including typing on the keyboard, reaction to physical buttons, or the touch screen. Asynchronous tasks, however, are executed on separate threads, so one task can begin before a task on a separate thread ends. Since there is generally only one processor, all threads are given a piece of processor time and are alternated between, allowing all threads to remain active.

As previously discussed, downloading an image from the web demands a large portion of processor time. Because of this, time-consuming operations must not be done while running an activity, or they will make the activity slow or become unresponsive. Therefore, the image must be downloaded on a separate thread, or asynchronously. On Android devices, asynchronous tasks are primarily executed by implementing the class AsyncTask, provided by the Android SDK. When an instantiation of AsyncTask is executed, three methods are called: onPreExecute, doInBackground, and onPostExecute, as shown in figure 6.

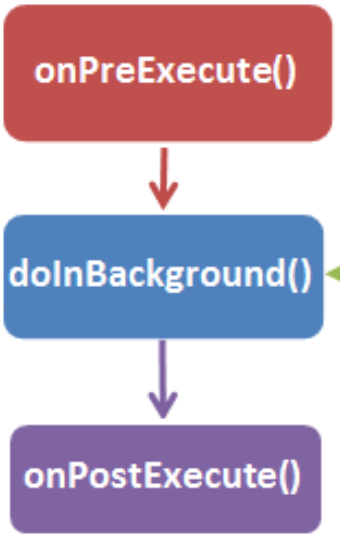


**Figure 6. Flow chart of methods for the class AsyncTask.**
The diagram represents the order in which the three methods, onPreExecute, doInBackground, and onPostExecute, are called.

In order to download the image asynchronously, AsyncTask was overridden as a private class inside the Image Activity. All three methods, onPreExecute, doInBackground, and

onPostExecute, are overridden inside this class. The function onPreExecute is executed first on the user interface thread. This method is used to set up the task that will be done on a separate thread. Next, the function doInBackground is called. This occurs on a new thread, separate from the user interface thread. This is also where the primary part of the task is executed. In the case of the Android application developed for LLE, this is where the image is downloaded as a Bitmap. Finally, the function onPostExecute is called, again on the user interface thread. This uses the result of the task executed in the background, in this case, to display the image to the user.

## 6. Conclusion

An Android application was developed for the display of static image sources on mobile devices. The application currently can access images specified by a URL from the OMEGA Shot Images and Reports webpage by an LLE employee with access to the OMEGA Shot Images and Reports. The application can generate a graph of pixel intensity with appropriate captions as well as overlay meta-data onto the downloaded images. This application is the foundation for a comprehensive application for the selection and display of images from live sources, such as the SI-800 Streak camera, with mobile devices.

Although there were security concerns, two-way communications between an Android device and image sources at the LLE can be achieved by utilizing ZeroC's Internet Communications Engine, referred to as ICE. ICE allows secure communication between client and server applications across multiple operating systems, including Android and iOS, the most popular mobile operating systems. ZeroC's ICE is already widely used at LLE. A client

application, similar to the Android application already created, will be able to communicate with an image service housed on a server at LLE using ICE. The image service will compress images and execute other resource intensive tasks that could not be executed on a smartphone. The image service will also prepare meta-data such as image and graph captions. The client application will be able to select and display image sources, including live image sources. It will also generate relevant graphs, apply colormaps to transferred images, and execute related image processing tasks.

## 7. Acknowledgements

I would first like to thank Dr. Stephen Craxton for interviewing me for and inviting me to LLE's Summer Research Program for High School Juniors**.** I would also like to thank Michael Charissis for providing technical support for setting up the Android IDE on the LLE computers. Additionally, I would like to thank Mr. Richard Kidder and Tyler Meyer for assisting with accessing ICE at LLE. Finally, I would like to thank Dr. Douglas Jacobs-Perkins for his endless support and guidance, as well as the idea for this project.