Table of Contents

[Statement of work 6](#_Toc404371335)

[SOW Changes 6](#_Toc404371336)

[SOW Scope 6](#_Toc404371337)

[Key Assumptions 6](#_Toc404371338)

[General Responsibilities 6](#_Toc404371339)

[CSCI Responsibilities 7](#_Toc404371340)

[CLIENT Responsibilities 7](#_Toc404371341)

[CSCI Tasks and Deliverables 7](#_Toc404371342)

[SOW Completion Criteria 7](#_Toc404371343)

[SOW Delivery Acceptance Criteria 8](#_Toc404371344)

[SOW Estimated Schedule/Timeline 8](#_Toc404371345)

[SOW Charges (Time & Materials) and Roles 8](#_Toc404371346)

[Designated Contacts/Invoice Address 8](#_Toc404371347)

[Designated Client Contacts 9](#_Toc404371348)

[Designated CSCI Contacts 9](#_Toc404371349)

[APPENDIX A: SOW Change Control Procedure 10](#_Toc404371350)

[APPENDIX B: SOW Change Request Form 11](#_Toc404371351)

[Feasibility and Market Analysis 13](#_Toc404371352)

[Executive Summary 13](#_Toc404371353)

[Description of Products and Services 13](#_Toc404371354)

[Technology Considerations 13](#_Toc404371355)

[Product/Service Marketplace 14](#_Toc404371356)

[Marketing Strategy 14](#_Toc404371357)

[Schedule 14](#_Toc404371358)

[Findings and Recommendations 15](#_Toc404371359)

[Requirements 16](#_Toc404371360)

[Purpose 16](#_Toc404371361)

[Platform Requirements 16](#_Toc404371362)

[General Application requirements 17](#_Toc404371363)

[Wave OBSERVER feature 17](#_Toc404371364)

[Future Educational feature 19](#_Toc404371365)

[Use Case Diagram 20](#_Toc404371366)

[Data Flow Diagram 21](#_Toc404371367)

[Class Diagrams 22](#_Toc404371368)

[Spectrum Analyzer 22](#_Toc404371369)

[Spectrogram 23](#_Toc404371370)

[Source Code 24](#_Toc404371371)

[SpectrumAnalyzer.java 24](#_Toc404371372)

[SpectrumAnalyzerView.java 26](#_Toc404371373)

[AudioRecordListener.java 33](#_Toc404371374)

[AudioRecorder.java 33](#_Toc404371375)

[Complex.java 36](#_Toc404371376)

[FFT.java 37](#_Toc404371377)

[Spectrogram.java 40](#_Toc404371378)

[SpectrogramView.java 46](#_Toc404371379)

[SpectrogramFileIO.java 60](#_Toc404371380)

[SaveDialog.java 61](#_Toc404371381)

[APRAD.java 62](#_Toc404371382)

[SplashScreen.java 64](#_Toc404371383)

[About.java 64](#_Toc404371384)

[Help.java 65](#_Toc404371385)

[APRAD\_UCD manifest.xml 65](#_Toc404371386)

[JavaDoc 67](#_Toc404371387)

[Overview 67](#_Toc404371388)

[Class SpectrumAnalyzer 68](#_Toc404371389)

[SpectrumAnalyzer 69](#_Toc404371390)

[onCreate 70](#_Toc404371391)

[onDrawableSampleAvailable 70](#_Toc404371392)

[Interface AudioRecordListener 70](#_Toc404371393)

[onDrawableSampleAvailable 71](#_Toc404371394)

[Class AudioRecorder 71](#_Toc404371395)

[AudioRecorder 74](#_Toc404371396)

[run 74](#_Toc404371397)

[getMaxFFTSample 74](#_Toc404371398)

[end 74](#_Toc404371399)

[registerFFTAvailListener 75](#_Toc404371400)

[unregisterFFTAvailListener 75](#_Toc404371401)

[notifyListenerDrawSpectrum 75](#_Toc404371402)

[Class Complex 75](#_Toc404371403)

[Complex 77](#_Toc404371404)

[plus 77](#_Toc404371405)

[minus 78](#_Toc404371406)

[times 78](#_Toc404371407)

[real 78](#_Toc404371408)

[imaginary 79](#_Toc404371409)

[toString 79](#_Toc404371410)

[Class FFT 79](#_Toc404371411)

[FFT 80](#_Toc404371412)

[calculateFFT 81](#_Toc404371413)

[getMaxFFTSample 81](#_Toc404371414)

[fft 81](#_Toc404371415)

[Class SpecAnalyView 82](#_Toc404371416)

[SpecAnalyView 83](#_Toc404371417)

[SpecAnalyView 83](#_Toc404371418)

[drawSpectrum 84](#_Toc404371419)

[surfaceChanged 84](#_Toc404371420)

[surfaceCreated 85](#_Toc404371421)

[surfaceDestroyed 85](#_Toc404371422)

[APRAD User Manual 86](#_Toc404371423)

[APRAD Antenna: 86](#_Toc404371424)

[APRAD Smartphone Application: 86](#_Toc404371425)

[System Requirements: 86](#_Toc404371426)

[Installation: 87](#_Toc404371427)

[Using a computer: 87](#_Toc404371428)

[Using the mobile device: 87](#_Toc404371429)

[Using the Software: 88](#_Toc404371430)

[APRAD Splash Screen: 88](#_Toc404371431)

[Menu Screen: 88](#_Toc404371432)

[Spectrum Analyzer: 89](#_Toc404371433)

[Spectrogram: 90](#_Toc404371434)

[Settings Menu: 91](#_Toc404371435)

[Help: 96](#_Toc404371436)

[About: 97](#_Toc404371437)

# Statement of work

This Statement of Work (SOW) outlines the tasks required for CSCI to provide ***Autonomous Portable (low frequency) Receiver and Display (APRAD)*** softwarefor CLIENT’s hardware antenna***.*** CSCI shall perform in accordance with this SOW, which shall be effective from ***20 August 2013*** until ***16 May 2014***.

Any terms and conditions contained in this SOW are for purposes of this SOW only. The tasks to be performed by CSCI are defined and an estimated schedule is provided. In addition, the responsibilities of CLIENT are listed.

The following are incorporated in and made a part of this SOW:

* “Appendix A: SOW Change Control Procedure”
* “Appendix B: SOW Change Request Form”

## SOW Changes

Changes to this SOW will be processed in accordance with the procedure described in “Appendix A: SOW Change Control Procedure.”

The review and the implementation of changes may result in modifications to the SOW Estimated Schedule, and/ or SOW Charges.

## SOW Scope

CLIENT has requested that CSCI provide an Android application that can monitor waves via a spectrogram and spectrum analyzer. The data used to create the visuals can be saved for further analysis in a program such as Matlab. Additionally, the CLIENT has requested a wave simulator which can be used by students and instructors in an academic setting.

## Key Assumptions

This SOW and CSCI’s estimates to perform the SOW are based on the following key assumptions:

* Android OS will be the development platform.
* CLIENT will be available to answer any questions regarding the technology.
* CLIENT is responsible for designing and deploying the antenna monitoring system.

## General Responsibilities

General responsibilities specific to this SOW include, but are not limited to the following sub-sections.

### CSCI Responsibilities

CSCI will provide the following to support achieving a successful completion of this SOW:

* Manage all aspects of CSCI resources in support of achieving a successful completion of the work contemplated by this SOW.
* Provide adequately trained and otherwise qualified resources to create the deliverables and services under this SOW.
* Facilitate weekly status meetings with the EE teammates.

### CLIENT Responsibilities

CLIENT will provide the following to support achieving a successful completion of this SOW:

* Management of all CLIENT project team resources working in relationship with CSCI on this SOW
* Timely access to all CLIENT subject matter experts that CSCI determines are required to complete stated deliverables.

## CSCI Tasks and Deliverables

The following items will be delivered to CLIENT as a result of this task. CLIENT shall have seven days from submission of each Deliverable or the end of the Spring 2014 semester (whichever comes first) to inspect and inform Supplier of any material issues; otherwise deliverable is deemed acceptable and complete. Deliverables not listed are explicitly out of scope and will not be delivered unless mutually agreed upon by CSCI and CLIENT in accordance with the procedure described in “Appendix B. SOW Change Control Procedure”

CSCI will develop and document the following:

* Android Spectrum Analyzer
* Android Spectrogram
* Ability to save captured data

## SOW Completion Criteria

CSCI shall have fulfilled its obligations under this SOW when any of the following first occurs:

* CSCI accomplishes the “CSCI Tasks and Deliverables” (section 1.5) and CLIENT has indicated acceptance of agreed upon “CSCI Tasks and Deliverables” (section 1.5); or
* Either Party terminates this SOW by providing two weeks written notice; or
* The Estimated End Date is reached.

## SOW Delivery Acceptance Criteria

CSCI shall have fulfilled its obligations under this SOW when any of the following first occurs:

* CSCI accomplishes the CSCI tasks described under “CSCI Tasks and Deliverables”; or
* CLIENT notifies CSCI that further Services are not required; or
* this SOW is terminated pursuant to the Agreement.; or
* The Estimated End Date is reached.

## SOW Estimated Schedule/Timeline

Work is estimated to start on ***3 September 2013*** and complete on ***16 May 2014***.

## SOW Charges (Time & Materials) and Roles

The hours authorized by CLIENT and specified below do not imply or commit a fixed-price contract. If CSCI determines that it is necessary to exceed the hours or the Estimated End Date, CSCI will inform CLIENT as soon as practical. In such event, CLIENT may authorize additional hours. If CLIENT alternatively chooses to terminate the Services, CLIENT agrees to pay CSCI for actual hours expended.

|  |  |  |  |
| --- | --- | --- | --- |
| **Resource Role** | **Hourly Rate** | **Hours** | **Consulting Total** |
| **Software Developer** | $30.00 | 320 | $9,600.00 |
| **Software Engineer** | $30.00 | 320 | $9,600.00 |
| **TOTAL** | | | **$19,200.00** |

* Applicable federal, state and local taxes are not included in the charges.

## Designated Contacts/Invoice Address

The following shall be the Designated Contacts for this SOW.

## Designated Client Contacts

|  | **EE Sponsor** | **EE Contact** | **EE Contact** |
| --- | --- | --- | --- |
| **Name** | Mark Golkowski | Adam Perez | Nicolas Gross |
| **Address** | North Classroom, Room 3815-A |  |  |
| **Phone** | 303.352.3852 (office) |  |  |
| **Email** | [mark.golkowski@ucdenver.edu](mailto:mark.golkowski@ucdenver.edu) | [adam.perez@ucdenver.edu](mailto:adam.perez@ucdenver.edu) | [nicolas.gross@ucdenver.edu](mailto:nicolas.gross@ucdenver.edu) |

## Designated CSCI Contacts

|  | **CSC Sponsor** | **CSC Contact** | **CSC Contact** |
| --- | --- | --- | --- |
| **Name** | Debra Parcheta | Michael Dewar | Robert Perlstein |
| **Address** | LW-844 |  |  |
| **Phone** | 303.246.7926 | 720.413.5118 |  |
| **Email** | [debra.parcheta@ucdenver.edu](mailto:debra.parcheta@ucdenver.edu) | [michael.dewar@ucdenver.edu](mailto:michael.dewar@ucdenver.edu) | [robert.perlstein@ucdenver.edu](mailto:robert.perlstein@ucdenver.edu) |

## APPENDIX A: SOW Change Control Procedure

The following provides a detailed procedure to follow if a change to this SOW is required.

* A SOW Change Request (“CR”) is the only agreed upon vehicle for communicating change.
* Either party may determine the need exists for a change and request or prepare a CR for review by both parties.
* The CR must describe the change, the rationale for the change and the effect the change will have on the SOW including, but, not limited to: (i) Time; (ii) Cost; (iii) Deliverables; (iv) Dates.
* The Designated Contact of the requesting Party will review the proposed change and determine whether to submit the request to the Designated Contact of the other Party.
* Both Designated Contacts will review the proposed change and approve it for further review or reject it. If the review is authorized, the Designated Contacts will sign the CR which will constitute approval of the review. CSCI will invoice CLIENT for any such charges. The review will determine the effect that the implementation of the CR will have on price, schedule and other terms and conditions of the SOW.
* Both Designated Contacts and the authorized CLIENT representative must agree in writing to authorize the implementation of the approved changes.
* A sample SOW Change Request Form is attached as Appendix B.

## APPENDIX B: SOW Change Request Form

Each of us agrees to modify the referenced SOW (Business Intelligence Consulting) and any applicable terms as follows:

|  |  |
| --- | --- |
| **Change Request Details** | |
| **Change Number:** |  |
| **Project Name:** |  |
| **Description of Change(s):** |  |
| **Requested by:** |  |
| **Date Requested:** |  |

* All other terms and conditions of the SOW will remain unchanged.
* This offer will expire if not executed by DD Month YYYY.

Each of us agrees that the complete agreement between us about these Services consists of 1) this Change Request, and 2) the referenced SOW, and 3) the AGREEMENT.

|  |  |  |
| --- | --- | --- |
| Agreed to: |  | Agreed to: |
| **CSCI** |  | CLIENT |
|  |  |  |
|  |  |  |
| Authorized Signature |  | Authorized Signature |
|  |  |  |
|  |  |  |
| Name (type or print): |  | Name (type or print): |
|  |  |  |
| Date: |  | Date: |

# Feasibility and Market Analysis

The feasibility study is performed in order to assess the viability of the proposed project. It takes into account the scope of work for the given time frame, as well as complexity and difficulty of the project. It also considers budget restrictions that might make the project impractical. Finally, the analysis evaluates the need for development of new software based availability of current software that may be able to fulfill the requirements outlined for the project.

## Executive Summary

The Autonomous Portable low frequency Receiver and Display (APRAD) project will allow researchers and students to capture, analyze and simulate very low frequency (VLF) (300 Hz – 30 KHz). There are other solutions in the marketplace however the APRAD project is the first to create a low cost and portable solution.

## Description of Products and Services

The APRAD project consists of a hand held antenna and a mobile application to capture VLF electromagnetic waves for research and educational purposes. Until now, making a science grade observation of VLF electromagnetic waves involved a very large antenna, up to 300 feet of cabling and a large AC powered receiver. These large setups are susceptible to interference from the electrical power grid which can compromise the fidelity of the measurement. Finding an ideal location is critical. The APRAD device will provide a real time diagnostic tool to evaluate the quality of the electromagnetic environment in the VLF band. The results of the site survey can be saved for further analysis with outside software such as MATLAB. Additionally, the APRAD application will have the capability to simulate electromagnetic waves generated in the application for educational purposes. The APRAD device is not limited to a field application as there are other uses for the VLF bands since they are not licensed by the government. The APRAD device could be used to detect interference when installing a RFID device since RFID also communicates in the VLF band.

## Technology Considerations

The application will be built for a Galaxy SII using Android 4.1.2 (Jelly Bean, API 17) as the minimum required SDK. This allows the application to be used on any modern device running a current version of the Android OS. If desired, the application has the potential to be ported over to the IOS or Windows platform. There are various spectrograms and spectrum analyzers available on the Android Marketplace but this will be the first application to combine a spectrogram, spectrum analyzer and a wave generator/visualizer.

## Product/Service Marketplace

There are many spectrograms and spectrum analyzers available through the Android Marketplace but there are none specifically for the monitoring and display of VLF electromagnetic waves. The APRAD application also allows the user to save samples of data to the removable storage card for offsite analysis which is a feature lacking in the current Android Marketplace product offering. There are also not any apps with the additional feature of being able to generate artificial waves for student observation. Although the primary use case for APRAD is field work, APRAD can also be re-purposed to identify other type of VLF interference such as electromagnetic waves generated by florescent lights.

Initially the software will be distributed via physical apk however once the application is complete, the option will be available to distribute through one of the Android marketplaces.

## Marketing Strategy

In order to be successful, the APRAD system must differentiate itself from competitors in order to appeal to customers in the online marketplace. To do this, APRAD will advertise the portability and convenience of the system. Current systems require AC power, a large antenna, cumbersome cabling and a large receiving unit. The APRAD system relies on the power of the portable device’s battery and a hand held antenna. The portable device also functions as the display so there is no need to carry additional equipment.

## Schedule

The APRAD project is expected to take eight months from project approval to launch of the APRAD system. The following is a high level schedule of some significant milestones for this initiative:

September 3, 2013: Initiate Project  
September 12, 2013: Project kickoff meeting  
December 12, 2013: Deliver application prototype and supporting documentation  
January 30, 2014: Finish Spectrum Analyzer and Test  
February 28, 2014: Finish Spectrogram and Test  
March 30, 2014: Finish Wave Visualizer and Test  
April 30, 2014: Finish Navigation Interface and Test  
May 16, 2014: Deliver final project and all project documentation

Upon approval of this project a detailed schedule will be created by the assigned project team to include all tasks and deliverables.

## Findings and Recommendations

Based on the information presented in this feasibility study, it is recommended that APRAD project moves forward and begins project initiation. The findings of this feasibility study show that this initiative will be highly beneficial to the Electrical Engineering department and has a high probability of success. Key findings are as follows:

Technology:

* Will utilize existing technology which lowers project risk
* Once in place this technology is simple to operate and maintain for a relatively low cost

Marketing:

* Since the APRAD system is unique, it will be a desirable system to implement
* The APRAD system helps to eliminate the barrier to entry for analysis of VLF electromagnetic waves

# Requirements

## Purpose

The purpose of this document is to describe the requirements for the Autonomous Portable low frequency Receiver and Display (APRAD) mobile application. The ARPAD device and associated mobile application is intended to make recordings of very low frequency (VLF: 300 Hz – 30 KHz) electromagnetic waves for research and educational purposes. APRAD leverages the fact that electromagnetic waves in this band have the same frequencies as audio signals and therefore mobile phones have the necessary hardware for digitization and recording. Currently, making science grade observations of very low frequency (VLF) electromagnetic waves involves a large antenna, up to 300 feet of cabling and a large receiver which requires AC power to make recordings of the waves. In these stationary setups interference from the electrical power grid can compromise the fidelity of the measurements and therefore selection of the location for antenna placement is of vital importance. If the antenna is placed in a spot which is not ideal, the operator must then return from the monitoring station to the antenna to make adjustments and in certain cases redeploy all hardware in a new location. For these situations the ARPAD will provide a portable survey tool that will provide real-time diagnosis of the electromagnetic environment in the VLF band. Upon successful completion, there will be a hand held antenna that will be able to interface with a mobile device to display frequency-time information of observed waves. This will eliminate the need to haul a large amount of equipment and significantly reduce the time it takes to deploy stationary recording hardware for scientific purposes. The APRAD project consists of two initiatives running in parallel. The EE department is designing the antenna and “black box” hardware that will receive and amplify the wave signal. The CSCI department is designing the mobile application that will interface with the “black box” via the phone 1/8” headphone/microphone jack. Additionally it has been requested that a visual educational tool be created to demonstrate wave movement in a 3-D graphical environment. This will allow ARPAD to be also used in the classroom as an educational tool in the classroom. The ARPAD mobile app will thus have two distinct functions: 1) record and display waves in real time picked up by the mobile antenna and 2) provide animations for visualizing key concepts of electromagnetic theory.

## Platform Requirements

1. The mobile application will run on the Android platform
2. The Samsung Galaxy SII will host the optimized version for the application
   1. The application can be optimized to run on other devices with some interface/functional tweaking. This may be minimal.
   2. The Galaxy SII is supports Android versions 2.3 (Gingerbread, API 9 and 10), 4.0 (Ice Cream Sandwich, API 14 and 15), and 4.1.2 (Jelly Bean, API 17). While these are not the latest Android OS versions (4.4, Kit Kat, API 19 as of May 16, 2014), they should be sufficient to support the needs of this project.
3. The headphone jack will be used to receive an analog signal
   1. The device where the application is installed must have the conventional three ring 1/8” headphone jack. The analog signal will be sent from the “black box” and will have a 1:1 relationship with the desired wave pattern to be monitored
   2. During “black box” development, the phone’s microphone input will be used to receive a wave form

## General Application requirements

1. This app will have two ‘parts’: the Spectrum Analyzer feature and the Spectrogram feature.
2. Both features of the app will be designed for use with the APRAD equipment.
3. The user will select which feature they would like to use as they run independently
   1. The user will be able to switch between features of the app as desired.
   2. Only on feature will be active at a time.
   3. There will be the ability to capture and save data for use in other applications and environments, such as educational settings or for data analysis in MATLAB.

## Wave OBSERVER feature

1. The primary users of this portion of the application are the members of the APRAD teams and other researchers that deploy antennas for scientific observations in the VLF band.
2. The application will accept an analog signal from the 1/8” microphone/headphone jack

2.1 The app will also read audible-frequency-range data from the device’s built in microphone

2.2 The built in microphone will act as a troubleshooting assistant if the app does not appear to be operating correctly with the cable plugged into the jack.

1. The application will have an option to display the frequency spectrum of the signal recorded over a specified time duration
   1. The “Y” axis will display the wave Amplitude in Decibels Relative to Full Scale (dBFS)
   2. The “X” axis will display the Frequency in Hertz (Hz)
      1. Nice to have: Pinch to zoom
      2. Nice to have: Panning
2. The application will have an option to display the signal in a spectrogram (dynamic frequency spectrum)
   1. The “Y” axis will display the wave Frequency in Hz
   2. The “X” axis will display the Time (seconds)
      1. Nice to have: Pinch to zoom
      2. Nice to have: Panning
   3. The Scale (dBFS) will be indicated as a color between blue and red
      1. Blue will indicate the lowest dBFS level
      2. Red will indicate the highest dBFS level
3. Fast Fourier Transformations (FFT) will be used to convert time to frequency and vice versa
4. The application will be able to export sixty seconds of data to a csv formatted file
   1. The file will include a date/time stamp in the header
   2. The file will include the sampling frequency in the header
   3. Nice to have: The file will include a gps stamp in the header
   4. The file will be formed as a 1 dimensional array
5. The sample rate will be adjustable
   1. Default will be 8000 Hz
   2. Max will be 48000Hz
   3. There will be a list of selectable sampling frequencies, not a text entry field. This will eliminate sampling frequency values that are incompatible with the software.
6. There will be an option in the Spectrogram to turn off the display and focus on capturing the input signal.
   1. Default will be ‘Live Data Rendering’ where the data is displayed as the signal is processed.
   2. The option to turn off the immediate display will be an option in the ‘Settings’ portion of the software.
7. The scaling of the data displayed will be adjustable
   1. Default will be Logarithmic in order to better display low amplitude signals.
   2. Option to change to linear scaling if that is desired.

## Future Educational feature

1. There will be a portion of the app, left undeveloped, reserved for an ‘Educational’ feature.
2. The education feature on the app will allow for use of the captured data from the Spectrogram in an academic setting.
3. This feature is planned to have the ability to synthesize wave data to give pure signals.
4. The wave data will be adjustable in the areas of frequency, amplitude and phase shift.
5. Multiple waves will be possible and the interaction between the two will be displayed. This will enable Physics and Electrical Engineering students to study resultant waves.
6. Development of this portion of the app could take place either as a future Senior Design project or as a separately funded part of the APRAD grant.

# Use Case Diagram

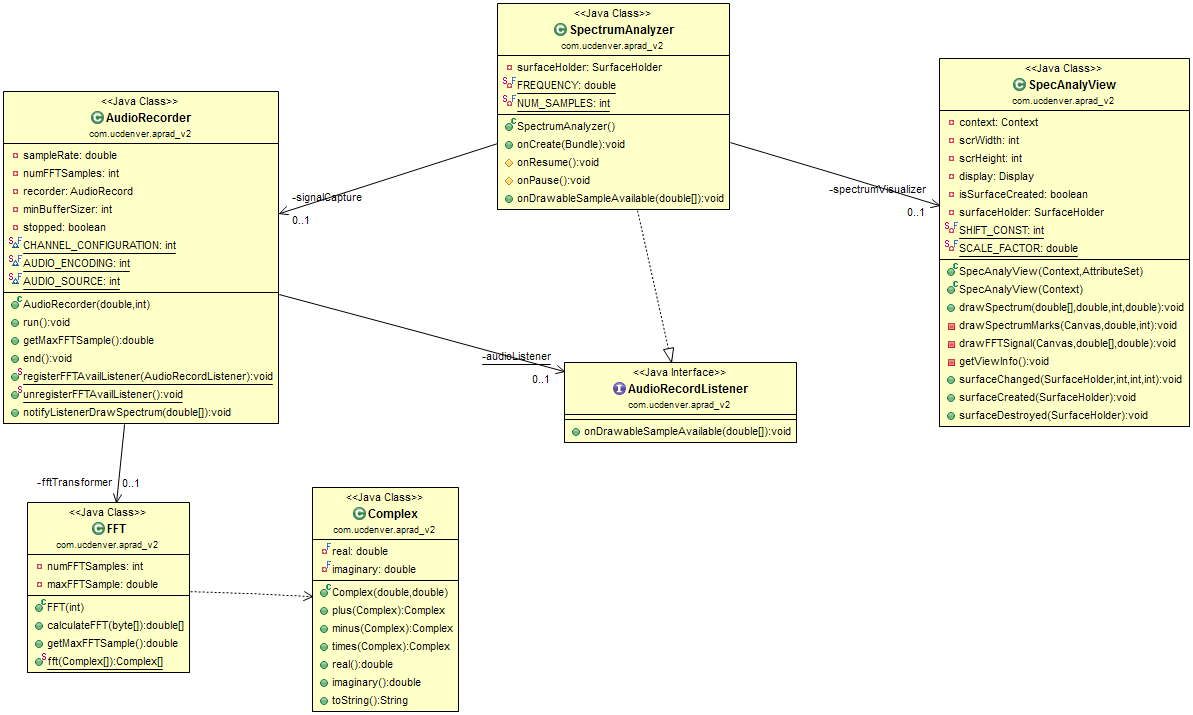


# Data Flow Diagram

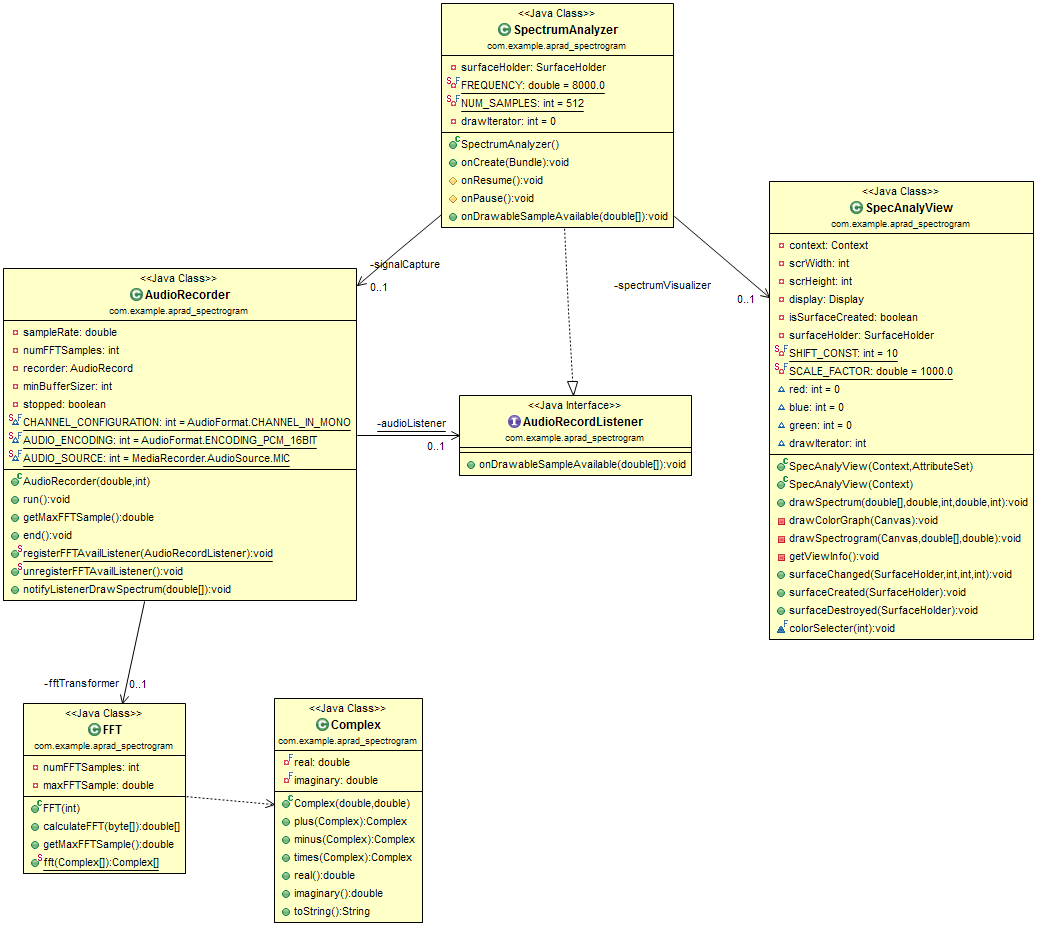


# Class Diagrams

## Spectrum Analyzer



## Spectrogram



# Source Code

## SpectrumAnalyzer.java

**package edu.ucdenver.aprad.spectrum\_analyzer;**

**import android.app.Activity;**

**import android.content.Intent;**

**import android.content.SharedPreferences;**

**import android.os.Bundle;**

**import android.util.Log;**

**import android.view.Menu;**

**import android.view.MenuInflater;**

**import android.view.MenuItem;**

**import android.view.SurfaceHolder;**

**import edu.ucdenver.aprad.Preferences;**

**import edu.ucdenver.aprad.R;**

**import edu.ucdenver.aprad.spectrogram.Spectrogram;**

**import edu.ucdenver.aprad.tools.AudioRecordListener;**

**import edu.ucdenver.aprad.tools.AudioRecorder;**

**import edu.ucdenver.aprad.tools.FFT;**

**/\*\***

**\* SpectrumAnalyzer main class which extends Activity**

**\* @author Dan Rolls**

**\* @author Michael Dewar**

**\*/**

**public class SpectrumAnalyzer extends Activity implements AudioRecordListener**

**{**

**private SpectrumAnalyzerView spectrumVisualizer;**

**private SurfaceHolder surfaceHolder;**

**private AudioRecorder signalCapture;**

**private FFT fft;**

**private static double frequency = 8000.0;**

**private static final int NUM\_SAMPLES = 512;**

**/\*\***

**\* Called when the activity is first created.**

**\*/**

**@Override**

**public void onCreate(Bundle savedInstanceState)**

**{**

**super.onCreate(savedInstanceState);**

**setContentView(R.layout.spectrum\_analyzer);**

**// get the Surface view which is used to draw the spectrum.**

**this.spectrumVisualizer = (SpectrumAnalyzerView) findViewById(R.id.surfaceView);**

**this.surfaceHolder = this.spectrumVisualizer.getHolder();**

**fft = new FFT( NUM\_SAMPLES );**

**}**

**/\*\***

**\* Initialize AudioRecorder and register the AudioRecorder listener**

**\*/**

**@Override**

**protected void onResume()**

**{**

**super.onResume();**

**updatePreferences();**

**this.signalCapture = new AudioRecorder( frequency, NUM\_SAMPLES );**

**AudioRecorder.registerFFTAvailListener(this);**

**}**

**private void updatePreferences(){**

**SharedPreferences sharedPreferences = getSharedPreferences( Preferences.PREFS\_NAME, 0 );**

**frequency = sharedPreferences.getFloat( Preferences.FREQUENCY, 8000.0f );**

**}**

**/\*\***

**\* End the signal capture and unregister listener**

**\*/**

**@Override**

**protected void onPause()**

**{**

**super.onPause();**

**this.signalCapture.end();**

**AudioRecorder.unregisterFFTAvailListener();**

**}**

**@Override**

**public boolean onCreateOptionsMenu(Menu menu)**

**{**

**MenuInflater inflater = getMenuInflater();**

**inflater.inflate(R.menu.aprad, menu);**

**return true;**

**}**

**@Override**

**public boolean onOptionsItemSelected(MenuItem item) {**

**// Handle item selection**

**switch (item.getItemId()) {**

**case R.id.settings:**

**Intent intent = new Intent( SpectrumAnalyzer.this, Preferences.class);**

**intent.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK | Intent.FLAG\_ACTIVITY\_CLEAR\_TASK);**

**this.startActivity(intent);**

**return true;**

**default:**

**return super.onOptionsItemSelected(item);**

**}**

**}**

**/\*\***

**\* Implements listener.**

**\* Creates a new Runnable but runs it on the UI thread**

**\* @param signal Array of FFT Signals**

**\*/**

**@Override**

**public void onDrawableSampleAvailable( short[] signal)**

**{**

**double[] transformedSignal = fft.calculateFFT( signal );**

**spectrumVisualizer.drawSpectrum(**

**transformedSignal,**

**frequency,**

**NUM\_SAMPLES,**

**signalCapture.getMaxFFTSample() );**

**}**

**}**

## SpectrumAnalyzerView.java

package edu.ucdenver.aprad.spectrum\_analyzer;

import android.content.Context;

import android.content.res.Resources;

import android.graphics.Canvas;

import android.graphics.Color;

import android.graphics.Paint;

import android.util.AttributeSet;

import android.util.Log;

import android.util.TypedValue;

import android.view.Display;

import android.view.SurfaceHolder;

import android.view.SurfaceView;

import android.view.WindowManager;

/\*\*

\* Class SpecAnalyView

\* Handles drawing the results to the UI

\* @author Dan Rolls

\* @author Michael Dewar

\*

\*/

public class SpectrumAnalyzerView extends SurfaceView implements SurfaceHolder.Callback

{

private Context context;

private int scrWidth;

private int scrHeight;

private int scaleWidth;

private int scaleHeight;

private Display display;

private boolean isSurfaceCreated;

private SurfaceHolder surfaceHolder;

private Canvas canvas;

Paint paint;

Paint legendPaint;

private static final int COLOR\_RANGE = 256;

int red = 0;

int blue = 0;

int green = 0;

private Resources resources;

private static final int SHIFT\_CONST = 50;

private static final double SCALE\_FACTOR = 1.0;

private static final int SIGNAL\_OFFSET\_X\_AXIS = 100;

private static final int SIGNAL\_OFFSET\_Y\_AXIS = 40;

private static final int SIGNAL\_PADDING\_X\_AXIS = 45;

private static final int SIGNAL\_PADDING\_Y\_AXIS = 30;

public static final double MAX\_FREQUENCY = 48000.00;

private double frequency = MAX\_FREQUENCY;

public static final int SIGNAL\_LENGTH = 256;

private static final float X\_AXIS\_LEFT\_PADDING = 20.0f;

private static final float SHORT\_GRID\_WIDTH = 4;

private static final float Y\_AXIS\_SPACING = 0;

private final int Y\_AXIS\_ROW\_COUNT = 10;

private static final float Y\_AXIS\_BOTTOM\_PADDING = 20.0f;

private static final float GRID\_TEXT\_SIZE = 8;

private static final int SPECTRUM\_HEIGHT = 480;

/\*\*

\* Constructor for SpecAnalyView

\* @param context Context of the application

\* @param attrs Attribute set of the application

\*/

public SpectrumAnalyzerView(Context context, AttributeSet attrs)

{

super(context, attrs);

init( context );

}

/\*\*

\* Constructor for SpecAnalyView passing in only the context

\* @param context Context of the application

\*/

public SpectrumAnalyzerView(Context context)

{

super(context);

init( context );

}

private void init( Context context ){

getHolder().addCallback(this);

legendPaint = new Paint();

surfaceHolder = getHolder();

setFocusable(true);

this.context = context;

this.display = ((WindowManager) context.getSystemService(Context.WINDOW\_SERVICE)).getDefaultDisplay();

paint = new Paint();

resources = getResources();

}

/\*\*

\* Main class to handle the drawing of the UI

\* @param signal Array of processed FFT bytes

\* @param samplingRate The rate at which the samples were harvested

\* @param numberOfFFTPoints The number of FFT points analyzed

\* @param maxFFTSample The max size of the FFT sample

\*/

public void drawSpectrum( double[] signal,

double samplingRate,

int numberOfFFTPoints,

double maxFFTSample ){

if(isSurfaceCreated)

{

loadScaleValues();

for( int i=0; i<4; ++ i ){

canvas = surfaceHolder.lockCanvas(null);

if( canvas != null ){

canvas.drawColor( Color.BLACK );

drawAxisX( canvas, samplingRate );

drawAxisY( canvas, signal );

drawSignalGraph( canvas, signal, maxFFTSample );

}

surfaceHolder.unlockCanvasAndPost( canvas );

}

}

}

/\*\*

\* Draws y-axis lines to provide clear reference to dB values for amplitude of signal

\*/

private void drawAxisY( Canvas canvas, double[] signal ){

paint.setColor(Color.WHITE);

paint.setTextSize( getGridRelativeSize() );

int yInterval = (SPECTRUM\_HEIGHT\*scaleHeight)/(Y\_AXIS\_ROW\_COUNT);

drawAxisYdBKey( canvas, signal );

for( int i = 0; i < Y\_AXIS\_ROW\_COUNT; ++ i ){

int y = i \* yInterval + (int)getAxisYBottomPadding();

drawAxisYdBValue( canvas, i, y );

drawAxisYRowLine( canvas, i, y );

}

}

private void drawAxisYRowLine(Canvas canvas, int i, int y) {

if( i % 3 == 0 ){

drawRowLineWhole( canvas, y );

} else {

drawRowLineShort( canvas, y );

}

}

/\*\*

\* Class to draw the spectrum ticks and numbers for the scale

\* @param canvas The canvas to draw on

\* @param samplingRate The rate at which the samples were harvested

\* @param numberOfFFTPoints The number of FFT points analyzed

\*/

private void drawAxisX( Canvas canvas, double samplingRate )

{

paint.setColor(Color.WHITE);

paint.setTextSize( getGridRelativeSize() );

int maxFrequency = (int) samplingRate / 1000 / 2;

int xAxisIntervalSize = (SIGNAL\_LENGTH\*scaleWidth) / maxFrequency;

for(int i = 0; i <= maxFrequency; ++ i )

{

drawAxisXFrequencyValue( canvas, xAxisIntervalSize, i );

drawAxisXColumnLine( xAxisIntervalSize, i );

}

}

private void drawAxisXFrequencyValue( Canvas canvas,

int xAxisIntervalSize,

int i ){

canvas.drawText(

String.valueOf(i),

xAxisIntervalSize\*i + getAxisXLeftPadding() ,

scrHeight,

paint );

}

private void drawAxisXColumnLine(int xAxisIntervalSize, int i) {

// TODO Auto-generated method stub

}

/\*\*

\* Class to draw the actual signal coming from the FFT class

\* @param canvas The canvas to draw on

\* @param signal Array of processed FFT bytes

\* @param maxFFTSample The max size of the FFT sample

\*/

private void drawSignalGraph( Canvas canvas,

double[] signal,

double maxFFTSample ){

int sampleValue, nextSampleValue;

paint.setColor(Color.YELLOW);

paint.setStrokeWidth(2);

int xAxisPadding = (int) getAxisXLeftPadding();

int yAxisPadding = (int) getAxisYBottomPadding();

for(int i = 0; i < SIGNAL\_LENGTH - 1; i++)

{

sampleValue = (int) signal[i];

nextSampleValue = (int) signal[i+1];

drawSignalGraphInterval(

canvas,

sampleValue,

nextSampleValue,

xAxisPadding,

yAxisPadding,

i );

}

}

private void drawSignalGraphInterval( Canvas canvas,

int sampleValue,

int nextSampleValue,

int xAxisPadding,

int yAxisPadding,

int i ){

canvas.drawLine(

i\*scaleWidth+xAxisPadding,

(scrHeight - sampleValue\*scaleHeight) - yAxisPadding,

i\*scaleWidth+scaleWidth+xAxisPadding,

(scrHeight - nextSampleValue\*scaleHeight) - yAxisPadding,

paint );

}

private void drawAxisYdBKey(Canvas canvas, double[] signal) {

canvas.drawText( "dB", 0, getAxisYBottomPadding()/2, paint );

}

private void drawAxisYdBValue( Canvas canvas, int i, int y ){

String dBValue = "-" + (9-i)\*10;

canvas.drawText( dBValue, 10, scrHeight - y, paint);

}

private void drawRowLineWhole(Canvas canvas, int y) {

int xOffset = (int) getAxisXLeftPadding();

canvas.drawLine( xOffset, scrHeight - y, xOffset + SIGNAL\_LENGTH\*scaleWidth, scrHeight - y, paint);

}

private void drawRowLineShort(Canvas canvas, int y) {

int xOffset = (int) getAxisXLeftPadding();

canvas.drawLine( xOffset, scrHeight - y, xOffset + getShortGridWidth(), scrHeight - y, paint);

}

/\*\*

\* Get's the screen dimension and populates the

\* class screen attributes

\*/

private void getViewInfo()

{

this.scrWidth = getWidth();

this.scrHeight = getHeight();

Log.i("GetView: ","Width: "+getWidth()+" - Height: "+getHeight());

}

/\*\*

\* Override of the surfaceChanged class

\* @param holder The surface holder

\* @param format Screen format

\* @param width Screen width

\* @param height screen height

\*/

@Override

public void surfaceChanged(SurfaceHolder holder, int format, int width,int height)

{

this.scrWidth = width;

this.scrHeight = height;

}

/\*\*

\* Override of the surfaceCreated class

\* @param holder The surface holder

\*/

@Override

public void surfaceCreated(SurfaceHolder holder)

{

isSurfaceCreated = true;

getViewInfo();

}

/\*\*

\* Override of the surfaceDestroyed class

\* @param holder the surface holder

\*/

@Override

public void surfaceDestroyed(SurfaceHolder holder)

{

isSurfaceCreated = false;

}

/\*\*

\* Loads the height and width scaling values for the current display if they are unset.

\* If less than 0, sets to 1

\*/

public void loadScaleValues() {

if( scaleHeight == 0 || scaleWidth == 0 ){

scaleHeight = getHeight()/SPECTRUM\_HEIGHT;

scaleWidth = (getWidth() - SIGNAL\_OFFSET\_X\_AXIS)/SIGNAL\_LENGTH;

if( scaleHeight < 1 ){

scaleHeight = 1;

}

if( scaleWidth < 1 ){

scaleWidth = 1;

}

}

}

public float getAxisXLeftPadding(){

return TypedValue.applyDimension(

TypedValue.COMPLEX\_UNIT\_DIP,

X\_AXIS\_LEFT\_PADDING,

resources.getDisplayMetrics() );

}

public float getAxisYBottomPadding(){

return TypedValue.applyDimension(

TypedValue.COMPLEX\_UNIT\_DIP,

Y\_AXIS\_BOTTOM\_PADDING,

resources.getDisplayMetrics() );

}

public float getShortGridWidth(){

return TypedValue.applyDimension(

TypedValue.COMPLEX\_UNIT\_DIP,

SHORT\_GRID\_WIDTH,

resources.getDisplayMetrics() );

}

public float getAxisYSpacing(){

return TypedValue.applyDimension(

TypedValue.COMPLEX\_UNIT\_DIP,

Y\_AXIS\_SPACING,

resources.getDisplayMetrics() );

}

private float getGridRelativeSize(){

return TypedValue.applyDimension(

TypedValue.COMPLEX\_UNIT\_DIP,

GRID\_TEXT\_SIZE,

resources.getDisplayMetrics() );

}

}

## AudioRecordListener.java

**package** edu.ucdenver.aprad.tools;

/\*\*

\* Interface for AudioRecord to let SpecAnalyView to start drawing

\* **@author** Dan Rolls

\* **@author** Michael Dewar

\*/

**public** **interface** AudioRecordListener {

**void** onDrawableSampleAvailable(**short**[] signal);

}

## AudioRecorder.java

package edu.ucdenver.aprad.tools;

import android.media.AudioFormat;

import android.media.AudioRecord;

import android.media.MediaRecorder;

import android.util.Log;

/\*\*

\* AudioRecorder

\* Extends Thread to run the AudioRecorder in it's own thread

\* @author Dan Rolls

\* @author Michael Dewar

\*/

public class AudioRecorder extends Thread

{

private AudioRecord recorder;

private int minBufferSizer;

private double sampleRate;

private int numFFTSamples;

private boolean stopped;

private static final int CHANNEL\_CONFIGURATION = AudioFormat.CHANNEL\_IN\_MONO;

private static final int AUDIO\_ENCODING = AudioFormat.ENCODING\_PCM\_16BIT;

private static final int AUDIO\_SOURCE = MediaRecorder.AudioSource.MIC;

private static AudioRecordListener audioListener;

/\*\*

\* Constructor for AudioRecorder

\* @param sampleRate Frequency of samples in Hz

\* @param points The number of FFT samples to take

\*/

public AudioRecorder(double sampleRate, int points)

{

this.sampleRate = sampleRate;

this.numFFTSamples = points;

start();

}

/\*\*

\* To use Thread, run must be implemented

\* Starts the AudioRecorder and collects data for transformation

\*/

@Override

public void run()

{

this.minBufferSizer = AudioRecord.getMinBufferSize(

(int) this.sampleRate,

CHANNEL\_CONFIGURATION,

AUDIO\_ENCODING );

this.recorder = new AudioRecord(

AUDIO\_SOURCE,

(int)this.sampleRate,

CHANNEL\_CONFIGURATION,

AUDIO\_ENCODING,

this.minBufferSizer );

if( this.recorder == null ){

return;

}

try {

this.recorder.startRecording();

} catch( IllegalStateException e ){

Log.e( "RecordingFailed", e.toString() );

}

short[] signalBuffer = null;

int bufferReadResult = 0;

while( !stopped ){

signalBuffer = new short[numFFTSamples];

bufferReadResult = this.recorder.read( signalBuffer, 0, numFFTSamples );

// Check to make sure we read something

if(bufferReadResult > 0)

{

notifyListenerSignalRead(signalBuffer);

} else {

Log.e( "AudioRecorder",

"There was an error reading the audio device - ERROR: "

+ bufferReadResult );

}

}

recorder.stop();

recorder.release();

}

/\*\*

\* Used to get Max FFT sample value from FFT Class

\* @return Max FFT Sample from the FFT Processing

\*/

public double getMaxFFTSample(){

return numFFTSamples;

}

/\*\*

\* Ends the thread

\*/

public void end(){

stopped = true;

}

/\*\*

\* register the audiolistener to the class listner

\* @param listener AudioRecord listener

\*/

public static void registerFFTAvailListener( AudioRecordListener listener )

{

audioListener = listener;

}

/\*\*

\* Unregisters listener. Sets listener to null

\*/

public static void unregisterFFTAvailListener()

{

audioListener = null;

}

/\*\*

\* Lets SpectrumAnalyzer know to go ahead and draw

\* @param signal The raw data converted into an FFT

\*/

public void notifyListenerSignalRead( short[] signal )

{

if( audioListener != null ){

audioListener.onDrawableSampleAvailable( signal );

}

}

}

## Complex.java

**package** edu.ucdenver.aprad.tools;

/\*\*

\* Complex Object

\* <P>Various attributes and methods related to calculation of Complex numbers.

\*

\* **@author** Dan Rolls

\* **@author** Michael Dewar

\* **@version** 1.0

\*/

**public** **class** Complex

{

**private** **final** **double** real;

**private** **final** **double** imaginary;

/\*\*

\* Constructor for Complex class

\* **@param** real The real part of the complex number

\* **@param** imag The imaginary part of the complex number

\*/

**public** Complex(**double** real, **double** imag)

{

**this**.real = real;

**this**.imaginary = imag;

}

/\*\*

\* Addition of two complex numbers

\* **@param** b The Complex number to be evaluated

\* **@return** this Complex plus b Complex

\*/

**public** Complex plus(Complex b)

{

Complex a = **this**;

**double** real = a.real + b.real;

**double** imag = a.imaginary + b.imaginary;

**return** **new** Complex(real, imag);

}

/\*\*

\* Subtraction of two complex numbers

\* **@param** b The complex number to be evaluated

\* **@return** this Complex minux b Complex

\*/

**public** Complex minus(Complex b)

{

Complex a = **this**;

**double** real = a.real - b.real;

**double** imag = a.imaginary - b.imaginary;

**return** **new** Complex(real, imag);

}

/\*\*

\* Multiplication of two complex numbers

\* **@param** b The complex to be evaluated

\* **@return** this Complex times b Complex

\*/

**public** Complex times(Complex b)

{

Complex a = **this**;

**double** real = a.real \* b.real - a.imaginary \* b.imaginary;

**double** imag = a.real \* b.imaginary + a.imaginary \* b.real;

**return** **new** Complex(real, imag);

}

/\*\*

\* Returns real part

\* **@return** real part of complex number

\*/

**public** **double** real()

{

**return** **this**.real;

}

/\*\*

\* Returns imaginary part

\* **@return** imaginary part of complex number

\*/

**public** **double** imaginary()

{

**return** **this**.imaginary;

}

/\*\*

\* toString class returner

\* **@return** String representation of Complex class

\*/

**public** String toString()

{

**if** (**this**.imaginary == 0)

**return** **this**.real + "";

**if** (**this**.real == 0)

**return** **this**.imaginary + "i";

**if** (**this**.imaginary < 0)

**return** **this**.real + " - " + (-**this**.imaginary) + "i";

**return** **this**.real + " + " + **this**.imaginary + "i";

}

}

## FFT.java

**package** edu.ucdenver.aprad.tools;

/\*\*

\* Class FFT

\* Used to do a Cooley-Tukey FFT type

\* **@link** http://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey\_FFT\_algorithm

\* **@author** Dan Rolls

\* **@author** Michael Dewar

\*/

**public** **class** FFT

{

**private** **int** numFFTSamples;

**private** **double** maxFFTSample;

/\*\*

\* Constructor

\* **@param** points The number of FFT Samples

\*/

**public** FFT(**int** points)

{

**this**.numFFTSamples = points;

}

/\*\*

\* Start doing calculations for FFT

\* **@param** signal Raw byte signal from AudioRecorder

\* **@return** double[] of FFT results

\*/

**public** **double**[] calculateFFT(**short**[] signal)

{

**double** temp;

Complex[] y;

Complex[] complexSignal = **new** Complex[numFFTSamples];

**double**[] signalProcess = **new** **double**[numFFTSamples/2];

**for**(**int** i = 0; i < numFFTSamples; i++){

temp = (**double**)signal[i] / 32768.0F;

complexSignal[i] = **new** Complex(temp,0.0);

}

y = *fft*(complexSignal);

maxFFTSample = 0.0;

**for**(**int** i = 0; i < (numFFTSamples/2); i++)

{

signalProcess[i] = Math.*sqrt*(Math.*pow*(y[i].real(), 2) + Math.*pow*(y[i].imaginary(), 2));

**if**(signalProcess[i] > maxFFTSample)

{

maxFFTSample = signalProcess[i];

}

}

**return** signalProcess;

}

/\*\*

\* Retrieves the maxFFT Sample

\* **@return**

\*/

**public** **double** getMaxFFTSample(){

**return** maxFFTSample;

}

/\*\*

\* Provides the finalized FFT result

\* **@param** complex Complex number

\* **@return** an array of complex numbers

\*/

**public** **static** Complex[] fft(Complex[] complex)

{

**int** N = complex.length;

// base case

**if** (N == 1) **return** **new** Complex[]

{

complex[0]

};

// In order for FFT to work, it needs to be a power of 2

**if** (N % 2 != 0)

{

**throw** **new** RuntimeException("N is not a power of 2");

}

// fft of even terms

Complex[] even = **new** Complex[N/2];

**for** (**int** i = 0; i < N/2; i++)

{

even[i] = complex[2\*i];

}

Complex[] q = *fft*(even);

// fft of odd terms

Complex[] odd = even;

**for** (**int** i = 0; i < N/2; i++) {

odd[i] = complex[2\*i + 1];

}

Complex[] r = *fft*(odd);

// combine even and odd results

Complex[] y = **new** Complex[N];

**for** (**int** i = 0; i < N/2; i++)

{

**double** k = -2 \* i \* Math.*PI* / N;

Complex wk = **new** Complex(Math.*cos*(k), Math.*sin*(k));

y[i] = q[i].plus(wk.times(r[i]));

y[i + N/2] = q[i].minus(wk.times(r[i]));

}

**return** y;

}

}

## Spectrogram.java

**package edu.ucdenver.aprad.spectrogram;**

**import java.io.File;**

**import java.io.FileInputStream;**

**import java.io.FileNotFoundException;**

**import java.io.FileOutputStream;**

**import java.io.IOException;**

**import edu.ucdenver.aprad.OpenSavedData;**

**import edu.ucdenver.aprad.Preferences;**

**import edu.ucdenver.aprad.R;**

**import edu.ucdenver.aprad.education.Education;**

**import edu.ucdenver.aprad.tools.AudioRecordListener;**

**import edu.ucdenver.aprad.tools.AudioRecorder;**

**import edu.ucdenver.aprad.tools.FFT;**

**import android.app.Activity;**

**import android.app.DialogFragment;**

**import android.content.Context;**

**import android.content.Intent;**

**import android.content.SharedPreferences;**

**import android.content.res.Resources;**

**import android.net.Uri;**

**import android.os.Bundle;**

**import android.util.TypedValue;**

**import android.view.Menu;**

**import android.view.MenuInflater;**

**import android.view.MenuItem;**

**import android.view.MotionEvent;**

**import android.view.SurfaceHolder;**

**import android.widget.Toast;**

**/\*\***

**\* The Spectrogram is used to map frequency and amplitude over time. This is**

**\* represented as frequency on the y axis, time on the x axis, amd amplitude**

**\* is the color of the point on the graph.**

**\***

**\* @author Dan Rolls**

**\* @author Michael Dewar**

**\* @author Robbie Perlstein**

**\*/**

**public class Spectrogram extends Activity implements AudioRecordListener**

**{**

**private SpectrogramView spectrogramView;**

**private AudioRecorder signalCapture;**

**private SpectrogramFileIO fileWriter;**

**private FFT fft;**

**private boolean stopped = false;**

**private double frequency = 8000.0;**

**private int buffer\_size = 512;**

**private boolean liveRender = false;**

**private boolean scaling = true;**

**private final int START\_STOP\_BUTTON\_X\_SCALAR = 15;**

**private final int START\_STOP\_BUTTON\_Y\_SCALAR = 3;**

**private short[][] rawSignals;**

**private int signal\_count;**

**private boolean rendering = false;**

**private long start\_time;**

**private long end\_time;**

**private boolean buttonPress = false;**

**@Override**

**public void onCreate(Bundle savedInstanceState)**

**{**

**super.onCreate( savedInstanceState );**

**setContentView( R.layout.spectrogram );**

**this.spectrogramView = (SpectrogramView) findViewById( R.id.surfaceView );**

**fileWriter = new SpectrogramFileIO();**

**fft = new FFT( buffer\_size );**

**}**

**/\*\***

**\* Initialize AudioRecorder and register the AudioRecorder listener**

**\*/**

**@Override**

**protected void onResume()**

**{**

**super.onResume();**

**reloadVisualizer();**

**updatePreferences();**

**this.spectrogramView.setFrequency( frequency );**

**this.spectrogramView.setLogScaling( scaling );**

**beginDataCollection();**

**}**

**@Override**

**public boolean onCreateOptionsMenu(Menu menu)**

**{**

**MenuInflater inflater = getMenuInflater();**

**inflater.inflate(R.menu.aprad, menu);**

**return true;**

**}**

**@Override**

**public boolean onOptionsItemSelected( MenuItem item ){**

**// Handle item selection**

**switch ( item.getItemId() ){**

**case R.id.settings:**

**Intent settingsIntent = new Intent( Spectrogram.this, Preferences.class );**

**settingsIntent.setFlags( Intent.FLAG\_ACTIVITY\_NEW\_TASK**

**| Intent.FLAG\_ACTIVITY\_CLEAR\_TASK );**

**this.startActivity(settingsIntent);**

**return true;**

**case R.id.open\_file:**

**Intent openSavedDataIntent = new Intent( Spectrogram.this,**

**OpenSavedData.class );**

**this.startActivity(openSavedDataIntent);**

**return true;**

**default:**

**return super.onOptionsItemSelected(item);**

**}**

**}**

**/\*\***

**\* End the signal capture and unregister listener**

**\*/**

**@Override**

**protected void onPause()**

**{**

**super.onPause();**

**stopped = true;**

**stopDataCollection();**

**}**

**@Override**

**public boolean onTouchEvent(MotionEvent event)**

**{**

**if( !buttonPress ){**

**buttonPress = true;**

**if( !rendering ){**

**if( event.getAction() == MotionEvent.ACTION\_DOWN ){**

**float relativeTextSize = spectrogramView.getRelativeTopBarSize();**

**if( event.getX() <= relativeTextSize \* START\_STOP\_BUTTON\_X\_SCALAR**

**&& event.getY() <= relativeTextSize \* START\_STOP\_BUTTON\_Y\_SCALAR ){**

**toggleRecord();**

**}**

**}**

**}**

**buttonPress = false;**

**}**

**return true;**

**}**

**private void reloadVisualizer(){**

**}**

**private void updatePreferences() {**

**SharedPreferences sharedPreferences = getSharedPreferences(Preferences.PREFS\_NAME, 0);**

**liveRender = sharedPreferences.getBoolean( Preferences.LIVE\_RENDER, true );**

**frequency = sharedPreferences.getFloat( Preferences.FREQUENCY, 8000.0f );**

**scaling = sharedPreferences.getBoolean( Preferences.LOG\_SCALING, true );**

**}**

**private void toggleRecord() {**

**if( stopped ){**

**beginDataCollection();**

**} else {**

**stopDataCollection();**

**renderCollectedData();**

**}**

**}**

**public void beginDataCollection(){**

**stopped = false;**

**this.signalCapture = new AudioRecorder(frequency,buffer\_size);**

**AudioRecorder.registerFFTAvailListener(this);**

**rawSignals = new short[SpectrogramView.MAX\_SIGNALS\_LENGTH][];**

**signal\_count = 0;**

**start\_time = System.currentTimeMillis();**

**}**

**public void stopDataCollection(){**

**end\_time = System.currentTimeMillis();**

**this.signalCapture.end();**

**stopped = true;**

**AudioRecorder.unregisterFFTAvailListener();**

**}**

**/\*\***

**\* Executes the FFT on the signal, and displays any unrendered signals.**

**\* <br>**

**\* Displays rendering indicator, then updates with start button.**

**\*/**

**public void renderCollectedData(){**

**rendering = true;**

**if( !liveRender ){**

**spectrogramView.undrawRecordingIndicator();**

**spectrogramView.drawButtonRendering();**

**drawSpectrogram();**

**}**

**rendering = false;**

**}**

**// TODO**

**public void saveDataDialog(){**

**DialogFragment newFragment = new SaveDialog();**

**newFragment.show(getFragmentManager(), "Save?");**

**if( true ){**

**saveData();**

**}**

**}**

**// TODO**

**public void saveData(){**

**String filename = "myfile";**

**String string = "Hello world!";**

**FileOutputStream outputStream;**

**try {**

**outputStream = openFileOutput( filename, Context.MODE\_PRIVATE );**

**outputStream.write( string.getBytes() );**

**outputStream.close();**

**} catch ( Exception e ){**

**e.printStackTrace();**

**}**

**File file;**

**try {**

**//String fileName = Uri.parse(filename).getLastPathSegment();**

**FileInputStream fio = new FileInputStream( "myfile" );**

**fio.toString();**

**}**

**catch (IOException e) {**

**// Error while creating file**

**}**

**}**

**/\*\***

**\* Implements AudioRecorder listener. <br>**

**\* Draws a new sample on the graph.**

**\*/**

**@Override**

**public void onDrawableSampleAvailable( short[] signal )**

**{**

**loadSpectrogramGrid();**

**storeNextSignal( signal );**

**validateContinuation();**

**// Render signal live**

**if( liveRender ){**

**drawSpectrogramColumn( signal\_count-1 );**

**// Record then render signal**

**} else if( recordingComplete() ){**

**renderCollectedData();**

**}**

**if( stopped ){**

**spectrogramView.drawTopDuration( getElapsedTime() );**

**saveDataDialog();**

**spectrogramView.drawButtonStart();**

**}**

**}**

**private long getElapsedTime() {**

**return end\_time - start\_time;**

**}**

**/\*\***

**\* Tests to see if singal limit has been reached, and updates relevant fields.**

**\*/**

**private void validateContinuation(){**

**if( signal\_count >= SpectrogramView.MAX\_SIGNALS\_LENGTH ){**

**stopDataCollection();**

**}**

**}**

**/\*\***

**\***

**\* @param signal next raw signal from AudioRecord**

**\*/**

**private void storeNextSignal( short[] signal ){**

**rawSignals[signal\_count] = signal;**

**++ signal\_count;**

**}**

**private void loadSpectrogramGrid(){**

**if( signal\_count == 0 ){**

**spectrogramView.loadScaleValues();**

**spectrogramView.clearSprectrogramView();**

**spectrogramView.drawGrid();**

**spectrogramView.drawButtonStop();**

**spectrogramView.drawTopSampleRate();**

**if( !liveRender ){**

**spectrogramView.drawRecordingIndicator();**

**}**

**}**

**}**

**private void drawSpectrogramColumn( int offset ){**

**double[] signal = fft.calculateFFT( rawSignals[offset] );**

**spectrogramView.drawSpectrogram(**

**signal,**

**frequency,**

**buffer\_size,**

**signalCapture.getMaxFFTSample(),**

**offset );**

**}**

**/\*\***

**\* Draws the spectrogram from the signals currently saved.**

**\*/**

**private void drawSpectrogram(){**

**double[][] signals = calculateRawSignalsFFT();**

**spectrogramView.drawSpectrogramRange(**

**signals,**

**frequency,**

**buffer\_size,**

**signalCapture.getMaxFFTSample(),**

**signal\_count,**

**signal\_count**

**);**

**}**

**private double[][] calculateRawSignalsFFT(){**

**double[][] signals = new double[SpectrogramView.MAX\_SIGNALS\_LENGTH][];**

**for( int i = 0; i < signal\_count; ++ i ){**

**signals[i] = fft.calculateFFT( rawSignals[i] );**

**}**

**return signals;**

**}**

**private boolean recordingComplete(){**

**return (signal\_count == SpectrogramView.MAX\_SIGNALS\_LENGTH);**

**}**

**}**

## SpectrogramView.java

**package** com.example.aprad\_spectrogram;

**package edu.ucdenver.aprad.spectrogram;**

**import java.util.Vector;**

**import android.content.Context;**

**import android.content.res.Resources;**

**import android.graphics.Canvas;**

**import android.graphics.Color;**

**import android.graphics.Paint;**

**import android.util.AttributeSet;**

**import android.util.Log;**

**import android.util.TypedValue;**

**import android.view.Display;**

**import android.view.SurfaceHolder;**

**import android.view.SurfaceView;**

**import android.view.WindowManager;**

**/\*\***

**\* The Spectrogram is used to map frequency and amplitude over time. This is**

**\* represented as frequency on the y axis, time on the x axis, amd amplitude**

**\* is the color of the point on the graph.**

**\* <br><br>**

**\* SpectrogramView is used to draw the information passed to the Spectrogram.**

**\***

**\* @author Robbie Perlstein**

**\* @author Dan Rolls**

**\* @author Michael Dewar**

**\***

**\*/**

**public class SpectrogramView extends SurfaceView implements SurfaceHolder.Callback**

**{**

**private int scrWidth;**

**private int scrHeight;**

**private int scaleWidth;**

**private int scaleHeight;**

**private boolean isSurfaceCreated;**

**private SurfaceHolder surfaceHolder;**

**private Resources resources;**

**private static final int COLOR\_RANGE = 256;**

**private static final int TIME\_INTERVAL\_COUNT = 6;**

**public static final int MAX\_SIGNALS\_LENGTH = 720;**

**public static final int SIGNALS\_HEIGHT = 256;**

**public static final double MAX\_FREQUENCY = 48000.00;**

**public static final double MIN\_FREQUENCY = 8000.00;**

**public static final int COLOR\_GRAPH\_HEIGHT = 360;**

**private static final int COLOR\_GRAPH\_WIDTH = 5;**

**private static final int SIGNAL\_PADDING\_X\_AXIS = 25;**

**private static final int SIGNAL\_PADDING\_Y\_AXIS = 20;**

**private static final int GRID\_TEXT\_SIZE = 10;**

**private static final String START\_RECORDING\_STR = "Start Recording";**

**private static final String STOP\_RECORDING\_STR = "Stop Recording";**

**private static final String RENDERING\_RECORDING\_STR = "Rendering Data...";**

**private static final String RECORDING\_INDICATOR\_STR = "Recording in Background...";**

**public static final int TOP\_BAR\_TEXT\_SIZE = 19;**

**public static final int TOP\_PADDING = 5;**

**private double frequency = MIN\_FREQUENCY;**

**private static Paint spectroPaint;**

**private static Paint legendPaint;**

**private static Paint topBarPaint;**

**private static Paint gridPaint;**

**private Canvas canvas;**

**private int red = 0;**

**private int blue = 0;**

**private int green = 0;**

**private boolean logScaling = true;**

**// SINGLETON relative scaling values**

**private Float relativeColorGraphWidth;**

**private Float relativeGridTextSize;**

**private Float relativePaddingAxisX;**

**private Float relativePaddingAxisY;**

**private Float relativeTopBarSize;**

**//////////////////////////////////////////////////////////////////////////////**

**/\*\***

**\* View that draws the spectrogram to the canvas**

**\*/**

**public SpectrogramView( Context context, AttributeSet attrs )**

**{**

**super(context, attrs);**

**init(context);**

**}**

**/\*\***

**\* View that draws the spectrogram to the canvas**

**\*/**

**public SpectrogramView( Context context )**

**{**

**super(context);**

**init(context);**

**}**

**/\*\***

**\* Initialize the SpectrogramView, for use with constructors.**

**\* @param context**

**\*/**

**private void init( Context context ){**

**canvas = null;**

**getHolder().addCallback(this);**

**surfaceHolder = getHolder();**

**setFocusable(true);**

**scrHeight = getHeight();**

**scrWidth = getWidth();**

**resources = getResources();**

**spectroPaint = new Paint();**

**legendPaint = new Paint();**

**getPaintTopBar();**

**getPaintGrid();**

**}**

**/\*\***

**\* Updates the canvas to display the current signal, and if no amplitude**

**\* legend has been drawn, it will draw that on the canvas as well. Requires**

**\* four passes to properly update the canvas, which has a performance cost.**

**\* @return number of signals stored**

**\* @param signal double array of 256 length**

**\* @param samplingRate**

**\* @param numberOfFFTPoints**

**\* @param maxFFTSample**

**\* @param drawIterator**

**\*/**

**public int drawSpectrogram( double[] signal,**

**double samplingRate,**

**int numberOfFFTPoints,**

**double maxFFTSample,**

**int offset ){**

**if( offset == MAX\_SIGNALS\_LENGTH ){**

**return offset;**

**}**

**if( isSurfaceCreated ){**

**// TODO have draw return a bitmap, and set the canvas bitmap**

**for( int i = 0; i < 4; ++ i ){**

**canvas = surfaceHolder.lockCanvas();**

**drawSpectrogramColumn( canvas, signal, maxFFTSample, offset );**

**surfaceHolder.unlockCanvasAndPost( canvas );**

**}**

**}**

**if( offset == MAX\_SIGNALS\_LENGTH ){**

**//drawButtonStart();**

**}**

**return offset;**

**}**

**/\*\***

**\* Loads the height and width scaling values for the current display if they are unset.**

**\* If less than 0, sets to 1**

**\*/**

**public void loadScaleValues() {**

**if( scaleHeight == 0 || scaleWidth == 0 ){**

**scaleHeight = (int) ((getHeight() - getRelativePaddingAxisY())/SIGNALS\_HEIGHT);**

**scaleWidth = (int) ((getWidth() - getRelativePaddingAxisX())/MAX\_SIGNALS\_LENGTH);**

**if( scaleHeight < 1 ){**

**scaleHeight = 1;**

**}**

**if( scaleWidth < 1 ){**

**scaleWidth = 1;**

**}**

**}**

**}**

**// TODO remove literals**

**/\*\***

**\* Draws left-side Y-axis grid lines correlating to frequency in KHz**

**\* @param canvas Canvas to be drawn on**

**\*/**

**private void drawGridKHz( Canvas canvas ){**

**// Calculate spacing and interval**

**int effectiveFrequency = (int) frequency / 2;**

**int gridIntervals = effectiveFrequency / 2 / 1000;**

**gridIntervals = (gridIntervals < 3 ) ? 4 : gridIntervals;**

**int yAxisSpacing = ( SIGNALS\_HEIGHT \* scaleHeight ) / gridIntervals;**

**int frequencyValue = 0;**

**int textOffsetX = 0;**

**int yAxisOffset = 0;**

**// Draw grid interval marks**

**for( int i = 0; i < gridIntervals + 1; ++ i ){**

**frequencyValue = ( effectiveFrequency / 1000 / gridIntervals ) \* i;**

**textOffsetX = ( frequencyValue >= 10 ) ? 0 : 5;**

**yAxisOffset = scrHeight - (yAxisSpacing \* i);**

**drawKHzValues( canvas, frequencyValue, textOffsetX, yAxisOffset );**

**drawKHzMarkings( canvas, yAxisOffset );**

**}**

**// Draw Legend indicator**

**drawKHzKey( canvas );**

**}**

**/\*\***

**\* Draws bottom X-axis grid lines indicating time elapsed**

**\* @param canvas Canvas to be drawn on**

**\*/**

**private void drawGridTime( Canvas canvas ){**

**int xAxisSpacing = (MAX\_SIGNALS\_LENGTH \* scaleWidth)/TIME\_INTERVAL\_COUNT;**

**// Draw grid interval marks**

**for( int i = 0; i < TIME\_INTERVAL\_COUNT + 1; ++ i ){**

**int xAxisOffset = (int) (xAxisSpacing\*i + getRelativePaddingAxisX());**

**String intervalStr = String.valueOf( i\*10 );**

**canvas.drawLine( xAxisOffset,**

**scrHeight - getRelativeGridTextSize(),**

**xAxisOffset,**

**scrHeight - getRelativeGridTextSize()\*2,**

**gridPaint );**

**canvas.drawText( intervalStr,**

**xAxisOffset - (gridPaint.measureText( intervalStr ) / 2),**

**scrHeight,**

**gridPaint );**

**}**

**canvas.drawText( "Sec",**

**MAX\_SIGNALS\_LENGTH \* scaleWidth + getRelativePaddingAxisX() + gridPaint.measureText("00"),**

**scrHeight,**

**gridPaint );**

**}**

**/\*\***

**\* Draws a "KHz" marking at the top of the leftside y-axis**

**\* of the graph.**

**\*/**

**private void drawKHzKey( Canvas canvas ) {**

**canvas.drawText( "KHz",**

**0,**

**scrHeight - (SIGNALS\_HEIGHT \* scaleHeight) - getRelativePaddingAxisX() - getRelativeGridTextSize(),**

**gridPaint );**

**}**

**/\*\***

**\* Draws the small line markings on the leftside y-axis**

**\* of the graph to correlate with the KHz values.**

**\*/**

**private void drawKHzMarkings( Canvas canvas, int yAxisOffset ){**

**float markingPadding = gridPaint.measureText( "0" )\*2;**

**canvas.drawLine( getRelativePaddingAxisX(),**

**yAxisOffset - getRelativePaddingAxisY(),**

**getRelativePaddingAxisX() - markingPadding,**

**yAxisOffset - getRelativePaddingAxisY(),**

**gridPaint );**

**}**

**/\*\***

**\* Draws the KHz value markings on the leftside y-axis**

**\* of the graph.**

**\*/**

**private void drawKHzValues( Canvas canvas,**

**int frequencyValue,**

**int textOffsetX,**

**int yAxisOffset ){**

**canvas.drawText( String.valueOf( frequencyValue ),**

**textOffsetX,**

**yAxisOffset - getRelativePaddingAxisY() + 5,**

**gridPaint );**

**}**

**/\*\***

**\* Draws right-side Y-axis color graph and legend.**

**\* <br>**

**\* Used to indicate the amplitude value of the signal displayed.**

**\*/**

**private void drawGridAmplitude( Canvas canvas ){**

**drawAmplitudeSpectrum( canvas );**

**if( logScaling ){**

**drawAmplitudeValues(**

**canvas,**

**new String[]{ "-93", "-75", "-50", "-25", "0" } );**

**drawAmplitudeKey( canvas, "Log" );**

**} else {**

**drawAmplitudeValues(**

**canvas,**

**new String[]{ "0%", "25%", "50%", "75%", "100%" } );**

**drawAmplitudeKey( canvas, "Lin" );**

**}**

**}**

**// TODO move literal to variables**

**// TODO change 360 to 256**

**// TODO change color range to remove black and white**

**/\*\***

**\* Prints the legend for amplitude versus color**

**\* @param canvas Required for drawing, get from surfaceHolder**

**\*/**

**private void drawAmplitudeSpectrum( Canvas canvas )**

**{**

**int iteration\_max = getScalarHeight();**

**float graphWidth = getRelativeColorGraphWidth();**

**for( int i = 0; i <= COLOR\_GRAPH\_HEIGHT; ++ i ){**

**final int colorIter = (int)(5.688888888888889 \* (double) i);**

**colorSelecter(colorIter);**

**legendPaint.setARGB( COLOR\_RANGE - 1, red, green, blue );**

**float yAxisHeight = 0;**

**for( int j = 0; j < iteration\_max; ++ j ){**

**yAxisHeight = (float) scrHeight - iteration\_max\*i + j;**

**canvas.drawLine(**

**scrWidth-graphWidth,**

**yAxisHeight,**

**scrWidth,**

**yAxisHeight,**

**legendPaint );**

**}**

**}**

**}**

**/\*\***

**\* Draws dB labels for Color Graph**

**\*/**

**private void drawAmplitudeValues( Canvas canvas, String[] intervalValues ){**

**float graphWidth = getRelativeColorGraphWidth();**

**int intervalHeight = ( COLOR\_GRAPH\_HEIGHT \* getScalarHeight() ) / 4;**

**for( int i = 0; i < intervalValues.length; ++ i ){**

**canvas.drawText(**

**intervalValues[i],**

**scrWidth - graphWidth - gridPaint.measureText( intervalValues[i] + " "),**

**scrHeight - intervalHeight\*i,**

**gridPaint );**

**}**

**}**

**/\*\***

**\* @return Scalar constant for a pixel**

**\*/**

**private int getScalarHeight() {**

**return (scaleHeight > 1) ? scaleHeight-1 : 1;**

**}**

**/\*\***

**\* Draws a key type marking on top of the amplitude color graph**

**\*/**

**private void drawAmplitudeKey( Canvas canvas, String markingStr ){**

**int yAxisOffset = COLOR\_GRAPH\_HEIGHT \* getScalarHeight();**

**canvas.drawText( markingStr,**

**scrWidth - gridPaint.measureText( markingStr + " "),**

**scrHeight - yAxisOffset - getRelativeTopBarSize(),**

**gridPaint );**

**}**

**//TODO move literal to variables**

**/\*\***

**\* Draws the colorized bitmap column of the signal at the current signal\_offset**

**\* @param canvas Required for drawing, get from surfaceHolder**

**\* @param signal double array of 256 length**

**\* @param maxFFTSample**

**\*/**

**private void drawSpectrogramColumn( Canvas canvas,**

**double[] signal,**

**double maxFFTSample,**

**int offset )**

**{**

**int scaleWidthMult = 0;**

**int scaleHeightMult = 0;**

**float startX = getRelativePaddingAxisX();**

**float startY = scrHeight - getRelativePaddingAxisY();**

**float stopX = getRelativePaddingAxisX();**

**float stopY = scrHeight - getRelativePaddingAxisY() + 1 + scaleHeight;**

**// This is THE magic n**

**double magic\_n = 100.0;**

**double signalValue = 0.0;**

**int colorAmplitude = 0;**

**for (int i = 0; i < SIGNALS\_HEIGHT; ++i) {**

**if( logScaling ){**

**signalValue = 20 \* ((int) Math.log(signal[i]));**

**} else {**

**signalValue = (0.0 + signal[i]);**

**}**

**colorAmplitude = (int)((1024 / magic\_n) \* signalValue);**

**colorSelecter(colorAmplitude);**

**spectroPaint.setARGB(COLOR\_RANGE - 1, red, green, blue);**

**scaleWidthMult = scaleWidth \* offset;**

**scaleHeightMult = scaleHeight \* i;**

**canvas.drawLine((startX + scaleWidthMult),**

**(startY - scaleHeightMult),**

**(stopX + scaleWidthMult),**

**(stopY - scaleHeightMult),**

**spectroPaint);**

**}**

**}**

**@Override**

**public void surfaceChanged( SurfaceHolder holder,**

**int format,**

**int width,**

**int height )**

**{**

**this.scrWidth = width;**

**this.scrHeight = height;**

**Log.i("Surface Changed: ","Surface Changed: new width: "+width+" - new height: "+height);**

**}**

**@Override**

**public void surfaceCreated( SurfaceHolder holder )**

**{**

**Log.d("Surface Created: ","surfaceCreated");**

**isSurfaceCreated = true;**

**}**

**@Override**

**public void surfaceDestroyed(SurfaceHolder holder)**

**{**

**isSurfaceCreated = false;**

**}**

**/\*\***

**\* Sets the color to be drawn on the bitmap next**

**\* @param colorIterator**

**\*/**

**final void colorSelecter(final int colorIterator) {**

**int colorFactor;**

**if (colorIterator < 0) {**

**colorFactor = 0;**

**}**

**else {**

**colorFactor = colorIterator;**

**}**

**if (colorFactor / COLOR\_RANGE == 0) {**

**this.red = 0;**

**this.green = 0;**

**this.blue = colorFactor % COLOR\_RANGE;**

**}**

**if (colorFactor / COLOR\_RANGE == 1) {**

**this.red = 0;**

**this.green = colorFactor % COLOR\_RANGE;**

**this.blue = COLOR\_RANGE - 1;**

**}**

**if (colorFactor / COLOR\_RANGE == 2) {**

**this.red = 0;**

**this.green = COLOR\_RANGE - 1;**

**this.blue = (COLOR\_RANGE & COLOR\_RANGE - 1 - colorFactor);**

**}**

**if (colorFactor / COLOR\_RANGE == 3) {**

**this.red = colorFactor % COLOR\_RANGE;**

**this.green = COLOR\_RANGE - 1;**

**this.blue = 0;**

**}**

**if (colorFactor / COLOR\_RANGE == 4) {**

**this.red = COLOR\_RANGE - 1;**

**this.green = COLOR\_RANGE - 1 - colorFactor % COLOR\_RANGE;**

**this.blue = 0;**

**}**

**if (colorFactor / COLOR\_RANGE == 5) {**

**this.red = COLOR\_RANGE - 1;**

**this.green = 0;**

**this.blue = colorFactor % COLOR\_RANGE;**

**}**

**if (colorFactor / COLOR\_RANGE == 6) {**

**this.red = COLOR\_RANGE - 1;**

**this.green = colorFactor % COLOR\_RANGE;**

**this.blue = COLOR\_RANGE - 1;**

**}**

**if (colorFactor / COLOR\_RANGE == 7) {**

**this.red = COLOR\_RANGE - 1;**

**this.green = COLOR\_RANGE - 1;**

**this.blue = COLOR\_RANGE - 1;**

**}**

**}**

**public void setFrequency( double frequency ){**

**this.frequency = frequency;**

**}**

**public void drawButtonStart(){**

**drawTopRecordingButton( RENDERING\_RECORDING\_STR, android.graphics.Color.BLACK );**

**drawTopRecordingButton( STOP\_RECORDING\_STR, android.graphics.Color.BLACK );**

**drawTopRecordingButton( START\_RECORDING\_STR, android.graphics.Color.WHITE );**

**}**

**public void drawButtonStop() {**

**drawTopRecordingButton( RENDERING\_RECORDING\_STR, android.graphics.Color.BLACK );**

**drawTopRecordingButton( START\_RECORDING\_STR, android.graphics.Color.BLACK );**

**drawTopRecordingButton( STOP\_RECORDING\_STR, android.graphics.Color.WHITE );**

**}**

**/\*\***

**\* Indicates the application is currently recording in the background**

**\* for future rendering. Placed in the middle of the screen.**

**\*/**

**private void drawRecordingIndicator( int color ){**

**topBarPaint.setColor( color );**

**for( int i = 0; i < 4; ++ i ){**

**canvas = surfaceHolder.lockCanvas();**

**canvas.drawText(**

**RECORDING\_INDICATOR\_STR,**

**scrWidth/2 - topBarPaint.measureText( RECORDING\_INDICATOR\_STR )/2,**

**scrHeight/2,**

**topBarPaint );**

**surfaceHolder.unlockCanvasAndPost( canvas );**

**}**

**}**

**/\*\***

**\* Indicates the application is currently recording in the background**

**\* for future rendering.**

**\*/**

**public void drawRecordingIndicator(){**

**drawRecordingIndicator( android.graphics.Color.WHITE );**

**}**

**/\*\***

**\* Indicates the application is currently recording in the background**

**\* for future rendering.**

**\*/**

**public void undrawRecordingIndicator(){**

**drawRecordingIndicator( android.graphics.Color.BLACK );**

**}**

**public void drawButtonRendering(){**

**drawTopRecordingButton( START\_RECORDING\_STR, android.graphics.Color.BLACK );**

**drawTopRecordingButton( STOP\_RECORDING\_STR, android.graphics.Color.BLACK );**

**drawTopRecordingButton( RENDERING\_RECORDING\_STR, android.graphics.Color.WHITE );**

**}**

**private void drawTopRecordingButton( String message, int color ){**

**float textPadding = getRelativeButtonPadding();**

**float relativeSize = getRelativeTopBarSize();**

**topBarPaint.setColor( color );**

**for( int i = 0; i < 4; ++ i ){**

**canvas = surfaceHolder.lockCanvas();**

**canvas.drawText( message, textPadding, relativeSize + textPadding, topBarPaint );**

**surfaceHolder.unlockCanvasAndPost( canvas );**

**}**

**}**

**public float getRelativeTopBarSize(){**

**if( relativeTopBarSize == null ){**

**relativeTopBarSize = TypedValue.applyDimension(**

**TypedValue.COMPLEX\_UNIT\_DIP,**

**TOP\_BAR\_TEXT\_SIZE,**

**resources.getDisplayMetrics() );**

**}**

**return relativeTopBarSize;**

**}**

**public float getRelativeButtonPadding(){**

**return TypedValue.applyDimension(**

**TypedValue.COMPLEX\_UNIT\_DIP,**

**TOP\_PADDING,**

**resources.getDisplayMetrics() );**

**}**

**private float getRelativeGridTextSize(){**

**if( relativeGridTextSize == null ){**

**relativeGridTextSize = TypedValue.applyDimension(**

**TypedValue.COMPLEX\_UNIT\_DIP,**

**GRID\_TEXT\_SIZE,**

**resources.getDisplayMetrics() );**

**}**

**return relativeGridTextSize;**

**}**

**private float getRelativeColorGraphWidth(){**

**if( relativeColorGraphWidth == null ){**

**relativeColorGraphWidth = TypedValue.applyDimension(**

**TypedValue.COMPLEX\_UNIT\_DIP,**

**COLOR\_GRAPH\_WIDTH,**

**resources.getDisplayMetrics() );**

**}**

**return relativeColorGraphWidth;**

**}**

**private float getRelativePaddingAxisX(){**

**if( relativePaddingAxisX == null ){**

**relativePaddingAxisX = TypedValue.applyDimension(**

**TypedValue.COMPLEX\_UNIT\_DIP,**

**SIGNAL\_PADDING\_X\_AXIS,**

**resources.getDisplayMetrics() );**

**}**

**return relativePaddingAxisX;**

**}**

**private float getRelativePaddingAxisY(){**

**if( relativePaddingAxisY == null ){**

**relativePaddingAxisY = TypedValue.applyDimension(**

**TypedValue.COMPLEX\_UNIT\_DIP,**

**SIGNAL\_PADDING\_Y\_AXIS,**

**resources.getDisplayMetrics() );**

**}**

**return relativePaddingAxisY;**

**}**

**/\*\***

**\* Erases the canvas and sets it entirely to black**

**\*/**

**public void clearSprectrogramView() {**

**for( int i = 0; i < 4; ++ i ){**

**canvas = surfaceHolder.lockCanvas();**

**canvas.drawColor( Color.BLACK );**

**surfaceHolder.unlockCanvasAndPost( canvas );**

**}**

**}**

**public void drawSpectrogramRange( double[][] signals,**

**double frequency,**

**int buffer\_size,**

**double maxFFTSample,**

**int intervalLenth,**

**int signal\_count ){**

**for( int j = 0; j < 4; ++ j ){**

**canvas = surfaceHolder.lockCanvas();**

**for( int i = signal\_count - intervalLenth;**

**i < signal\_count;**

**++ i ){**

**drawSpectrogramColumn(**

**canvas, signals[i], maxFFTSample, i );**

**}**

**surfaceHolder.unlockCanvasAndPost(canvas);**

**}**

**}**

**/\*\***

**\* Draws labeled X & Y axes, and color graph**

**\*/**

**public void drawGrid(){**

**for( int i = 0; i < 4; i++ ){**

**canvas = surfaceHolder.lockCanvas();**

**drawGridAmplitude( canvas );**

**drawGridKHz( canvas );**

**drawGridTime( canvas );**

**surfaceHolder.unlockCanvasAndPost( canvas );**

**}**

**}**

**/\*\***

**\* Displays the duration of a signal sample as a decimal value in the**

**\* top left corner.<br><br>**

**\* @param duration time in milliseconds**

**\*/**

**public void drawTopDuration( long duration ){**

**float time = (float) duration / 1000;**

**float textPadding = getRelativeButtonPadding();**

**float relativeSize = getRelativeTopBarSize();**

**Paint paint = getPaintTopBar();**

**topBarPaint.setColor( android.graphics.Color.WHITE );**

**String durationMessage = "Time: " + String.format("%.2f", time) + "s ";**

**for(int i=0;i<4;i++){**

**canvas = surfaceHolder.lockCanvas();**

**canvas.drawText(**

**durationMessage,**

**scrWidth - paint.measureText( durationMessage ) - textPadding,**

**relativeSize + textPadding,**

**paint );**

**surfaceHolder.unlockCanvasAndPost( canvas );**

**}**

**}**

**private Paint getPaintTopBar() {**

**if( topBarPaint == null ){**

**topBarPaint = new Paint();**

**topBarPaint.setColor( android.graphics.Color.WHITE );**

**topBarPaint.setTextSize( getRelativeTopBarSize() );**

**}**

**return topBarPaint;**

**}**

**private Paint getPaintGrid() {**

**if( gridPaint == null ){**

**gridPaint = new Paint();**

**gridPaint.setColor( android.graphics.Color.WHITE );**

**gridPaint.setTextSize( getRelativeGridTextSize() );**

**}**

**return gridPaint;**

**}**

**public void setSurfaceHolder( SurfaceHolder surfaceHolder ){**

**this.surfaceHolder = surfaceHolder;**

**}**

**/\*\***

**\* Draws top X-axis centered sample rate text string in dark grey**

**\*/**

**public void drawTopSampleRate(){**

**float textPadding = getRelativeButtonPadding();**

**float relativeSize = getRelativeTopBarSize();**

**topBarPaint.setColor( android.graphics.Color.GRAY );**

**String sampleRateMessage = "Sample Rate: " + String.format("%.0f", frequency) + " Hz";**

**for( int i = 0; i < 4 ; ++ i ){**

**canvas = surfaceHolder.lockCanvas();**

**canvas.drawText(**

**sampleRateMessage,**

**scrWidth/2 - topBarPaint.measureText( sampleRateMessage )/2,**

**relativeSize + textPadding,**

**topBarPaint );**

**surfaceHolder.unlockCanvasAndPost( canvas );**

**}**

**}**

**public void setLogScaling( boolean scaling ){**

**logScaling = scaling;**

**}**

**}**

## SpectrogramFileIO.java

package edu.ucdenver.aprad.spectrogram;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.util.Locale;

import java.util.Scanner;

import android.app.Activity;

import android.content.Context;

public class SpectrogramFileIO extends Activity {

private final Integer SIGNAL\_COUNT = 700;

private final Integer SIGNAL\_LENGTH = 256;

public void writeFile( double[][] signals ){

String filename = "test\_file"; //String.valueOf( System.currentTimeMillis() ) + ".aprad";

String data = new String();

for( int i = 0; i < 1; ++ i ){

for( int j = 0; j < signals[i].length; ++ j ){

data += String.valueOf( signals[i][j] );

if( j != signals[i].length - 1 ){

data += ", ";

} else {

data += "\n";

}

}

}

try {

FileOutputStream output = openFileOutput( filename, Context.MODE\_PRIVATE );

output.write(data.getBytes());

output.close();

} catch (Exception e) {

e.printStackTrace();

}

}

public double[][] readFile( Context context, String filename ){

double[][] signals = new double[SIGNAL\_COUNT][SIGNAL\_LENGTH];

try {

FileInputStream input = openFileInput( filename );

Scanner inputScanner = new Scanner(input);

inputScanner.useLocale(Locale.US);

int j = 0;

while( inputScanner.hasNextLine() && j < SIGNAL\_COUNT ){

String line = inputScanner.nextLine();

Scanner lineScanner = new Scanner(line);

lineScanner.useDelimiter(",\\s\*");

int i = 0;

while( lineScanner.hasNextDouble() && i < SIGNAL\_LENGTH ){

signals[i][j] = lineScanner.nextDouble();

++ i;

}

++ j;

}

} catch ( FileNotFoundException e ) {

e.printStackTrace();

}

return signals;

}

}

## SaveDialog.java

package edu.ucdenver.aprad.spectrogram;

import edu.ucdenver.aprad.R.string;

import android.R;

import android.app.AlertDialog;

import android.app.Dialog;

import android.app.DialogFragment;

import android.content.DialogInterface;

import android.os.Bundle;

public class SaveDialog extends DialogFragment {

@Override

public Dialog onCreateDialog(Bundle savedInstanceState) {

// Use the Builder class for convenient dialog construction

AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

builder.setMessage(string.save\_dialog)

.setPositiveButton(string.save, new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int id) {

// FIRE ZE MISSILES!

}

})

.setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int id) {

// User cancelled the dialog

}

});

// Create the AlertDialog object and return it

return builder.create();

}

}

## APRAD.java

package edu.ucdenver.aprad;

import edu.ucdenver.aprad.R;

import edu.ucdenver.aprad.education.Education;

import edu.ucdenver.aprad.info.About;

import edu.ucdenver.aprad.info.Help;

import edu.ucdenver.aprad.spectrogram.Spectrogram;

import edu.ucdenver.aprad.spectrum\_analyzer.SpectrumAnalyzer;

import android.app.Activity;

import android.content.Context;

import android.content.Intent;

import android.content.SharedPreferences;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuInflater;

import android.view.MenuItem;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;

public class Aprad extends Activity {

Button spectrumAnalyzerButton;

Button spectrogramButton;

Button educationButton;

Button helpButton;

Button aboutButton;

public static final String SHARED\_PREFERENCES = "SharedPreferences";

SharedPreferences sharedPreferences;

@Override

public void onCreate( Bundle savedInstanceState ){

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

getSharedPreferences(SHARED\_PREFERENCES, Context.MODE\_PRIVATE);

spectrumAnalyzerButton = (Button) findViewById(R.id.spectrumAnalyzerButton);

spectrumAnalyzerButton.setOnClickListener( new OnClickListener() {

@Override

public void onClick( View v ){

Intent intent = new Intent( v.getContext(), SpectrumAnalyzer.class);

v.getContext().startActivity(intent);

}

});

spectrogramButton = (Button) findViewById(R.id.spectrogramButton);

spectrogramButton.setOnClickListener( new OnClickListener() {

@Override

public void onClick( View v ){

Intent intent = new Intent( v.getContext(), Spectrogram.class);

v.getContext().startActivity(intent);

}

});

educationButton = (Button) findViewById(R.id.educationButton);

educationButton.setOnClickListener( new OnClickListener() {

@Override

public void onClick( View v ){

Intent intent = new Intent( v.getContext(), Education.class);

v.getContext().startActivity(intent);

}

});

helpButton = (Button) findViewById(R.id.helpButton);

helpButton.setOnClickListener( new OnClickListener() {

@Override

public void onClick( View v ){

Intent intent = new Intent( v.getContext(), Help.class);

v.getContext().startActivity(intent);

}

});

aboutButton = (Button) findViewById(R.id.aboutButton);

aboutButton.setOnClickListener( new OnClickListener() {

@Override

public void onClick( View v ){

Intent intent = new Intent( v.getContext(), About.class);

v.getContext().startActivity(intent);

}

});

}

@Override

public boolean onCreateOptionsMenu(Menu menu)

{

MenuInflater inflater = getMenuInflater();

inflater.inflate(R.menu.aprad, menu);

return true;

}

@Override

public boolean onOptionsItemSelected( MenuItem item ){

// Handle item selection

switch ( item.getItemId() ){

case R.id.settings:

Intent settingsIntent = new Intent( Aprad.this, Preferences.class );

settingsIntent.setFlags( Intent.FLAG\_ACTIVITY\_NEW\_TASK

| Intent.FLAG\_ACTIVITY\_CLEAR\_TASK );

this.startActivity(settingsIntent);

return true;

case R.id.open\_file:

Intent openSavedDataIntent = new Intent( Aprad.this,

OpenSavedData.class );

this.startActivity(openSavedDataIntent);

return true;

default:

return super.onOptionsItemSelected(item);

}

}

}

## SplashScreen.java

package edu.ucdenver.aprad;

import android.app.Activity;

import android.content.Intent;

import android.os.Bundle;

import android.os.Handler;

public class SplashScreen extends Activity {

private static int TIMER = 3000;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.splash);

new Handler().postDelayed(new Runnable() {

@Override

public void run() {

Intent intent = new Intent( SplashScreen.this, Aprad.class );

startActivity( intent );

// close this activity

finish();

}

}, TIMER);

}

}

## About.java

package edu.ucdenver.aprad.info;

import android.app.Activity;

import android.os.Bundle;

import edu.ucdenver.aprad.R;

/\*\*

\* Displays information about who made the program.

\*

\* @author Robbie Perlstein

\* @author Dan Rolls

\* @author Michael Dewar

\*/

public class About extends Activity {

@Override

public void onCreate(Bundle savedInstanceState)

{

super.onCreate(savedInstanceState);

setContentView(R.layout.about);

}

}

## Help.java

package edu.ucdenver.aprad.info;

import android.app.Activity;

import android.os.Bundle;

import edu.ucdenver.aprad.R;

/\*\*

\* Displays information about how to use the program.

\*

\* @author Robbie Perlstein

\* @author Dan Rolls

\* @author Michael Dewar

\*/

public class Help extends Activity {

@Override

public void onCreate(Bundle savedInstanceState)

{

super.onCreate(savedInstanceState);

setContentView(R.layout.help);

}

}

## APRAD\_UCD manifest.xml

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<manifest xmlns:android=*"http://schemas.android.com/apk/res/android"*

package=*"edu.ucdenver.aprad"*

android:versionCode=*"1"*

android:versionName=*"1.0"* >

<instrumentation android:name=*"android.test.InstrumentationTestRunner"*

android:targetPackage=*"edu.ucdenver.aprad"*

android:label=*"APRAD Tests"* />

<uses-sdk

android:minSdkVersion=*"13"*

android:targetSdkVersion=*"16"* />

<uses-permission android:name=*"android.permission.RECORD\_AUDIO"*/>

<application

android:allowBackup=*"true"*

android:icon=*"@drawable/icon"*

android:label=*"@string/app\_name"*

android:theme=*"@style/AppTheme"* >

<uses-library android:name=*"android.test.runner"* />

<activity

android:name=*"edu.ucdenver.aprad.SplashScreen"*

android:label=*"@string/app\_name"*

android:screenOrientation=*"portrait"*

android:configChanges=*"orientation|keyboardHidden"* >

<intent-filter>

<action android:name=*"android.intent.action.MAIN"* />

<category android:name=*"android.intent.category.LAUNCHER"* />

</intent-filter>

</activity>

<activity android:name=*"edu.ucdenver.aprad.Aprad"*></activity>

<activity android:name=*"edu.ucdenver.aprad.spectrum\_analyzer.SpectrumAnalyzer"*

android:screenOrientation=*"landscape"*

android:configChanges=*"orientation|keyboardHidden"* ></activity>

<activity android:name=*"edu.ucdenver.aprad.spectrogram.Spectrogram"*

android:screenOrientation=*"landscape"*

android:configChanges=*"orientation|keyboardHidden"* ></activity>

<activity android:name=*"edu.ucdenver.aprad.education.Education"*></activity>

<activity android:name=*"edu.ucdenver.aprad.info.Help"*></activity>

<activity android:name=*"edu.ucdenver.aprad.info.About"*></activity>

<activity android:name=*"edu.ucdenver.aprad.Preferences"*></activity>

<activity android:name=*"edu.ucdenver.aprad.OpenSavedData"*></activity>

</application>

</manifest>

# JavaDoc

## Overview

|  |  |
| --- | --- |
| **Interface Summary** | |
| [**AudioRecordListener**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecordListener.html) | Interface for AudioRecord to let SpecAnalyView to start drawing |

|  |  |
| --- | --- |
| **Class Summary** | |
| [**AudioRecorder**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecorder.html) | AudioRecorder Extends Thread to run the AudioRecorder in it's own thread |
| [**BuildConfig**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\BuildConfig.html) |  |
| [**Complex**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html) | Complex Object |
| [**FFT**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\FFT.html) | Class FFT Used to do a Cooley-Tukey FFT type |
| [**R**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\R.html) |  |
| [**R.attr**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\R.attr.html) |  |
| [**R.dimen**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\R.dimen.html) |  |
| [**R.drawable**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\R.drawable.html) |  |
| [**R.id**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\R.id.html) |  |
| [**R.layout**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\R.layout.html) |  |
| [**R.menu**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\R.menu.html) |  |
| [**R.string**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\R.string.html) |  |
| [**R.style**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\R.style.html) |  |
| [**SpecAnalyView**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\SpecAnalyView.html) | Class SpecAnalyView Handles drawing the results to the UI |
| [**SpectrumAnalyzer**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\SpectrumAnalyzer.html) | SpectrumAnalyzer main class which extends Activity |

## Class SpectrumAnalyzer

java.lang.Object

extended by Activity

extended by **com.ucdenver.aprad\_v2.SpectrumAnalyzer**

**All Implemented Interfaces:**

[AudioRecordListener](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecordListener.html)

public class **SpectrumAnalyzer**

extends Activity

implements [AudioRecordListener](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecordListener.html)

SpectrumAnalyzer main class which extends Activity

**Author:**

Dan Rolls, Michael Dewar

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**SpectrumAnalyzer**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\SpectrumAnalyzer.html#SpectrumAnalyzer%28%29)() |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| void | [**onCreate**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\SpectrumAnalyzer.html#onCreate%28Bundle%29)(Bundle savedInstanceState)            Called when the activity is first created. |
| void | [**onDrawableSampleAvailable**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\SpectrumAnalyzer.html#onDrawableSampleAvailable%28double[]%29)(double[] signal)            Implements listener. |

|  |
| --- |
| **Methods inherited from class java.lang.Object** |
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

|  |
| --- |
| **Constructor Detail** |

### SpectrumAnalyzer

public **SpectrumAnalyzer**()

|  |
| --- |
| **Method Detail** |

### onCreate

public void **onCreate**(Bundle savedInstanceState)

Called when the activity is first created.

### onDrawableSampleAvailable

public void **onDrawableSampleAvailable**(double[] signal)

Implements listener. Creates a new Runnable but runs it on the UI thread

**Specified by:**

[onDrawableSampleAvailable](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecordListener.html#onDrawableSampleAvailable%28double[]%29) in interface [AudioRecordListener](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecordListener.html)

## Interface AudioRecordListener

**All Known Implementing Classes:**

[SpectrumAnalyzer](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\SpectrumAnalyzer.html)

public interface **AudioRecordListener**

Interface for AudioRecord to let SpecAnalyView to start drawing

**Author:**

Dan Rolls, Michael Dewar

|  |  |
| --- | --- |
| **Method Summary** | |
| void | [**onDrawableSampleAvailable**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecordListener.html#onDrawableSampleAvailable%28double[]%29)(double[] signal) |

|  |
| --- |
| **Method Detail** |

### onDrawableSampleAvailable

void **onDrawableSampleAvailable**(double[] signal)

## Class AudioRecorder

java.lang.Object

extended by java.lang.Thread

extended by **com.ucdenver.aprad\_v2.AudioRecorder**

**All Implemented Interfaces:**

java.lang.Runnable

public class **AudioRecorder**

extends java.lang.Thread

AudioRecorder Extends Thread to run the AudioRecorder in it's own thread

**Author:**

Dan Rolls, Michael Dewar

|  |
| --- |
| **Nested Class Summary** |

|  |
| --- |
| **Nested classes/interfaces inherited from class java.lang.Thread** |
| java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler |

|  |
| --- |
| **Field Summary** |

|  |
| --- |
| **Fields inherited from class java.lang.Thread** |
| MAX\_PRIORITY, MIN\_PRIORITY, NORM\_PRIORITY |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**AudioRecorder**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecorder.html#AudioRecorder%28double,%20int%29)(double sampleRate, int points)            Constructor for AudioRecorder |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| void | [**end**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecorder.html#end%28%29)()            Ends the thread |
| double | [**getMaxFFTSample**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecorder.html#getMaxFFTSample%28%29)()            Used to get Max FFT sample value from FFT Class |
| void | [**notifyListenerDrawSpectrum**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecorder.html#notifyListenerDrawSpectrum%28double[]%29)(double[] fftSignal)            Lets SpectrumAnalyzer know to go ahead and draw |
| static void | [**registerFFTAvailListener**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecorder.html#registerFFTAvailListener%28com.ucdenver.aprad_v2.AudioRecordListener%29)([AudioRecordListener](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecordListener.html) listener)            register the audiolistener to the class listner |
| void | [**run**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecorder.html#run%28%29)()            To use Thread, run must be implemented Starts the AudioRecorder and collects data for transformation |
| static void | [**unregisterFFTAvailListener**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecorder.html#unregisterFFTAvailListener%28%29)()            Unregisters listener. |

|  |
| --- |
| **Methods inherited from class java.lang.Thread** |
| activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getAllStackTraces, getContextClassLoader, getDefaultUncaughtExceptionHandler, getId, getName, getPriority, getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setDefaultUncaughtExceptionHandler, setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start, stop, stop, suspend, toString, yield |

|  |
| --- |
| **Methods inherited from class java.lang.Object** |
| equals, getClass, hashCode, notify, notifyAll, wait, wait, wait |

|  |
| --- |
| **Constructor Detail** |

### AudioRecorder

public **AudioRecorder**(double sampleRate,

int points)

Constructor for AudioRecorder

**Parameters:**

sampleRate - Frequency of samples in Hz

points - The number of FFT samples to take

|  |
| --- |
| **Method Detail** |

### run

public void **run**()

To use Thread, run must be implemented Starts the AudioRecorder and collects data for transformation

**Specified by:**

run in interface java.lang.Runnable

**Overrides:**

run in class java.lang.Thread

### getMaxFFTSample

public double **getMaxFFTSample**()

Used to get Max FFT sample value from FFT Class

**Returns:**

Max FFT Sample from the FFT Processing

### end

public void **end**()

Ends the thread

### registerFFTAvailListener

public static void **registerFFTAvailListener**([AudioRecordListener](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\AudioRecordListener.html) listener)

register the audiolistener to the class listner

**Parameters:**

listener - AudioRecord listener

### unregisterFFTAvailListener

public static void **unregisterFFTAvailListener**()

Unregisters listener. Sets listener to null

### notifyListenerDrawSpectrum

public void **notifyListenerDrawSpectrum**(double[] fftSignal)

Lets SpectrumAnalyzer know to go ahead and draw

**Parameters:**

fftSignal - The raw data converted into an FFT

## Class Complex

java.lang.Object

extended by **com.ucdenver.aprad\_v2.Complex**

public class **Complex**

extends java.lang.Object

Complex Object

Various attributes and methods related to calculation of Complex numbers.

**Version:**

1.0

**Author:**

Dan Rolls, Michael Dewar

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**Complex**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html#Complex%28double,%20double%29)(double real, double imag)            Constructor for Complex class |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| double | [**imaginary**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html#imaginary%28%29)()            Returns imaginary part |
| [Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html) | [**minus**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html#minus%28com.ucdenver.aprad_v2.Complex%29)([Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html) b)            Subtraction of two complex numbers |
| [Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html) | [**plus**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html#plus%28com.ucdenver.aprad_v2.Complex%29)([Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html) b)            Addition of two complex numbers |
| double | [**real**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html#real%28%29)()            Returns real part |
| [Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html) | [**times**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html#times%28com.ucdenver.aprad_v2.Complex%29)([Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html) b)            Multiplication of two complex numbers |
| java.lang.String | [**toString**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html#toString%28%29)()            toString class returner |

|  |
| --- |
| **Methods inherited from class java.lang.Object** |
| equals, getClass, hashCode, notify, notifyAll, wait, wait, wait |

|  |
| --- |
| **Constructor Detail** |

### Complex

public **Complex**(double real,

double imag)

Constructor for Complex class

**Parameters:**

real - The real part of the complex number

imag - The imaginary part of the complex number

|  |
| --- |
| **Method Detail** |

### plus

public [Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html) **plus**([Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html) b)

Addition of two complex numbers

**Parameters:**

b - The comples number to be evaluated

**Returns:**

this Complex plus b Complex

### minus

public [Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html) **minus**([Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html) b)

Subtraction of two complex numbers

**Parameters:**

b - The complex number to be evaluated

**Returns:**

this Complex minux b Complex

### times

public [Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html) **times**([Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html) b)

Multiplication of two complex numbers

**Parameters:**

b - The complex to be evaluated

**Returns:**

this Complex times b Complex

### real

public double **real**()

Returns real part

**Returns:**

real part of complex number

### imaginary

public double **imaginary**()

Returns imaginary part

**Returns:**

imaginary part of complex number

### toString

public java.lang.String **toString**()

toString class returner

**Overrides:**

toString in class java.lang.Object

**Returns:**

String representation of Complex class

## Class FFT

java.lang.Object

extended by **com.ucdenver.aprad\_v2.FFT**

public class **FFT**

extends java.lang.Object

Class FFT Used to do a Cooley-Tukey FFT type

**Author:**

Dan Rolls, Michael Dewar

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**FFT**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\FFT.html#FFT%28int%29)(int points)            Constructor |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| double[] | [**calculateFFT**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\FFT.html#calculateFFT%28byte[]%29)(byte[] signal)            Start doing calculations for FFT |
| static [Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html)[] | [**fft**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\FFT.html#fft%28com.ucdenver.aprad_v2.Complex[]%29)([Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html)[] complex)            Provides the finalized FFT result |
| double | [**getMaxFFTSample**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\FFT.html#getMaxFFTSample%28%29)()            Retrieves the maxFFT Sample |

|  |
| --- |
| **Methods inherited from class java.lang.Object** |
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

|  |
| --- |
| **Constructor Detail** |

### FFT

public **FFT**(int points)

Constructor

**Parameters:**

points - The number of FFT Samples

|  |
| --- |
| **Method Detail** |

### calculateFFT

public double[] **calculateFFT**(byte[] signal)

Start doing calculations for FFT

**Parameters:**

signal - Raw byte signal from AudioRecorder

**Returns:**

double[] of FFT results

### getMaxFFTSample

public double **getMaxFFTSample**()

Retrieves the maxFFT Sample

**Returns:**

### fft

public static [Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html)[] **fft**([Complex](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\Complex.html)[] complex)

Provides the finalized FFT result

**Parameters:**

complex - Complex number

**Returns:**

an array of complex numbers

## Class SpecAnalyView

java.lang.Object

extended by SurfaceView

extended by **com.ucdenver.aprad\_v2.SpecAnalyView**

public class **SpecAnalyView**

extends SurfaceView

Class SpecAnalyView Handles drawing the results to the UI

**Author:**

Dan Rolls, Michael Dewar

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**SpecAnalyView**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\SpecAnalyView.html#SpecAnalyView%28Context%29)(Context context)            Constructor for SpecAnalyView passing in only the context |  |
| [**SpecAnalyView**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\SpecAnalyView.html#SpecAnalyView%28Context,%20AttributeSet%29)(Context context, AttributeSet attrs)            Constructor for SpecAnalyView |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| void | [**drawSpectrum**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\SpecAnalyView.html#drawSpectrum%28double[],%20double,%20int,%20double%29)(double[] signal, double samplingRate, int numberOfFFTPoints, double maxFFTSample)            Main class to handle the drawing of the UI |
| void | [**surfaceChanged**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\SpecAnalyView.html#surfaceChanged%28SurfaceHolder,%20int,%20int,%20int%29)(SurfaceHolder holder, int format, int width, int height)            Override of the surfaceChanged class |
| void | [**surfaceCreated**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\SpecAnalyView.html#surfaceCreated%28SurfaceHolder%29)(SurfaceHolder holder)            Override of the surfaceCreated class |
| void | [**surfaceDestroyed**](file:///D:\data\eclipse\Dropbox\csc4738\workspace\APRAD_V2\doc\com\ucdenver\aprad_v2\SpecAnalyView.html#surfaceDestroyed%28SurfaceHolder%29)(SurfaceHolder holder)            Override of the surfaceDestroyed class |

|  |
| --- |
| **Methods inherited from class java.lang.Object** |
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

|  |
| --- |
| **Constructor Detail** |

### SpecAnalyView

public **SpecAnalyView**(Context context,

AttributeSet attrs)

Constructor for SpecAnalyView

**Parameters:**

context - Context of the application

attrs - Attribute set of the application

### SpecAnalyView

public **SpecAnalyView**(Context context)

Constructor for SpecAnalyView passing in only the context

**Parameters:**

context - Context of the application

|  |
| --- |
| **Method Detail** |

### drawSpectrum

public void **drawSpectrum**(double[] signal,

double samplingRate,

int numberOfFFTPoints,

double maxFFTSample)

Main class to handle the drawing of the UI

**Parameters:**

signal - Array of processed FFT bytes

samplingRate - The rate at which the samples were harvested

numberOfFFTPoints - The number of FFT points analyzed

maxFFTSample - The max size of the FFT sample

### surfaceChanged

public void **surfaceChanged**(SurfaceHolder holder,

int format,

int width,

int height)

Override of the surfaceChanged class

**Parameters:**

holder - The surface holder

format - Screen format

width - Screen width

height - screen height

### surfaceCreated

public void **surfaceCreated**(SurfaceHolder holder)

Override of the surfaceCreated class

**Parameters:**

holder - The surface holder

### surfaceDestroyed

public void **surfaceDestroyed**(SurfaceHolder holder)

Override of the surfaceDestroyed class

**Parameters:**

holder - the surface holder

# APRAD User Manual

There are two parts to the APRAD system, the receiving antenna and the display app for an Android smartphone.

## APRAD Antenna:

1. Ensure that the jumper next to the primary of the transformer and the SMA jumper are disconnected (these are used for testing purposes). Ensure that the 3.5mm jumper and that only one of the gain jumpers are connected.

2. Ensure that a charged 12V battery is inserted in the battery holder.

3. Connect the magnetic loop antenna to the primary of the transformer through the 3-pin connector.

4. Connect a receiving device that is capable of receiving audio signals (e.g., phone or computer) to the audio jack using a double-ended male 3.5mm 4-pin audio cable.

5. Using the receiving device, determine if the strength of the signal is correct. If the strength is not correct, adjust the gain of the amplifier by moving the gain jumper. If the ADJ (adjustable) gain jumper is chosen, then adjustments need to be made on the trim-pot next to the jumpers.

6. Once the desired signal strength has been obtained, the system is ready to be used. Move the antenna in both azimuth and zenith directions to explore received magnetic field strengths relative to frequency.

7. To turn off the system, unplug the antenna and receiving device. Remove the battery and carefully store the system.

Note: When the voltage of the battery becomes less than 7V, a green LED (labeled: Low Battery) will light - indicating that the battery needs to be changed.

## APRAD Smartphone Application:

### System Requirements:

1. Operating System: This application is only for Android OS version 17 (4.1.2 - Jellybean) and above. The software is not backward compatible. There is currently no iOS version of the software available.
2. Processor: The software is optimized for a dual-core, 1200MHz processor. Performance on single-core systems or slower processing speed may be unacceptable.
3. Required Memory: 1024 MB RAM.
4. Required built-in storage: 16 GB. Recommended: 32 GB. Less storage may be used, but it will affect the amount of data that can be captured and recorded by the software.
5. Storage expansion: 32GB Micro SD card is recommended for large data collection monitoring and storage.
6. Device input: The mobile device used with the APRAD software must have a 3.5mm ear jack (TRRS) for input from the antenna. The antenna does not work with a MicroUSB input.

## Installation:

### Using a computer:

1. Download the APRAD app (aprad.apk) from the CU Denver web page.
2. Plug the destination device into the computer using a USB cable.
3. Transfer the app to the mobile device.

### Using the mobile device:

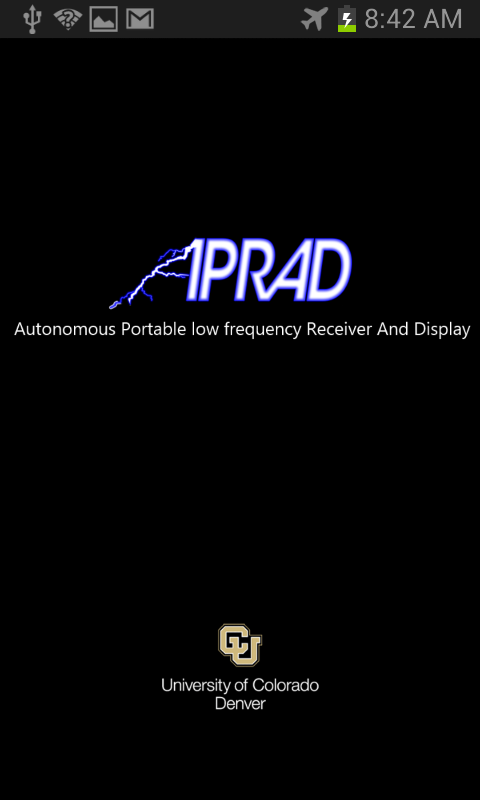
1. Navigate to the settings menu on the Android device.
2. Check the ‘Unknown Sources’ box to allow installation of non-Market apps. You can return to this setting immediately after installing the APRAD software to uncheck the box, if desired.
3. Download the APRAD app (aprad.apk) from the CU Denver web page.
4. If you are prompted to confirm the installation of the app, click ‘Yes’ or ‘OK’.

## Using the Software:

There are two main functions included with the APRAD software: the Spectrum Analyzer and the Spectrogram. The app’s features will work with or without the APRAD Antenna attached to the mobile device. The antenna is used when conducting an electro-magnetic survey. The antenna may be unplugged in order to trouble-shoot the app with audio-frequency data received by the built-in microphone on the device.

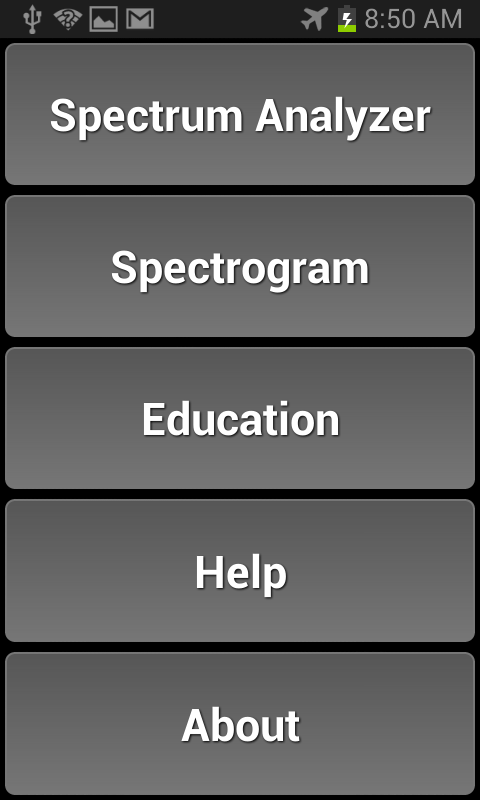
### APRAD Splash Screen:

When the application launches, the splash screen is displayed:



### Menu Screen:

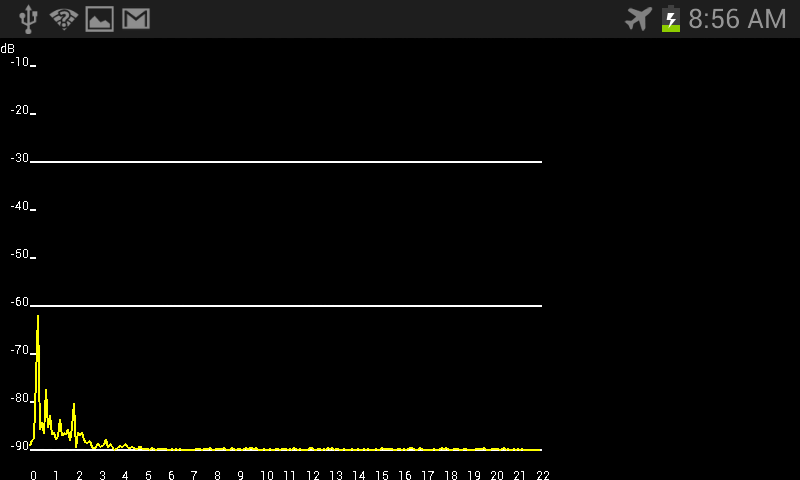
The first screen allows a user to choose which part of the app to select, the Spectrum Analyzer, for instantaneous readings of the environment, or the Spectrogram, for a display of frequency and amplitude data over time. There are also informational sections to the App including Help and About. These will be discussed separately.



Simply press the button corresponding to the desired analysis tool.

### Spectrum Analyzer:

The Spectrum Analyzer displays frequency data along the horizontal axis, and amplitude intensity data along the vertical axis.



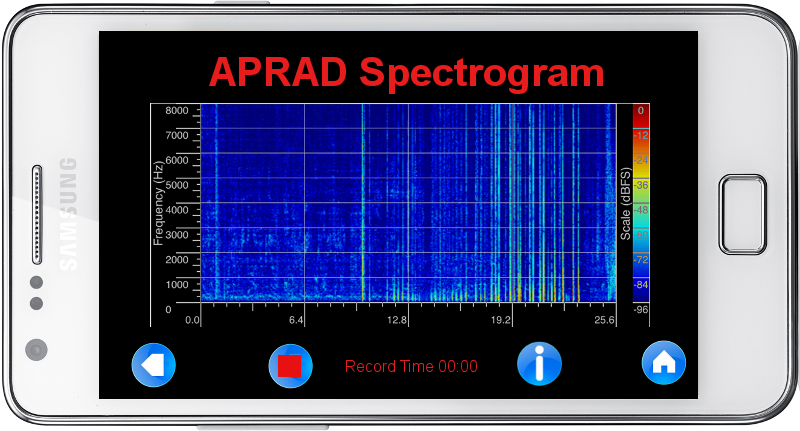
The ‘Back’ button on the phone/device itself (not in the display window of the phone) will return to the APRAD Menu screen. Please refer to the Operations Manual of the device on which the app is installed for specific instructions.

The ‘Home’ button on the phone/device will minimize the APRAD app and display the app area of the device. Pressing the APRAD app again will return to the app in the Spectrum Analyzer area.

The features of the Spectrum Analyzer include a frequency vs. dB plot that updates in real time to display the amplitude of incoming signals from the APRAD antenna. The taller the peak in the Analyzer display, the more intense the signal at that frequency.

### Spectrogram:

The spectrogram displays frequency and amplitude data over time.



There is a scale to the right side of the display for reference for the amplitude colors used.

The ‘Back’ button in the lower left side of the screen will return to the previous screen.

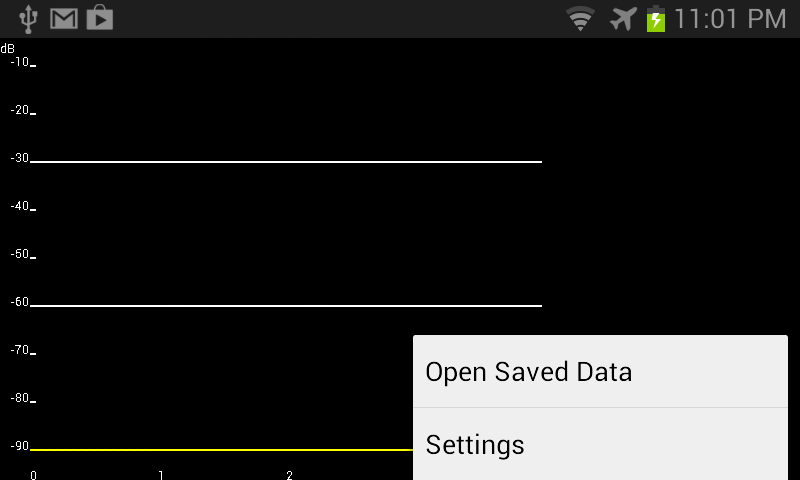
The ‘Home’ button in the lower right side of the screen will return to the main title screen for the APRAD app.

The ‘Settings’ button on the lower tight of the display will open the Settings menu where options are available for the capture and display of data (see #4, Settings screen below).

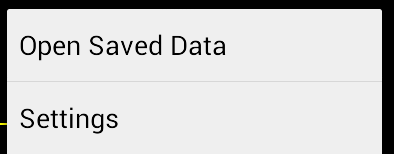
The ‘Record’ button on the lower left of the display will start recording data from the APRAD Antenna (or the built-in device microphone, if the antenna is unplugged). Up to 60 seconds of data can be recorded and stored for future reference.

### Settings Menu:

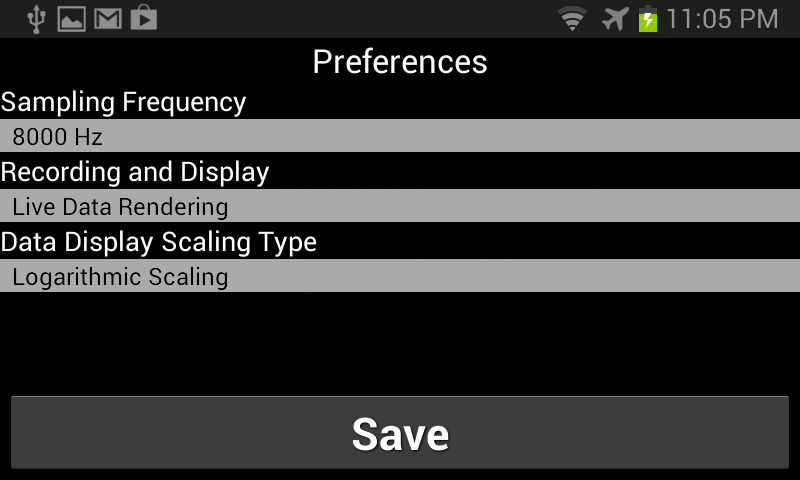
The Settings screen is where adjustments can be made in the audio sample rate and the number of FFT sample per second.



The location of the Settings pop-up menu.

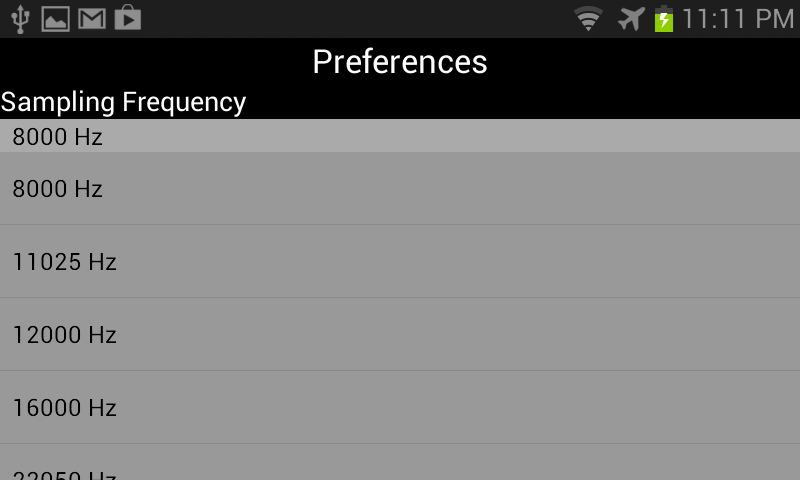


In order to access the settings menu for the APRAD app, simply press the ‘Settings’ button on the device. Consult the operating manual for the specific device on which the APRAD software is installed for the location of the ‘Settings’ button. When pressed, the ‘Settings’ pop-up menu will display with the choices of ‘Open Saved Data’ and ‘Settings’. Click the ‘Settings’ option to access the settings menu.



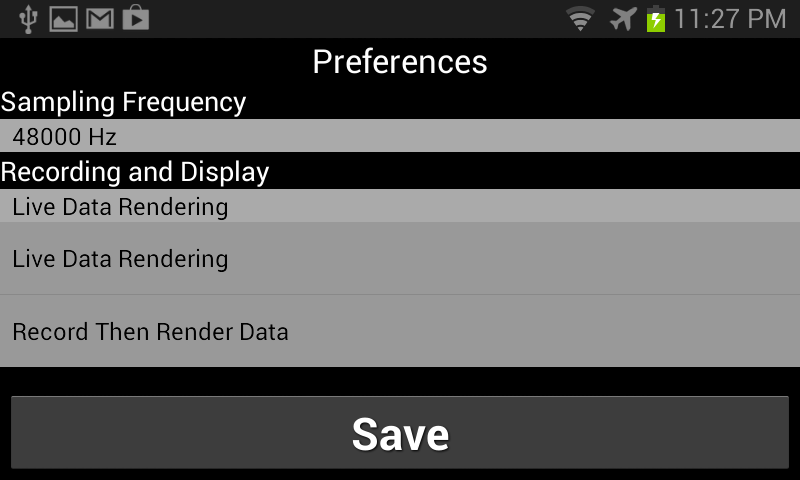
The ‘Settings’ menu enables adjustments to the APRAD software. Sampling frequency changes the rate at which samples are taken of the input signal. The higher the sampling frequency, the higher the range of frequencies the APRAD software can display in the Spectrum Analyzer and Spectrogram.

Sampling Frequency option:



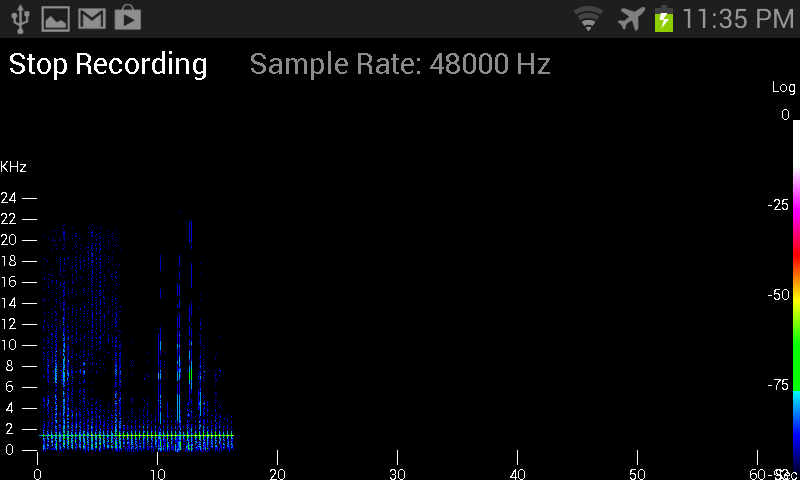
Part of the range of Sampling Frequencies is shown in the image above. The available options for sampling frequency are (all in kHz): 8, 11.025, 12, 16, 22.050, 24, 32, 44.1, and 48. The sampling frequency directly correlates to the range of frequencies displayed in the Spectrum Analyzer and Spectrogram. The displayed frequency range will be half that of the sampling frequency. For example, if a sampling frequency of 16 kHz is chosen, the displayed frequency range will be up to 8 kHz. Correspondingly, if a sampling frequency of 48 kHz is selected, the displayed frequency range will be up to 22 kHz. As a trade-off for a larger frequency range, the battery life of the operating device could be reduced, the level of fine detail will be reduced and the longer it will take to render data in the Spectrogram.

Recording and Display options:



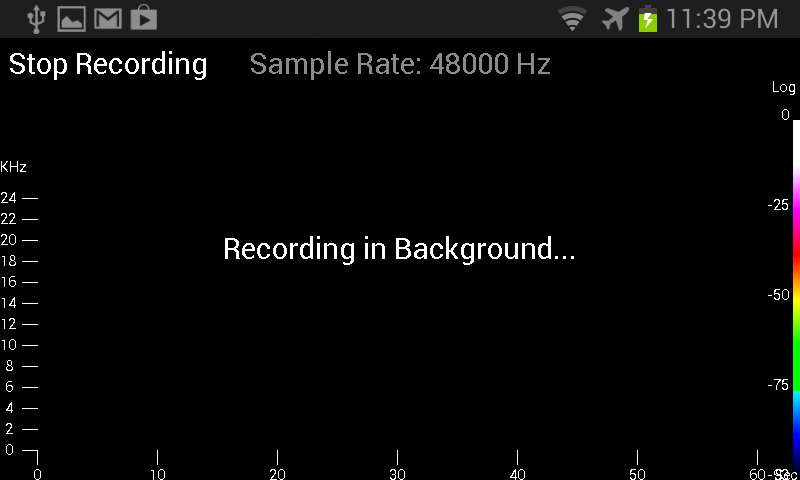
The Recording and Display options affect how data will be captured and displayed in the Spectrogram. The options available for Recording and Display are ‘Live Data Rendering’ and ‘Record then Render Data’.

When the Live Data Rendering mode is selected, the Spectrogram will display the frequency vs. time data in real time as each set of samples is being captured by the input of the device:

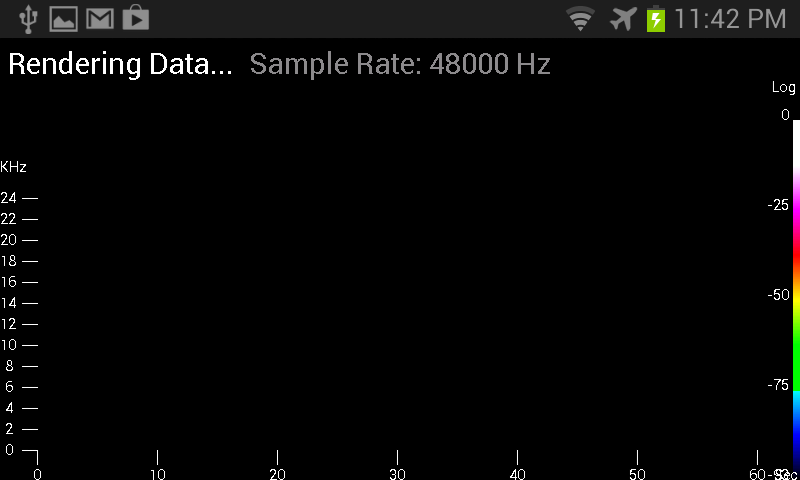


Notice the Stop Recording button at the top right of the screen and the selected sample rate. The Spectrogram will immediately display the frequency vs. time graph when in Live Data Rendering mode. This option provides immediate feedback on the captured signals and allows the user to see the results of adjustments to the antenna orientation immediately.

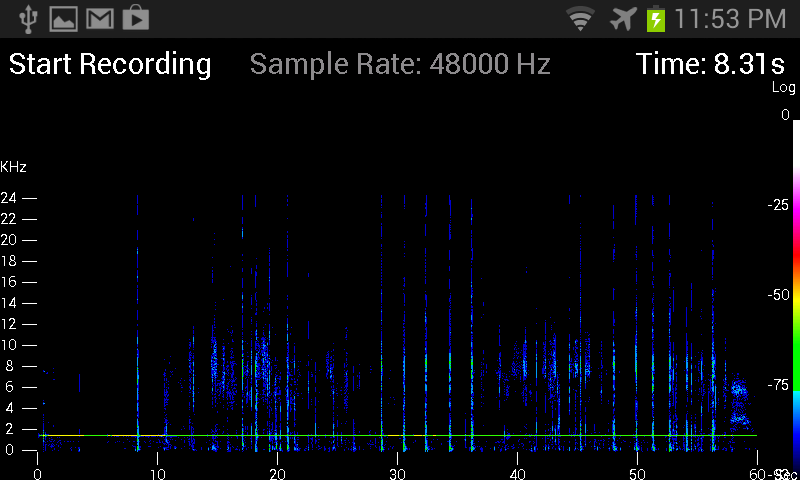
When the ‘Record Then Render Data’ mode is selected, the processor-intensive task of displaying the data is suppressed during the capture of the input signal data. This allows for the device to capture as much detail as possible and then calculate the display separately, after recording has stopped. While the signal data is being captured, the screen will indicate that the APRAD Spectrogram is recording in the background:



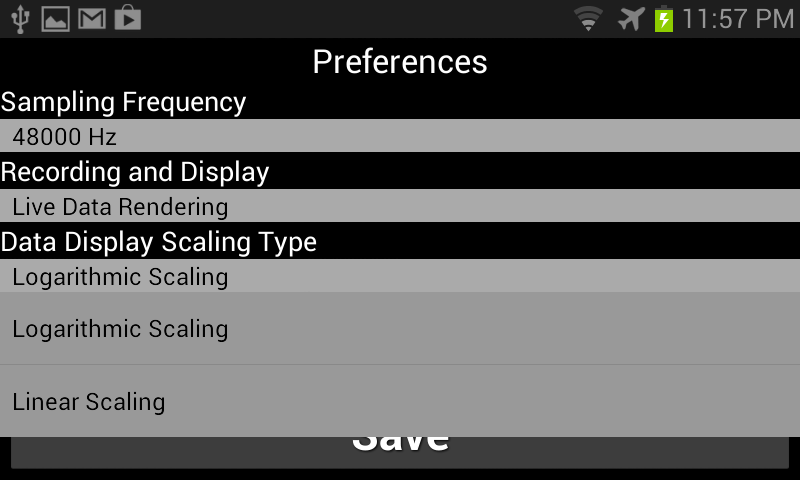
Once the capture process is completed, either by the user pressing the ‘Stop Recording’ button on the screen, or by the maximum buffer being filled (720 sample sets), the software will indicate that the Spectrogram is rendering the data to be displayed:



Once the Spectrogram has finished the processing of the input signal, the data is then displayed to the screen. Notice that the display can be much cleaner and without unwanted distortions that can appear in the ‘Live Data Rendering’ option. It may be beneficial for a user to start a survey in ‘Live’ mode to establish a possible favorable location, then switch to ‘Record Then Render Data’ mode to get cleaner data that can indicate subtle differences in the input signal not necessarily detected in ‘Live’ mode.



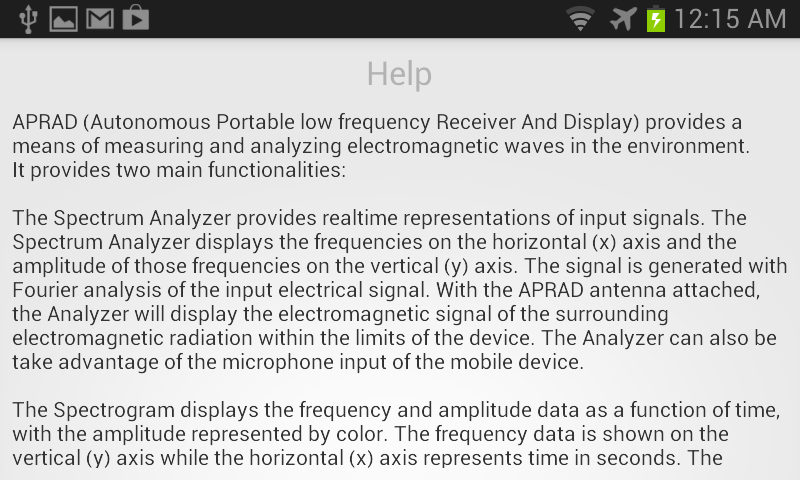
Data Display Scaling Type:



Using the Display Scaling Type option allows for setting the APRAD system into Linear or Logarithmic Scaling mode. Linear mode displays differences in the amplitude of the input signal in a linear fashion: when the input signal doubles in intensity, the display signal indicates a doubled amplitude on the graph display (for both Spectrum Analyzer and Spectrogram). By contrast, the Logarithmic Scaling mode shows the amplitude differences in a semi-log plot where the time (x-axis) is displayed in linear mode but the amplitude is displayed in log10 format. This allows for greater detail to be displayed in signals that have a much smaller intensity. While in the mode, each doubling of amplitude of the input signal results in a 6 dB change in the amplitude of the displayed data. A frequency displayed at -52 dB is actually 8 times more intense than a frequency displayed at -70 dB.

### Help:

The Help screen is a summarized version of this User Manual. It contains useful information regarding the operation of the APRAD software and general guidelines for display and capture settings.



### About:

The About screen provides copyright information for the APRAD system and software as well as general contact information.

