S.A.K-Overlay

BY: LOUIS JENKINS

What is S.A.K-Overlay?

- Stands for “Swiss-Army-Knife” Overlay.
- The original all-in-one Overlay AND Window Manager for Android.
- Simple and intuitive Window Manager
  - Multitasking
  - Dynamic UI
  - Widgets
    - Resize and Move at will
  - Snapping
- Transparent

Why did I choose this for my App?

- Like learning new things
  - Explore the UI/UX side
  - As well as the low-level backend
- Practical
  - Could use it daily, for any given task
    - Preferably gaming
  - Extremely Fun!!!
    - Anything goes!

Third Party libraries

- RxJava and RxAndroid
  - React library wrappers for Java and Android
    - Turns anything into an Observable or Observer
    - Reactor and Observer design pattern
    - Extremely efficient and elegant in design
- Mp4Parser
  - Allows me to obtain the duration of a video
  - Allows me to concatenate two or more videos

RxJava - Terminology Simplified

- Observer
  - Observes and listens for an event.
- Observable
  - The event itself.
- Subject
  - Proxy
    - Acts as both an Observer and an Observable
    - Used to pass events without being tightly coupled
    - Example
      - Event Bus
      - Broadcast Receiver
- Examples:
  - OnTouchListener
    - Touch/MotionEvent -> Observable
    - Listener/Callback -> Observer

RxJava – Processing Operators

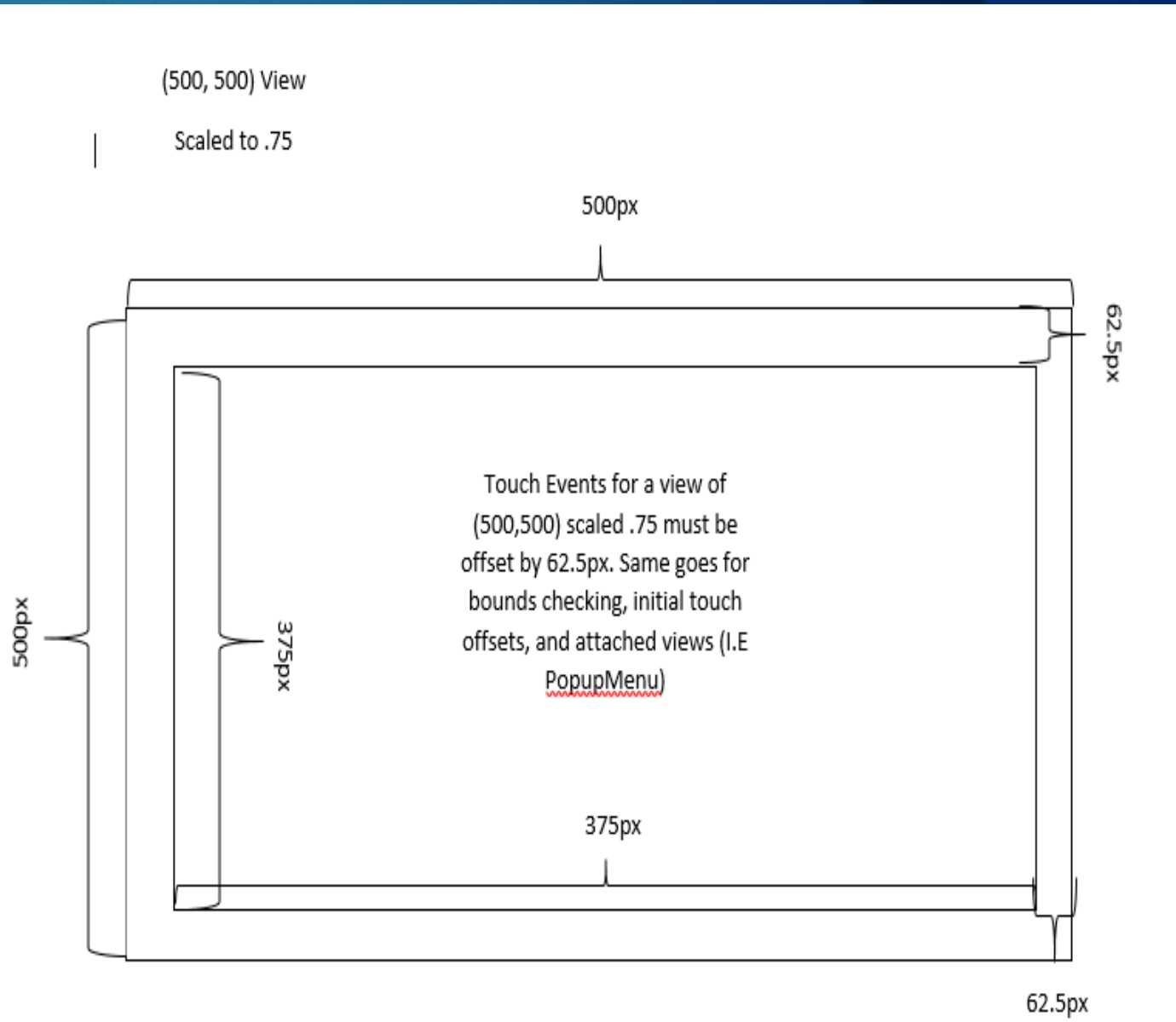
- Operators
  - Map
    - Transform one item into another
      - I.E: y = f(x); Put in X, get out Y!
  - Filter
    - Using a predicate, filter out unwanted results
      - I.E Any numbers greater than or equal to 10
  - Subscribe
    - When this event finally gets through the operators and past any filters, this gets called
      - I.E, the callback after processing is finished

RxJava – Threading Operators

- Operators
  - ObserveOn
    - The thread the end processed result is called on.
      - Via a callback subscribe()
  - SubscribeOn
    - The thread which handles all preprocessing and processing
      - Essentially whether or not to use a background thread
  - Schedulers
    - IO-Bound
      - Optimized for synchronous blocking operations
    - CPU-Bound
      - Optimized for asynchronous computational operations

What it takes to move a view

- In Android, scaled views are merely scaled within their original rect/canvas, hence the actual width and height remain the same, making interpreting touch events rather difficult… Example process…
  - Obtain initial touch offset
  - Determine if view is gesturing in a way that implies it should snap
  - Get delta of difference in actual view size and scaled view size
    - Use this along with the current touch location to determine where it should move
      - Then determine if it is in bounds
        - THEN adjust bounds of screen by delta offset
          - THEN finally you can
- Finally, you get where you can move the view.



RxJava – A More Dynamic UI

```
private void setupReactive() {
    observableFromTouch(mContentView.findViewById(R.id.title_bar_move))
            .observeOn(AndroidSchedulers.mainThread()) // The Observer, the UI Thread, waits for processed events containing the information needed to manipulate views.
            .subscribeOn(Schedulers.computation()) // The Observable's events are processed on a computational thread, which is a non I/O-Bound thread. Perfect for this.
            .map(new Func1<MotionEvent, TouchEventInfo>() { // Map transforms one item to another item. We process the MotionEvent and create an object that encapsulates straight-forward inst
                @Override
                public TouchEventInfo call(MotionEvent event) {
                    return move(event);
                }
            })
            .filter(new Func1<TouchEventInfo, Boolean>() { // Here we "filter" unwanted processed items. If it returns null, it does not have to move at all.
                @Override
                public Boolean call(TouchEventInfo info) {
                    return info != null;
                }
            })
            .subscribe(new Action1<TouchEventInfo>() { // This part is ran on the UI Thread. The MainThread does a lot less work than before, which is good.
                @Override
                public void call(TouchEventInfo info) {
                    int x = info.getX(), y = info.getY();
                    if (x != Integer.MAX_VALUE && y != Integer.MAX_VALUE) { // If X and Y are dummy values, we do not set them.
                        mContentView.setX(info.getX());
                        mContentView.setY(info.getY());
                    }
                    mSnapMask = info.getMask();
                    if (mSnapMask != 0)
                        snap(mSnapMask); // Keep in mind, that if it is 0, then no bits are set.
                }
            });
    RxView.globalLayouts(getActivity().findViewById(R.id.main_layout)).concatWith(RxView.globalLayouts(mContentView))
            .subscribe((Action1) (aVoid) -> { boundsCheck(); });
    observableFromTouch(mContentView.findViewById(R.id.resize_button))
            .observeOn(AndroidSchedulers.mainThread())
            .subscribeOn(Schedulers.computation())
            .map((Func1) (event) -> { return resize(event); })
            .filter(new Func1<Point, Boolean>() {
                @Override
                public Boolean call(Point point) {
                    return point != null;
                }
            })
            .subscribe((Action1) (point) -> {
                    mContentView.setLayoutParams(new FrameLayout.LayoutParams(point.x, point.y));
            });
}
```

RxJava + RetroLambda (Future Overhaul)

```
Lambda Version:
    onTouch
        .observeOn(AndroidSchedulers.mainThread())
        .subscribeOn(Schedulers.computation())
        .map(e -> move(e))
        .filter(p -> p != null)
        .subscribe(p -> {
            mContentView.setX(p.x);
            mContentView.setY(p.y);
            }
        );
    RxView.touches(mContentView.findViewById(R.id.resize_button))
        .observeOn(AndroidSchedulers.mainThread())
        .subscribeOn(Schedulers.computation())
        .map(e -> resize(e))
        .filter(p -> p != null)
        .subscribe(p -> mContentView.setLayoutParams(new FrameLayout.LayoutParams(p.x, p.y)
        );
```

Dynamic UI – What it takes

```
public TouchEventInfo move(MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            mContentView.bringToFront();
            touchXOffset = (prevX = (int) event.getRawX()) - (int) mContentView.getX();
            touchYOffset = (prevY = (int) event.getRawY()) - (int) mContentView.getY();
            return null;
        case MotionEvent.ACTION_MOVE:
            mSnapHint = getSnapMask(prevX, prevY, (tmpX = (int) event.getRawX()), (tmpY = (int) event.getRawY()));
            prevX = tmpX;
            prevY = tmpY;
            width = mContentView.getWidth();
            height = mContentView.getHeight();
            int scaleDiffX = MeasureTools.scaleDiffToInt(width, Globals.SCALE_X.get()) / 2;
            int scaleDiffY = MeasureTools.scaleDiffToInt(height, Globals.SCALE_Y.get()) / 2;
            int moveX = Math.min(Math.max(tmpX - touchXOffset, -scaleDiffX), Globals.MAX_X.get() - width + scaleDiffX);
            int moveY = Math.min(Math.max(tmpY - touchYOffset, -scaleDiffY), Globals.MAX_Y.get() - height + scaleDiffY);
            return new TouchEventInfo(moveX, moveY, 0);
        case MotionEvent.ACTION_UP:
            return new TouchEventInfo(Integer.MAX_VALUE, Integer.MAX_VALUE, mSnapHint);
        default:
            return null;
    }
}

public Point resize(MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            Point p = MeasureTools.getScaledCoordinates(mContentView);
            tmpX2 = p.x;
            tmpY2 = p.y;
            return null;
        case MotionEvent.ACTION_MOVE:
            int diffX = (int) event.getRawX() - tmpX2;
            int diffY = (int) event.getRawY() - tmpY2;
            int scaleDiffX = MeasureTools.scaleDiffToInt(mContentView.getWidth(), Globals.SCALE_X.get());
            int scaleDiffY = MeasureTools.scaleDiffToInt(mContentView.getHeight(), Globals.SCALE_Y.get());
            int width = Math.min(Math.max((int) (diffX / Globals.SCALE_X.get()), 250), Globals.MAX_X.get() + scaleDiffX);
            int height = Math.min(Math.max((int) (diffY / Globals.SCALE_Y.get()), 250), Globals.MAX_Y.get() + scaleDiffY);
            return new Point(width, height);
        default:
            return null;
    }
}
```

TouchEventInfo Plain Old Data

```
public class TouchEventInfo {
    private int mX, mY, mMask;

    public static final int RIGHT = 1;

    public static final int LEFT = 1 << 1;

    public static final int UPPER = 1 << 2;

    public static final int BOTTOM = 1 << 3;

    public TouchEventInfo(int x, int y, int snapMask) {
        this.mX = x;
        this.mY = y;
        this.mMask = snapMask;
    }

    public int getX() { return mX; }

    public void setX(int x) { mX = x; }

    public int getY() { return mY; }

    public void setY(int y) { mY = y; }

    public int getMask() { return mMask; }

    public void setMask(int mask) { mMask = mask; }
}
```

AeroSnap Implementation

Determine Snap

```
public int getSnapMask(int oldX, int oldY, int newX, int newY) {
    int snapMask = 0;
    int transitionX = newX - oldX;
    int transitionY = newY - oldY;
    int snapOffsetX = MeasureTools.scaleToInt(mContentView.getWidth(), Globals.SCALE_X.get()) / 10;
    int snapOffsetY = MeasureTools.scaleToInt(mContentView.getHeight(), Globals.SCALE_Y.get()) / 10;
    if (transitionX > 0 && newX + snapOffsetX >= Globals.MAX_X.get()) {
        snapMask |= TouchEventInfo.RIGHT;
    }
    if (transitionX < 0 && MeasureTools.getScaledCoordinates(mContentView).x <= snapOffsetX) {
        snapMask |= TouchEventInfo.LEFT;
    }
    if (transitionY < 0 && MeasureTools.getScaledCoordinates(mContentView).y <= snapOffsetY) {
        snapMask |= TouchEventInfo.UPPER;
    }
    if (transitionY > 0 && newY + snapOffsetY >= Globals.MAX_Y.get()) {
        snapMask |= TouchEventInfo.BOTTOM;
    }
    return snapMask;
}
```

Apply Snap

```
public void snap(int snapHint) {
    int maxWidth = getActivity().findViewById(R.id.main_layout).getWidth();
    int maxHeight = getActivity().findViewById(R.id.main_layout).getHeight();
    int width = 0, height = 0, x = 0, y = 0;
    if ((snapHint & TouchEventInfo.RIGHT) != 0) {
        width = maxWidth / 2;
        height = maxHeight;
        x = maxWidth / 2;
    }
    if ((snapHint & TouchEventInfo.LEFT) != 0) {
        width = maxWidth / 2;
        height = maxHeight;
    }
    if ((snapHint & TouchEventInfo.UPPER) != 0) {
        if (width == 0) {
            width = maxWidth;
        }
        height = maxHeight / 2;
    }
    if ((snapHint & TouchEventInfo.BOTTOM) != 0) {
        if (width == 0) {
            width = maxWidth;
        }
        height = maxHeight / 2;
        y = maxHeight / 2;
    }
    width = (int) (width / Globals.SCALE_X.get());
    height = (int) (height / Globals.SCALE_Y.get());
    x -= MeasureTools.scaleDiffToInt(width, Globals.SCALE_X.get()) / 2;
    y -= MeasureTools.scaleDiffToInt(height, Globals.SCALE_Y.get()) / 2;
    mContentView.setX(x);
    mContentView.setY(y);
    mContentView.setLayoutParams(new FrameLayout.LayoutParams(width, height));
}
```

Widgets?

- Sticky-Note
  - Allows you to record notes and/or your thoughts
- Web Browser
  - Browse the web with a minimal browser
- Google Maps
  - Allows you to keep track of where you are, and where you want to go.
- Screen Recorder
  - Record those valuable moments !

Serialization – How it works; pt.1

- BaseFloatingFragment
  - Keeps track of attributes
    - X, Y, Z, Width, Height, etc.
  - Handles movement and resizing and overall view manipulation.
  - Contains it's own custom life-cycle methods
    - Unpack()
      - Unpack any serialized data.
      - Posted to view's handler to ensure it is fully inflated.
    - Setup()
      - Setup any extra data
      - Like Unpack(), posted to content view's handler.
    - CleanUp()
      - Called when appropriate to destroy this fragment.
    - Serialize()
      - Handles serialization of data that needs to be persisted.
      - BaseClass handles View state, the subclasses override to include their own.
      - Maps each to a String-String ArrayMap.
      - Easily marshalling to JSON directly by the Key-Value pair.

Serialization – How it works; pt.2

- Deconstruction and Reconstruction
  - Handled from MainActivity
    - OnPause()
      - Serialize
    - OnCreate()
      - Deserialize
  - Uses AsyncTasks to handle background processing.
    - Each Attribute read/written from/to an ArrayMap<String, String>
  - Reconstructed from a FloatingFragmentFactory
    - By Layout Tag

Deserialization(left) and Serialization(right)

```java
@Override
protected List<ArrayMap<String, String>> doInBackground(Void... params) {
    List<ArrayMap<String, String>> mapList = new ArrayList<>();
    try {
        JsonReader reader = new JsonReader(new FileReader(file));
        reader.beginArray();
        while(reader.peek() == JsonToken.BEGIN_OBJECT){
            ArrayMap<String, String> map = new ArrayMap<>();
            reader.beginObject();
            while(reader.hasNext()){
                map.put(reader.nextName(), reader.nextString());
            }
            reader.endObject();
            mapList.add(map);
        }
        reader.endArray();
        reader.close();
    } catch (IOException e) {
        Log.e(getClass().getSimpleName(), e.getMessage());
        return null;
    }
    return mapList;
}
```

```java
@Override
protected Void doInBackground(ArrayMap<String, String>... params) {
    try {
        JsonWriter writer = new JsonWriter(new FileWriter(file));
        writer.setIndent(" ");
        writer.beginArray();
        for(ArrayMap<String, String> map : params){
            writer.beginObject();
            for(Map.Entry<String, String> entry: map.entrySet()){
                writer.name(entry.getKey()).value(entry.getValue());
            }
            writer.endObject();
        }
        writer.endArray();
        writer.flush();
        writer.close();
    } catch (IOException e) {
        Log.e(getClass().getSimpleName(), e.getMessage());
        return null;
    }
    return null;
}
```

Floating Fragments Serialization and Deserialization Implementations

- Adds each fragment not just to FragmentManager, but also maintains a weak reference list of it's own
  - Weak Referencing allows Garbage Collector to collect the FloatingFragment when it is supposed to be destroyed
    - If WeakReference.get() returns null, it has been collected and we skip on, otherwise we obtain an atomic strong reference and promptly release.
  - Prevents memory leaks

```
private List<WeakReference<FloatingFragment>> mFragments = new ArrayList<>();
private void deserializeFloatingFragments(){
    final File jsonFile = new File(getExternalFilesDir(null), JSON_FILENAME);
    if (jsonFile.exists()) {
        new FloatingFragmentDeserializer() {
            @Override
            protected void onPreExecute() {
                this.file = jsonFile;
            } // Sets the file handle.

            @Override
            protected void onPostExecute(List<ArrayMap<String, String>> mapList) {
                FloatingFragmentFactory factory = FloatingFragmentFactory.getInstance();
                FragmentTransaction transaction = getFragmentManager().beginTransaction();
                for (ArrayMap<String, String> map : mapList) {
                    FloatingFragment fragment = factory.getFragment(map);
                    mFragments.add(new WeakReference<>(fragment));
                    transaction.add(R.id.main_layout, fragment);
                }
                transaction.commit();
            }
        }.execute();
    }
}
@SuppressWarnings("unchecked")
private void serializeFloatingFragments() {
    List<ArrayMap<String, String>> mapList = new ArrayList<>();
    for (WeakReference<FloatingFragment> fragmentWeakReference : mFragments) {
        // Atomic operation, once obtained as strong reference, it is safe to dereference.
        FloatingFragment fragment = fragmentWeakReference.get();
        // A fragment is dead when it is dismissed and is still contained in this list.
        if (fragment != null && !fragment.isDead()) {
            mapList.add(fragment.serialize());
        }
    }
    (FloatingFragmentSerializer) () -> {
            this.file = new File(getExternalFilesDir(null), JSON_FILENAME);
    }.execute(mapList.toArray(new ArrayMap[0]));
}
private void addFragment(FloatingFragment fragment) {
    if(fragment == null) {
        Toast.makeText(MainActivity.this, "There can only be one instance of this widget!", Toast.LENGTH_LONG).show();
        return;
    }
    mFragments.add(new WeakReference<>(fragment));
    getFragmentManager().beginTransaction().add(R.id.main_layout, fragment).commit();
```

BaseFloatingFragment's Serialize and Unpack implementations

- Here you can see the implementation of serialize and unpack of the BaseFloatingFragment.
  - As it handles serializing the view and unpacking it, any floating fragments that do not need to bother with serialization at all do not need to override anything as it's already handled.
- Naively expects any such data to fit as a String.
  - Later, if need be, I will add complexity to handle marshalling reference types directly.

```
public ArrayMap<String, String> serialize() {
    ArrayMap<String, String> map = new ArrayMap<>();
    map.put(Globals.Keys.LAYOUT_TAG, LAYOUT_TAG);
    map.put(Globals.Keys.X_COORDINATE, Integer.toString(x));
    map.put(Globals.Keys.Y_COORDINATE, Integer.toString(y));
    map.put(Globals.Keys.WIDTH, Integer.toString(width));
    map.put(Globals.Keys.HEIGHT, Integer.toString(height));
    map.put(Globals.Keys.MINIMIZED, Boolean.toString(mContentView.getVisibility() == View.INVISIBLE));
    return map;
}

/**
 * Function called to unpack any serialized data that was originally in JSON format. This function
 * should be overriden if there is a need to unpack any extra serialized data, and the very first call
 * MUST be the super.unpack(), as this ensures that the base data gets unpacked first.
 * <p/>
 * It is safe to call getContentView() and should be used to update the view associated with this fragment.
 */
protected void unpack() {
    x = Integer.parseInt(mContext.get(Globals.Keys.X_COORDINATE));
    y = Integer.parseInt(mContext.get(Globals.Keys.Y_COORDINATE));
    width = Integer.parseInt(mContext.get(Globals.Keys.WIDTH));
    height = Integer.parseInt(mContext.get(Globals.Keys.HEIGHT));
    mContentView.setX(x);
    mContentView.setY(y);
    mContentView.setLayoutParams(new FrameLayout.LayoutParams(width, height));
    // If this is override, the subclass's unpack would be done after X,Y,Width, and Height are set.
}
```

Screen Recorder; How it works

- Note: There is a critical OS-level bug triggered by a race condition causing the FrameBuffer to deadlock
  - Nothing I can do about this
  - Only on Nexus 7 2012 edition on Lollipop (5.1.1)
  - Makes device unresponsive until reboot.
- Started from ScreenRecorderFragment
  - Bind Service to Fragment
    - Fragment can now call stop(), start() and pause()
      - Checks if it is possible in current state
        - If so, execute
  - Service starts foreground notification and creates view
    - View gets attached to WindowManager, hence drawn on top of other activities.

Screen Recorder - RecorderState

```java
public enum RecorderState {
    DEAD(1),
    STARTED(1 << 1),
    PAUSED(1 << 2),
    STOPPED(1 << 3);

    private int mMask;

    public int getMask() { return mMask; }

    /**
     * Very convenient method to get all masks at once, which allows getting all but one or two
     * super easy to do. It loops through each state then bitwise OR's them into one.
     *
     * @return All bitmasks together.
     */
    public static int getAllMask() {
        int totalMask = 0;
        for (RecorderState state : values()) {
            totalMask |= state.getMask();
        }
        return totalMask;
    }

    RecorderState(int bitmask) { mMask = bitmask; }

    @Override
    public String toString() {
        switch (this) {
            case DEAD:
                return "Dead";
            case STARTED:
                return "Recording";
            case PAUSED:
                return "Paused";
            case STOPPED:
                return "Stopped";
            default:
                return null;
        }
    }
}
```

Screen Recorder - RecorderCommands

```
public enum RecorderCommand {
    START(
            RecorderState.getAllMask() & ~RecorderState.STARTED.getMask()
    ),
    PAUSE(
            RecorderState.STARTED.getMask()
    ),
    STOP(
            RecorderState.STARTED.getMask() | RecorderState.PAUSED.getMask()
    ),
    DIE(
            RecorderState.getAllMask() & ~RecorderState.DEAD.getMask()
    );

    /**
     * Determines whether or not the command is possible by checking if the bit for the possible state
     * is set.
     *
     * @param state State to check.
     * @return True if it is a possible command for the given state.
     */
    public boolean isPossible(RecorderState state) {
        return (mPossibleStatesMask & state.getMask()) != 0;
    }

    private int mPossibleStatesMask;

    RecorderCommand(int possibleStates) { mPossibleStatesMask = possibleStates; }

    @Override
    public String toString() {
        switch (this) {
            case START:
                return "Start";
            case PAUSE:
                return "Pause";
            case STOP:
                return "Stop";
            case DIE:
                return "Die";
            default:
                return null;
        }
    }
```

Screen Recorder - Commands

Die & Stop commands

```
public void die() {
    if (!RecorderCommand.DIE.isPossible(mState)) return;
    if (mRecorder != null) {
        mRecorder.reset();
        mRecorder.release();
    }
    if (mDisplay != null) {
        mDisplay.release();
    }
    if (mProjection != null) {
        mProjection.stop();
    }
    changeState(RecorderState.DEAD);
    stopForeground(true);
    stopSelf();
}

public boolean stop() {
    if (!RecorderCommand.STOP.isPossible(mState)) return false;
    try {
        Log.i(getClass().getName(), "Stopping recorder...");
        mRecorder.stop();
        Log.i(getClass().getName(), "Resetting Screen Recorder...");
        mRecorder.reset();
        Log.i(getClass().getName(), "Releasing VirtualDisplay...");
        mDisplay.release();
        mDisplay = null;
        changeState(RecorderState.STOPPED);
        return true;
    } catch (IllegalStateException e) {
        logErrorAndChangeState(e);
        return false;
    }
}
```

Start Command

```
public boolean start(RecorderInfo info) {
    mLastRecorderInfo = info;
    int width = info.getWidth(), height = info.getHeight();
    boolean audioEnabled = info.isAudioEnabled();
    String fileName = info.getFileName();
    if (!RecorderCommand.START.isPossible(mState)) return false;
    Log.i(getClass().getName(), "Checking for permissions...");
    if (mProjection == null) {
        Log.i(getClass().getName(), "Starting activity for permission...");
        Intent intent = new Intent(this, PermissionActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        startActivity(intent);
        return false;
    }
    String errMsg;
    if (!(errMsg = checkStartParameters(width, height, fileName)).isEmpty()) {
        Toast.makeText(RecorderService.this, errMsg, Toast.LENGTH_LONG).show();
        return false;
    }
    if (!initialize(width, height, audioEnabled, fileName)) {
        return false;
    }
    try {
        Log.i(getClass().getName(), "Preparing Recorder...");
        mRecorder.prepare();
        mDisplay = createVirtualDisplay(width, height);
        Log.i(getClass().getName(), "Started!");
        mRecorder.start();
        changeState(RecorderState.STARTED);
    } catch (IOException | IllegalStateException e) {
        logErrorAndChangeState(e);
        return false;
    }
    return true;
}
```

Drawing Views over other Apps

```
private void setupFloatingView() {
    final WindowManager manager = (WindowManager) getSystemService(WINDOW_SERVICE);
    final WindowManager.LayoutParams params = new WindowManager.LayoutParams(
            WindowManager.LayoutParams.WRAP_CONTENT,
            WindowManager.LayoutParams.WRAP_CONTENT,
            WindowManager.LayoutParams.TYPE_PHONE,
            WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE,
            PixelFormat.TRANSLUCENT);
    params.gravity = Gravity.TOP | Gravity.LEFT;
    params.x = 0;
    params.y = 0;
    final ViewGroup layout = (ViewGroup) ((LayoutInflater) getSystemService(LAYOUT_INFLATER_SERVICE)).inflate(R.layout.screen_recorder_controller_view, null);
    final ImageButton controller = (ImageButton) layout.findViewById(R.id.screen_recorder_controller_button);
    final TextView stateText = (TextView) layout.findViewById(R.id.screen_recorder_controller_state);
    // Apply Listeners and Callbacks here...
    manager.addView(layout, params);
```

Future Implementations

- AppWidgetHost
  - Remembers your selected app widgets, and automatically binds them for you
    - Requires root
- AppHosting
  - Host other apps as a FloatingFragment!
    - Similar to Dual Screen feature in current versions of Android
    - Definitely requires root!
- LazyInflater
  - Inflate your own XML at runtime inside of a FloatingFragment
    - Or use our Drag and Drop tool to create one the easy way!
- Enhanced Menu Options
  - Mac OSX style Menu Options at top of screen
    - Meant to have ready by presentation
- An Actual Overlay
  - Like the Recorder Controller, have the overlay sit on top of another app, so both apps are always in the foreground.
- Gestures
  - Minimize all other windows with a shake! Restore the original state with another!
- ScreenRecorder Buffering And Streaming
  - Record your last moments, the efficient way!
    - Records the last X minutes of time in either a circular byte buffer or into a mapped byte buffer (mmap)
  - Stream your recording over a file descriptor (Easiest way to do it)

Questions?

- FAQ
  - Will I be releasing this on the App Store when it is finished?
    - Yes, as soon as majority of the bugs are fixed, and the non-root features I plan to implement are implemented, It will be released
      - Probably in about 3 – 4 months
  - How much will it cost?
    - Nothing, and it will be open source where people may contribute, but not distribute for money.
  - Will there be ads?
    - No, I hate ads, and there is no way they will fit in with my app.
  - How will I make money?
    - Donations. Hopefully.