Live Wallpaper Working Guide

Technical Overview

Requirements

MT6000

Doc No:	RM6000-R3A-TOD-V1.0EN
Version:	V1.0
Release date:	2017-02-17
Classification:	Internal

Keywords

Technical Overview

MediaTek Inc.

Postal address

No. 1, Dusing 1st Rd. , Hsinchu Science Park, Hsinchu City, Taiwan 30078

MTK support office address

No. 1, Dusing 1st Rd. , Hsinchu Science Park, Hsinchu City, Taiwan 30078

Internet

http://www.mediatek.com/

Document Revision History

Revision	Date	Author	Description
V1.0	2016-12-28	A VIjayan	Initial Release from DS6000-D2A-DMT-V1.1EN

Table of Contents

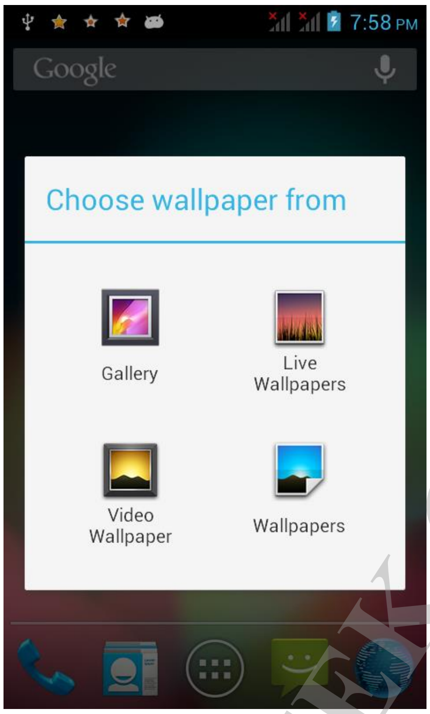
Lists of Tables

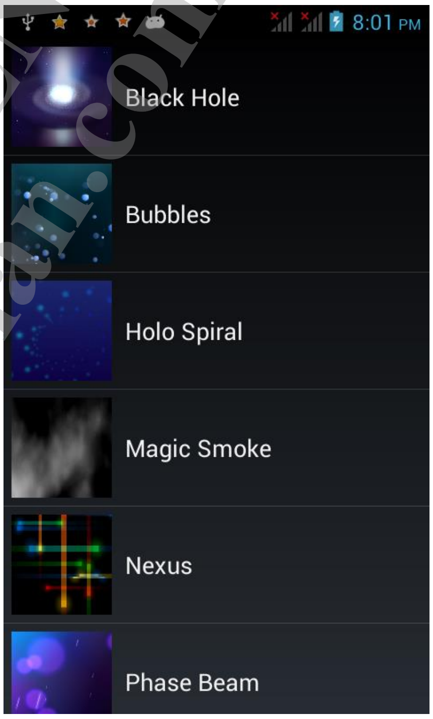
Lists of Figures

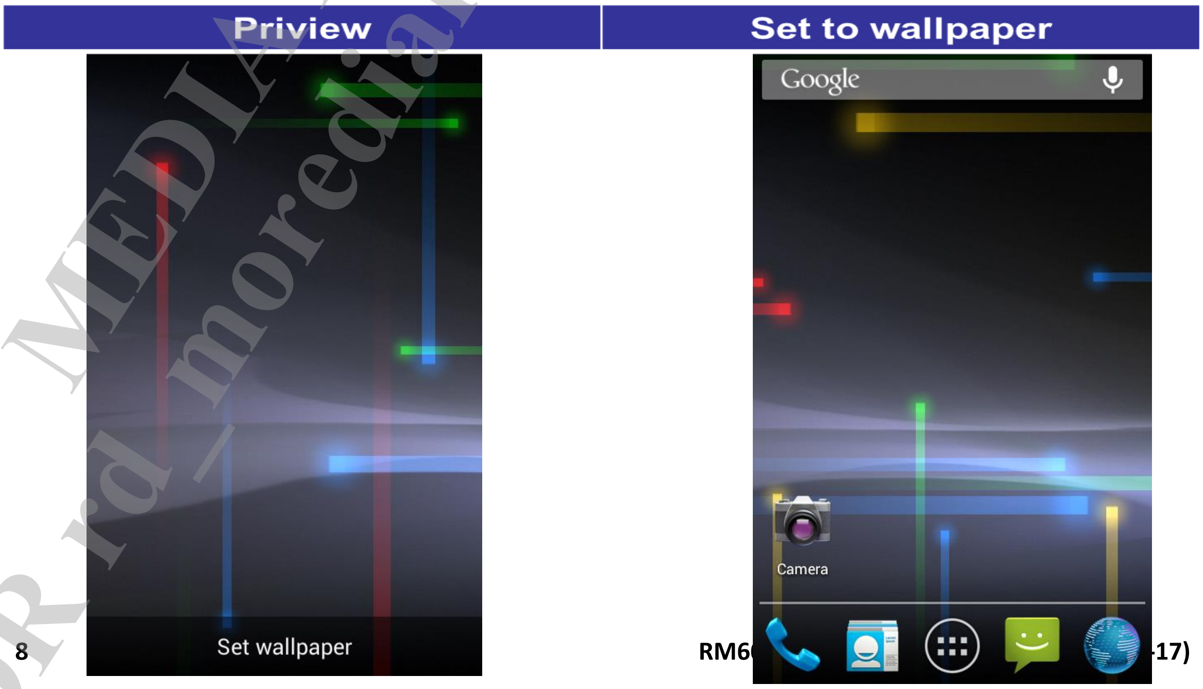
1 Introduction

Live wallpaper are the animated wallpaper which is present along with system wallpapers as an option with the user to select and apply as wallpaper.

Entering LivePicker

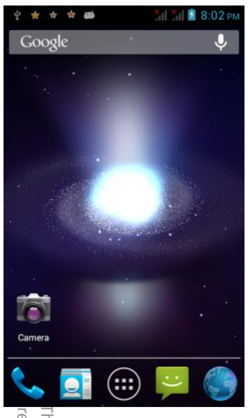
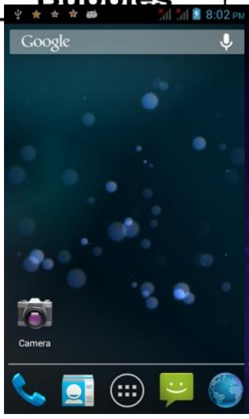
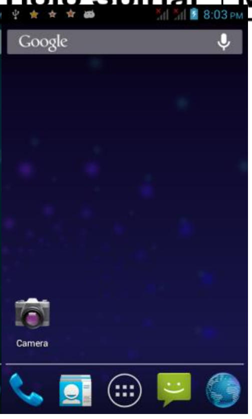
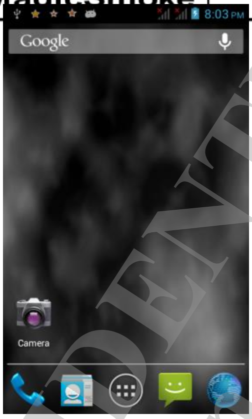
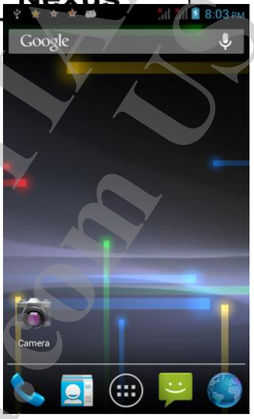




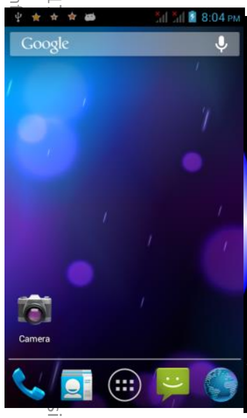
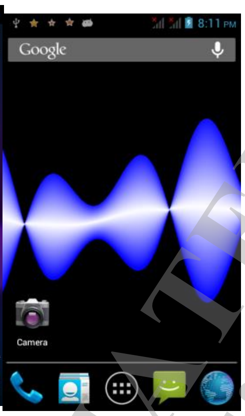
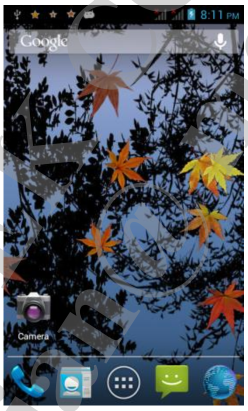


2 Overview of LiveWallpaper

Black Hole	Bubbles	Holo Spirial	MagicSmoke	Nexus

PhaseBeam	Spectrum	Water		

2.1 Purpose

Purpose of live wallpaper is to set a live image which can be guided by the user. Its ana interactive image. Provide along with static image to choose from.

- MIN_WALLPAPER_CRASH_TIME=10000
- If a wallpaper service is alive no long this duration, then it will be reverted to static wallpaper

2.2 Scope

The document provide the programming details of the Wallpaper working and manager.

2.3 Who Should Read This Document

This document is primarily intended for:

- Engineers with technical knowledge of the module or learning the module.
- Customers who integrate the live wallpaper with user-defined applications

2.4 How to Use This Manual

This segment explains how information is distributed in this document, and presents some cues and examples to simplify finding and understanding information in this document. Table 2-1 presents an overview of the chapters and appendices in this document.

Table 2-1. Chapter Overview

#	Chapter	Contents
1	Introduction	Describes the scope and layout of this document.

2.4.1 Terms and Conventions

This document uses special terms and typographical conventions to help you easily identify various information types in this document. These cues are designed to simply finding and understanding the information this document contains.

Table 2-2. Conventions

Convention	Usage	Example
[1]	Serial number of a document in the order of appearance in the References topic	Look up Chapter 2: System Architecture in [1]
void xx(zz)	Source code	static int __stdcall cb_download_bloader_init(void *usr_arg){}
☞	Important	

Convention	Usage	Example

3 Definitions

For the purposes of the present document, the following terms and definitions apply:

Enhanced Network Service Access Point Identifier (Enhanced NSAPI): integer value in the range [128; 255], identifying a certain Multimedia Broadcast/Multicast Service (MBMS) UE Context. G-PDU: is a user data message, It consists of a T-PDU plus a GTP header

GTP Tunnel: in the GTP-U plane is defined for each PDP Context or each MBMS service in the GSNs and/or each RAB in the RNC. A GTP tunnel in the GTP-C plane is defined for all PDP Contexts with the same PDN Connection (for Tunnel Management messages and UE Specific MBMS message), for each MBMS service (for Service Specific MBMS messages) or for each MS (for other types of messages). A GTP tunnel is identified in each node with a TEID, an IP address and a UDP port number. A GTP tunnel is necessary to forward packets between an external packet data network and an MS user.

☞ [Random filler text. Not intended for actual reading.] Must keep the chapter even it have empty content.

4 Abbreviations

Please note the abbreviations and their explanations provided in Table 4-1. They are used in many fundamental definitions and explanations in this document and are specific to the information that this document contains.

Table 4-1. Abbreviations

Abbreviations	Explanation
MTK	MediaTek, Asia’s largest fabless IC design company.

☞ [Random filler text. Not intended for actual reading.] Must keep the chapter even it have empty content.

5 Overview

This chapter first gives a brief description of the Live Wallpaper and its working , architecture design flow.

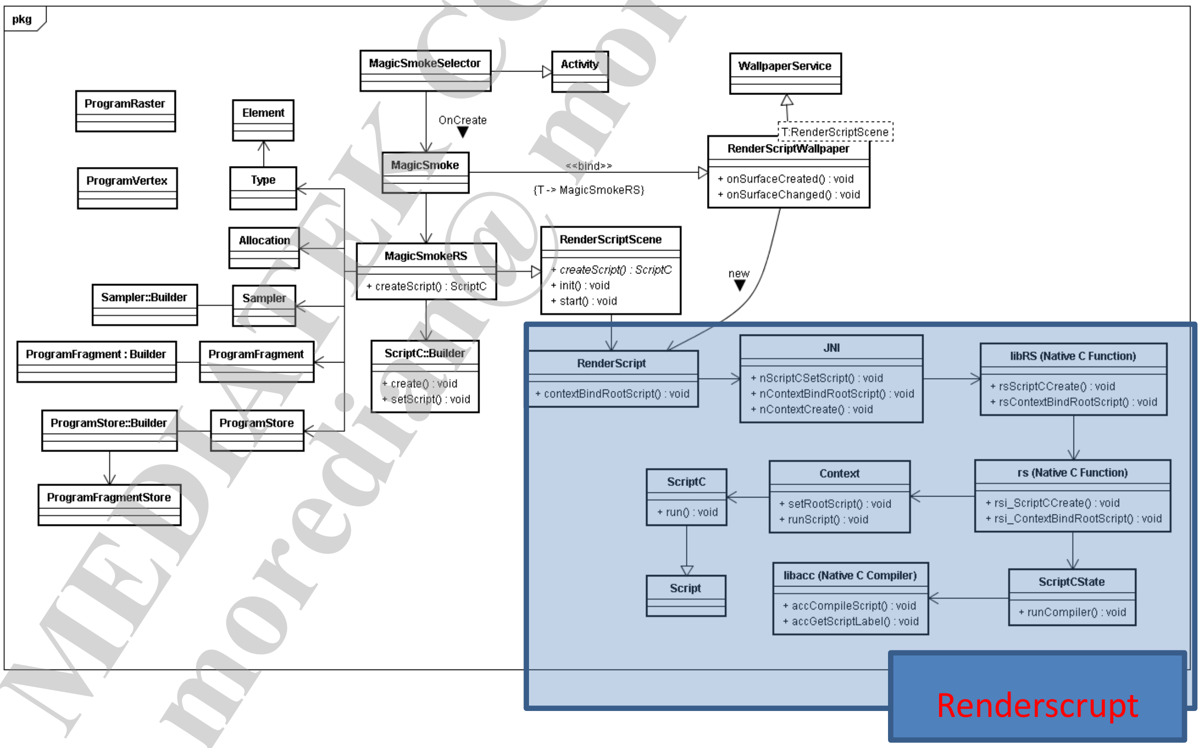
How user can set the live wlallpaper.

How various H/W component works in the feature.

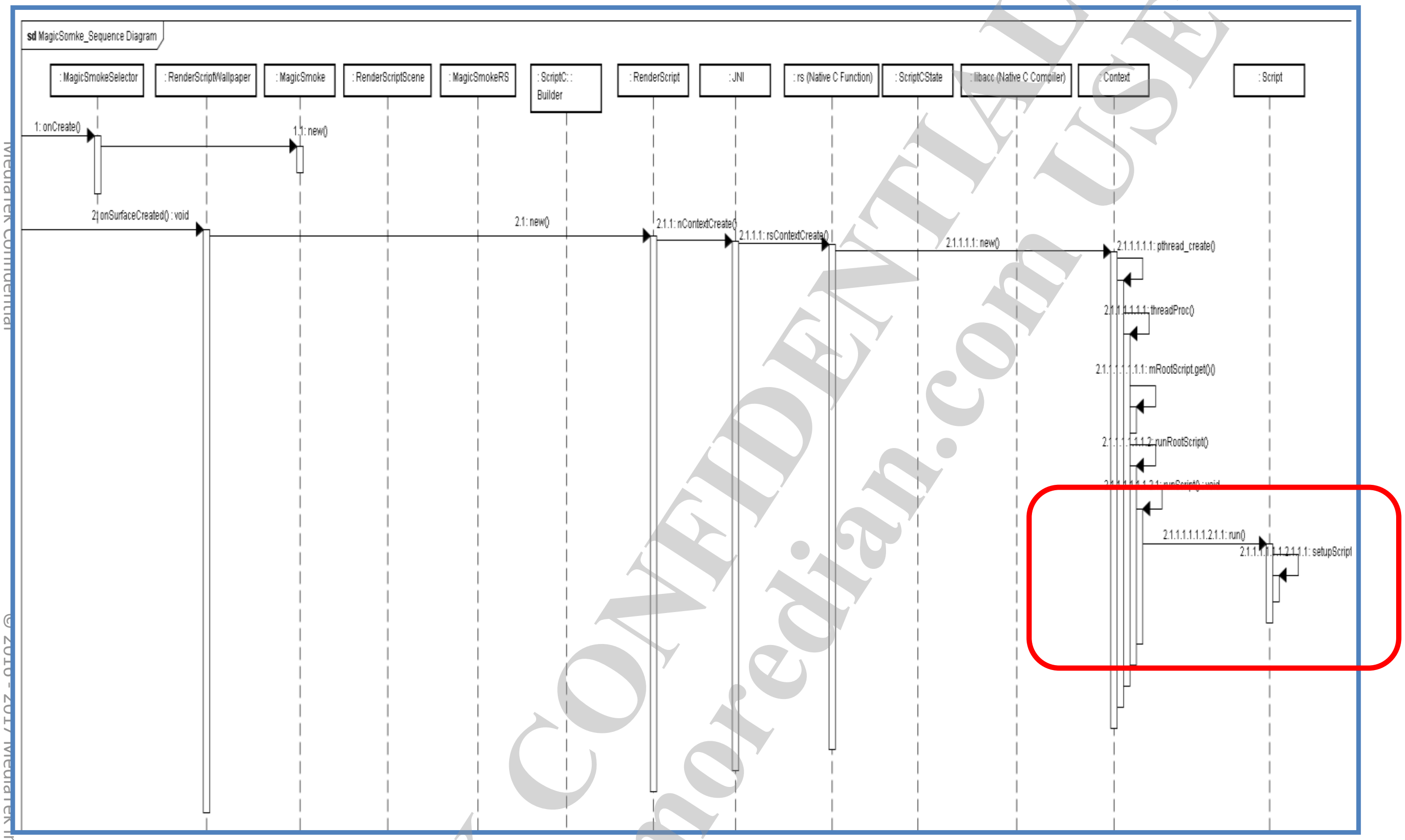
5.1 Background

There are multiple components working in coordination to fulfill the feature. The components include GPU, rederscript architecture.

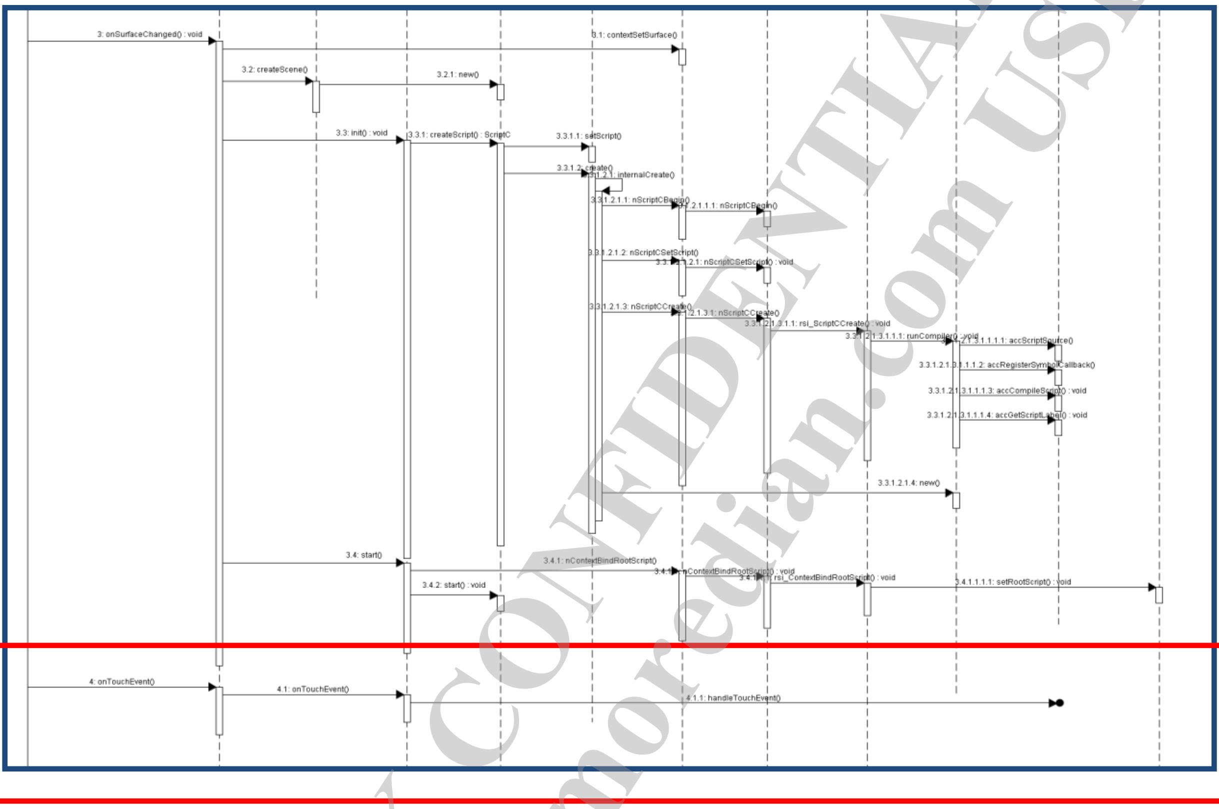
5.2 Architecture of Live Wallpaper



5.2.1 Interaction of LiveWallpapar and RS:


sd MagicSomke_Sequence Diagram
: MagicSmokeSelector
: RenderScriptWallpaper
: MagicSmoke
: RenderScriptScene
: MagicSmokeRS
: ScriptC: : Builder
: RenderScript
: JNI
: rs (Native C Function)
: ScriptCState
: libacc (Native C Compiler)
: Context
: Script
1: onCreate()
1.1: new()
2: onSurfaceCreated() : void
2.1: new()
2.1.1: nContextCreate()
2.1.1.1: rsContextCreate()
2.1.1.1.1: new()
2.1.1.1.1.1: pthread_create()
2.1.1.1.1.1.1: threadProc()
2.1.1.1.1.1.1.1: mRootScript.get()()
2.1.1.1.1.1.1.2: runRootScript()
2.1.1.1.1.1.1.2.1: runScript() : void
2.1.1.1.1.1.1.2.1.1: run()
2.1.1.1.1.1.1.2.1.1.1: setupScript

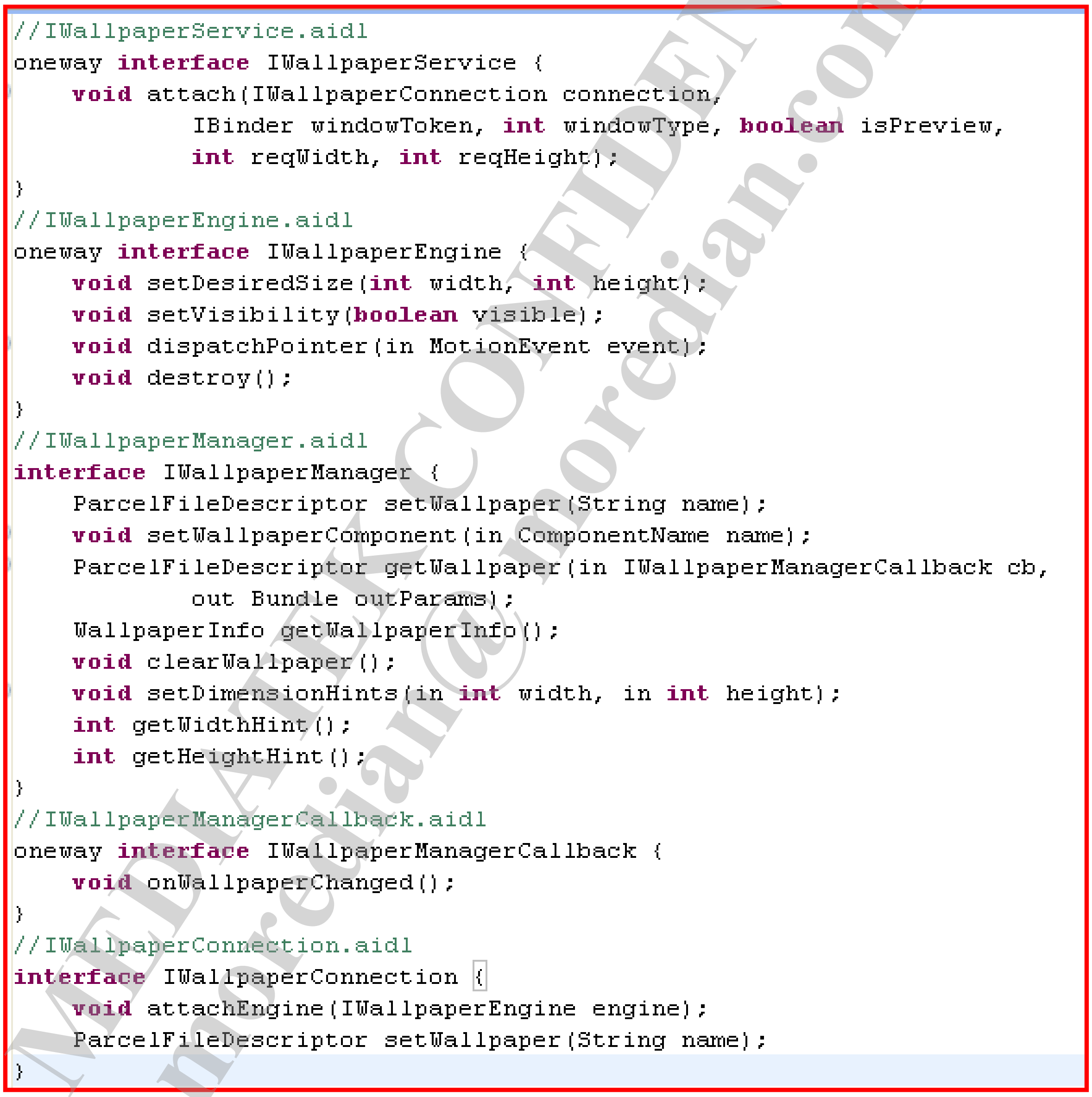
5.2.2 Interaction of LiveWallpapar and RS:


3: onSurfaceChanged() : void
3.1: contextSetSurface()
3.2: createScene()
3.2.1: new()
3.3: init() : void
3.3.1: createScript() : ScriptC
3.3.1.1: setScript()
3.3.1.2: create()
3.3.1.2.1: internalCreate()
3.3.1.2.1.1: nScriptCBegin()
3.3.1.2.1.1.1: nScriptCBegin()
3.3.1.2.1.2: nScriptCSetScript()
3.3.1.2.1.2.1: nScriptCSetScript() : void
3.3.1.2.1.3: nScriptCCreate()
3.3.1.2.1.3.1: nScriptCCreate()
3.3.1.2.1.3.1.1: rsi_ScriptCCreate() : void
3.3.1.2.1.3.1.1.1: runCompiler() : void
3.3.1.2.1.3.1.1.1.1: accScriptSource()
3.3.1.2.1.3.1.1.1.2: accRegisterSymbolCallback()
3.3.1.2.1.3.1.1.1.3: accCompileScript() : void
3.3.1.2.1.3.1.1.1.4: accGetScriptLabel() : void
3.3.1.2.1.4: new()
3.4: start()
3.4.1: nContextBindRootScript()
3.4.1.1: nContextBindRootScript() : void
3.4.1.1.1: rsi_ContextBindRootScript() : void
3.4.1.1.1.1: setRootScript() : void
3.4.2: start() : void
4: onTouchEvent()
4.1: onTouchEvent()
4.1.1: handleTouchEvent()

6 Classes and methods

6.1 Interfaces:

- **AIDL interfaces**
- **Communication between different process**



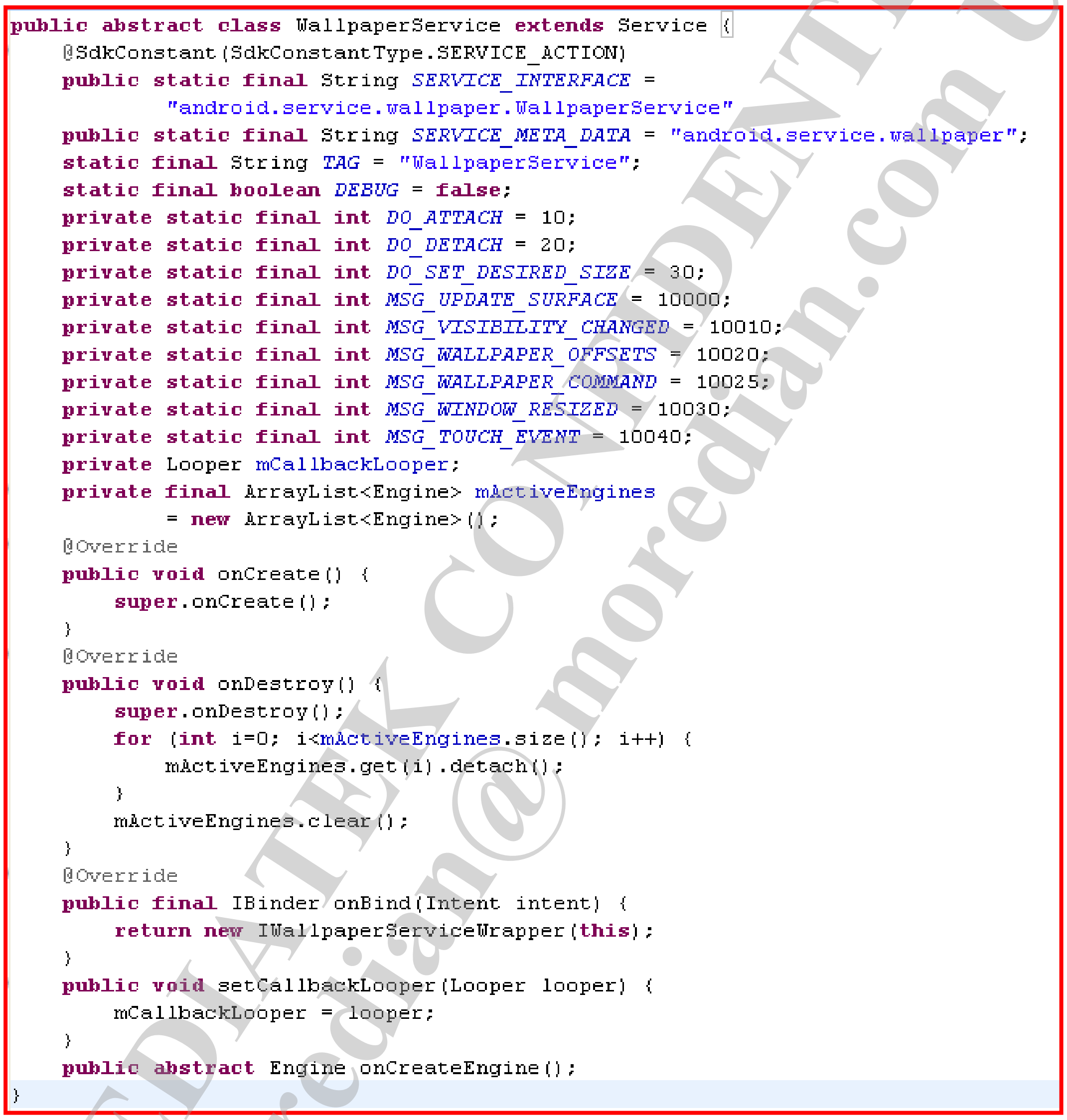
```
//IWallpaperService.aidl
oneway interface IWallpaperService {
    void attach(IWallpaperConnection connection,
            IBinder windowToken, int windowType, boolean isPreview,
            int reqWidth, int reqHeight);
}
//IWallpaperEngine.aidl
oneway interface IWallpaperEngine {
    void setDesiredSize(int width, int height);
    void setVisibility(boolean visible);
    void dispatchPointer(in MotionEvent event);
    void destroy();
}
//IWallpaperManager.aidl
interface IWallpaperManager {
    ParcelFileDescriptor setWallpaper(String name);
    void setWallpaperComponent(in ComponentName name);
    ParcelFileDescriptor getWallpaper(in IWallpaperManagerCallback cb,
            out Bundle outParams);
    WallpaperInfo getWallpaperInfo();
    void clearWallpaper();
    void setDimensionHints(in int width, in int height);
    int getWidthHint();
    int getHeightHint();
}
//IWallpaperManagerCallback.aidl
oneway interface IWallpaperManagerCallback {
    void onWallpaperChanged();
}
//IWallpaperConnection.aidl
interface IWallpaperConnection {
    void attachEngine(IWallpaperEngine engine);
    ParcelFileDescriptor setWallpaper(String name);
}
```

6.2 Class : WallpaperService

responsible for showing a live wallpaper behind applications that would like to sit on top of it.

its only purpose is to generate instances of WallpaperService.Engine as needed

Abstract class to be subclassed to implement a live wallpaper



```java
public abstract class WallpaperService extends Service {
    @SdkConstant(SdkConstantType.SERVICE_ACTION)
    public static final String SERVICE_INTERFACE =
            "android.service.wallpaper.WallpaperService";
    public static final String SERVICE_META_DATA = "android.service.wallpaper";
    static final String TAG = "WallpaperService";
    static final boolean DEBUG = false;
    private static final int DO_ATTACH = 10;
    private static final int DO_DETACH = 20;
    private static final int DO_SET_DESIRED_SIZE = 30;
    private static final int MSG_UPDATE_SURFACE = 10000;
    private static final int MSG_VISIBILITY_CHANGED = 10010;
    private static final int MSG_WALLPAPER_OFFSETS = 10020;
    private static final int MSG_WALLPAPER_COMMAND = 10025;
    private static final int MSG_WINDOW_RESIZED = 10030;
    private static final int MSG_TOUCH_EVENT = 10040;
    private Looper mCallbackLooper;
    private final ArrayList<Engine> mActiveEngines
            = new ArrayList<Engine>();
    @Override
    public void onCreate() {
        super.onCreate();
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        for (int i=0; i<mActiveEngines.size(); i++) {
            mActiveEngines.get(i).detach();
        }
        mActiveEngines.clear();
    }
    @Override
    public final IBinder onBind(Intent intent) {
        return new IWallpaperServiceWrapper(this);
    }
    public void setCallbackLooper(Looper looper) {
        mCallbackLooper = looper;
    }
    public abstract Engine onCreateEngine();
}
```

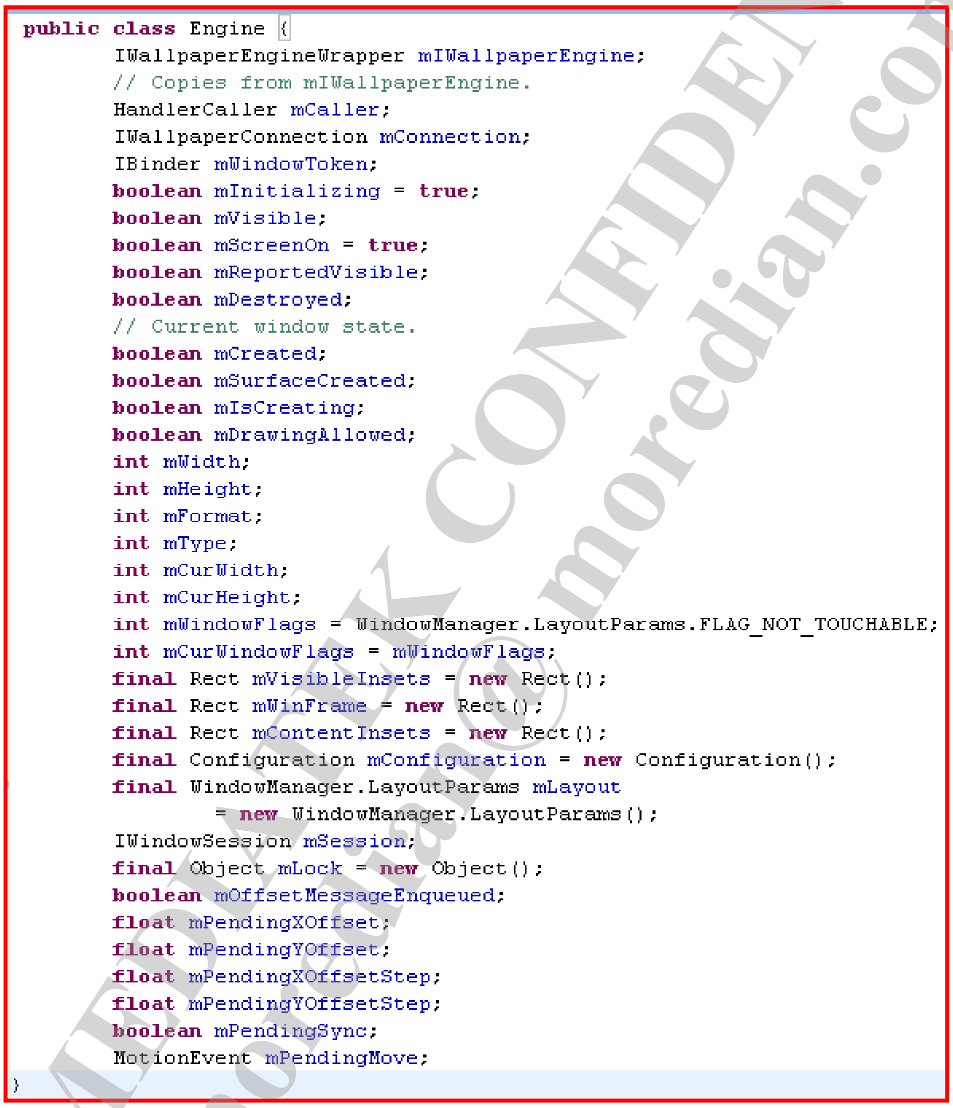
6.3 WallpaperService.Engine

The actual implementation of a wallpaper

You must implement onCreateEngine() to return your concrete Engine implementation.

What to do with it?

Message handler


```
public class Engine {
        IWallpaperEngineWrapper mIWallpaperEngine;
        // Copies from mIWallpaperEngine.
        HandlerCaller mCaller;
        IWallpaperConnection mConnection;
        IBinder mWindowToken;
        boolean mInitializing = true;
        boolean mVisible;
        boolean mScreenOn = true;
        boolean mReportedVisible;
        boolean mDestroyed;
        // Current window state.
        boolean mCreated;
        boolean mSurfaceCreated;
        boolean mIsCreating;
        boolean mDrawingAllowed;
        int mWidth;
        int mHeight;
        int mFormat;
        int mType;
        int mCurWidth;
        int mCurHeight;
        int mWindowFlags = WindowManager.LayoutParams.FLAG_NOT_TOUCHABLE;
        int mCurWindowFlags = mWindowFlags;
        final Rect mVisibleInsets = new Rect();
        final Rect mWinFrame = new Rect();
        final Rect mContentInsets = new Rect();
        final Configuration mConfiguration = new Configuration();
        final WindowManager.LayoutParams mLayout
                = new WindowManager.LayoutParams();
        IWindowSession mSession;
        final Object mLock = new Object();
        boolean mOffsetMessageEnqueued;
        float mPendingXOffset;
        float mPendingYOffset;
        float mPendingXOffsetStep;
        float mPendingYOffsetStep;
        boolean mPendingSync;
        MotionEvent mPendingMove;
}
```

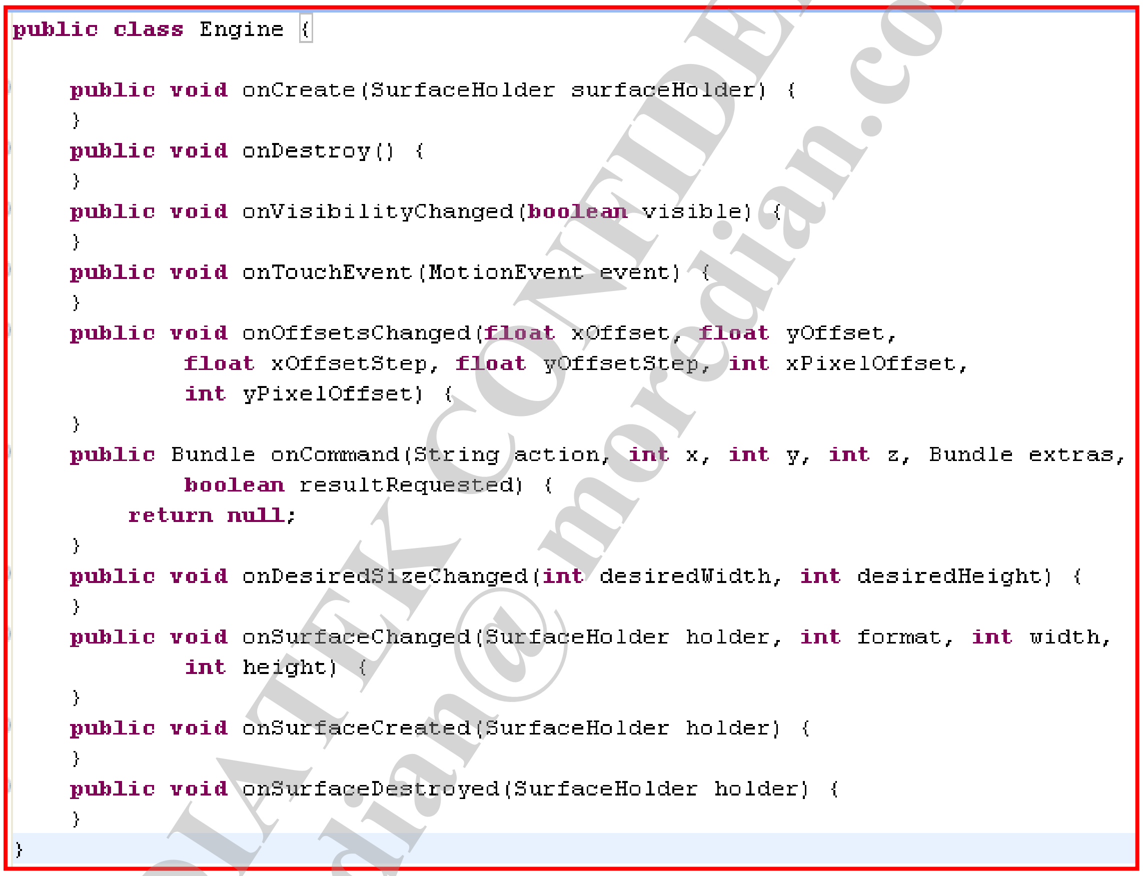
6.4 WallpaperService.Engine

Message handlers

Just for being overridden in subclass

How this be called?

Called by message handler wrapper methods described as follows:



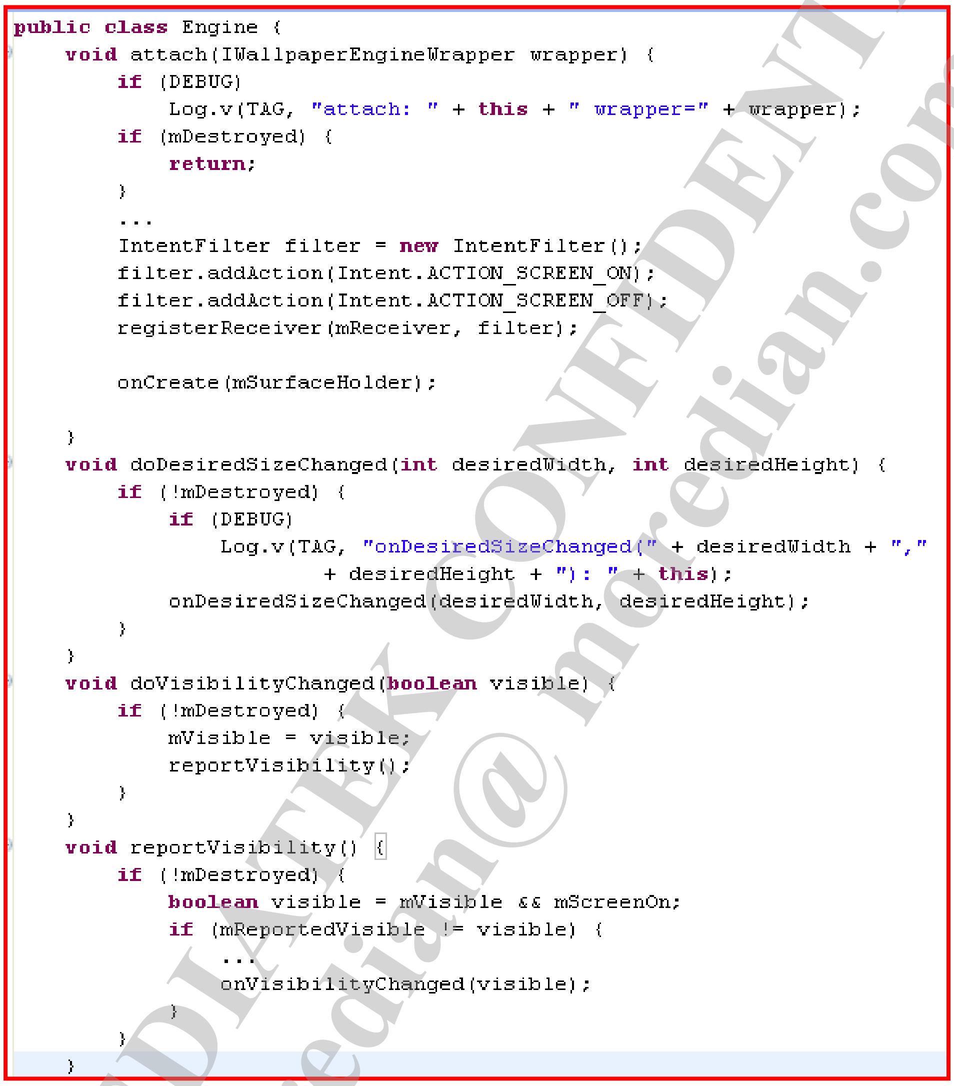
```java
public class Engine {

    public void onCreate(SurfaceHolder surfaceHolder) {
    }
    public void onDestroy() {
    }
    public void onVisibilityChanged(boolean visible) {
    }
    public void onTouchEvent(MotionEvent event) {
    }
    public void onOffsetsChanged(float xOffset, float yOffset,
            float xOffsetStep, float yOffsetStep, int xPixelOffset,
            int yPixelOffset) {
    }
    public Bundle onCommand(String action, int x, int y, int z, Bundle extras,
            boolean resultRequested) {
        return null;
    }
    public void onDesiredSizeChanged(int desiredWidth, int desiredHeight) {
    }
    public void onSurfaceChanged(SurfaceHolder holder, int format, int width,
            int height) {
    }
    public void onSurfaceCreated(SurfaceHolder holder) {
    }
    public void onSurfaceDestroyed(SurfaceHolder holder) {
    }
}
```

Message handler wrapper

Do something and then, call the message handler

Used by IWallpaperEngineWrapper

figure1



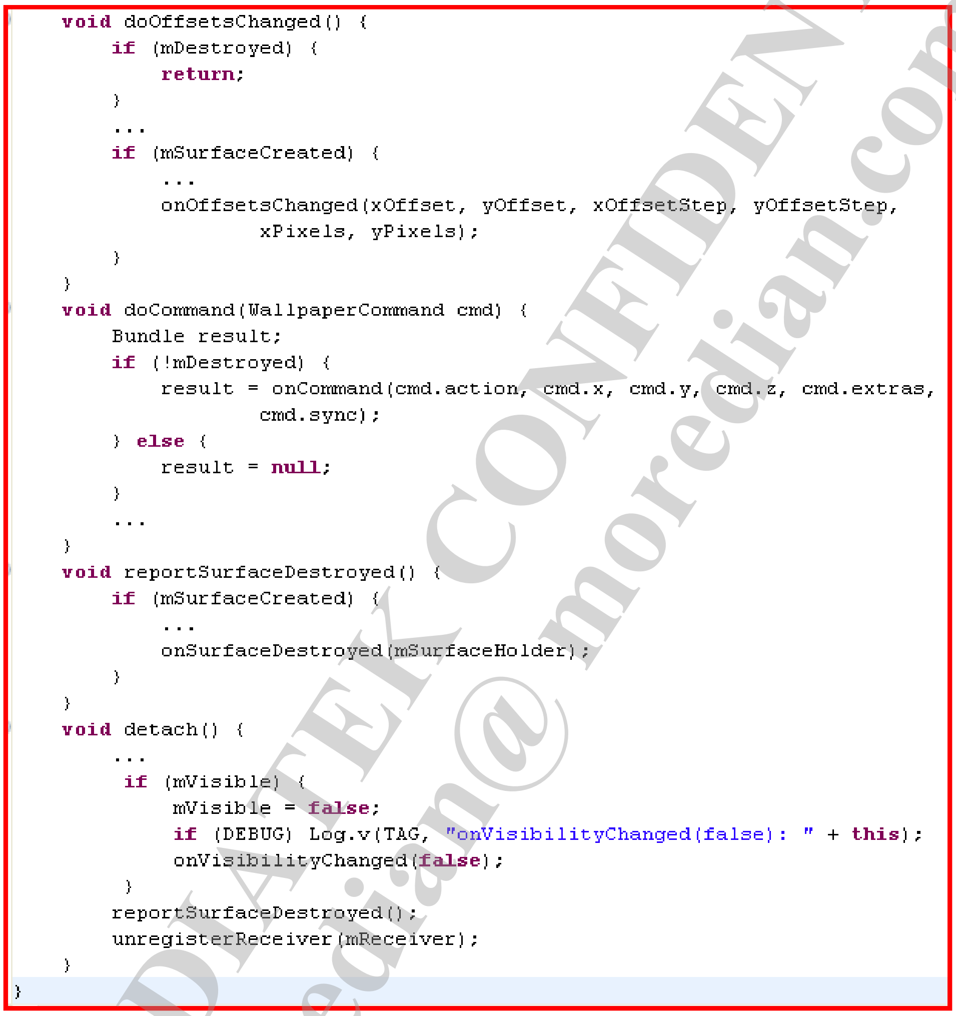
```
public class Engine {
    void attach(IWallpaperEngineWrapper wrapper) {
        if (DEBUG)
            Log.v(TAG, "attach: " + this + " wrapper=" + wrapper);
        if (mDestroyed) {
            return;
        }
        ...
        IntentFilter filter = new IntentFilter();
        filter.addAction(Intent.ACTION_SCREEN_ON);
        filter.addAction(Intent.ACTION_SCREEN_OFF);
        registerReceiver(mReceiver, filter);

        onCreate(mSurfaceHolder);

    }
    void doDesiredSizeChanged(int desiredWidth, int desiredHeight) {
        if (!mDestroyed) {
            if (DEBUG)
                Log.v(TAG, "onDesiredSizeChanged(" + desiredWidth + ","
                        + desiredHeight + "): " + this);
            onDesiredSizeChanged(desiredWidth, desiredHeight);
        }
    }
    void doVisibilityChanged(boolean visible) {
        if (!mDestroyed) {
            mVisible = visible;
            reportVisibility();
        }
    }
    void reportVisibility() {
        if (!mDestroyed) {
            boolean visible = mVisible && mScreenOn;
            if (mReportedVisible != visible) {
                ...
                onVisibilityChanged(visible);
            }
        }
    }
```

Message handler wrapper

Do something and then, call the message handler

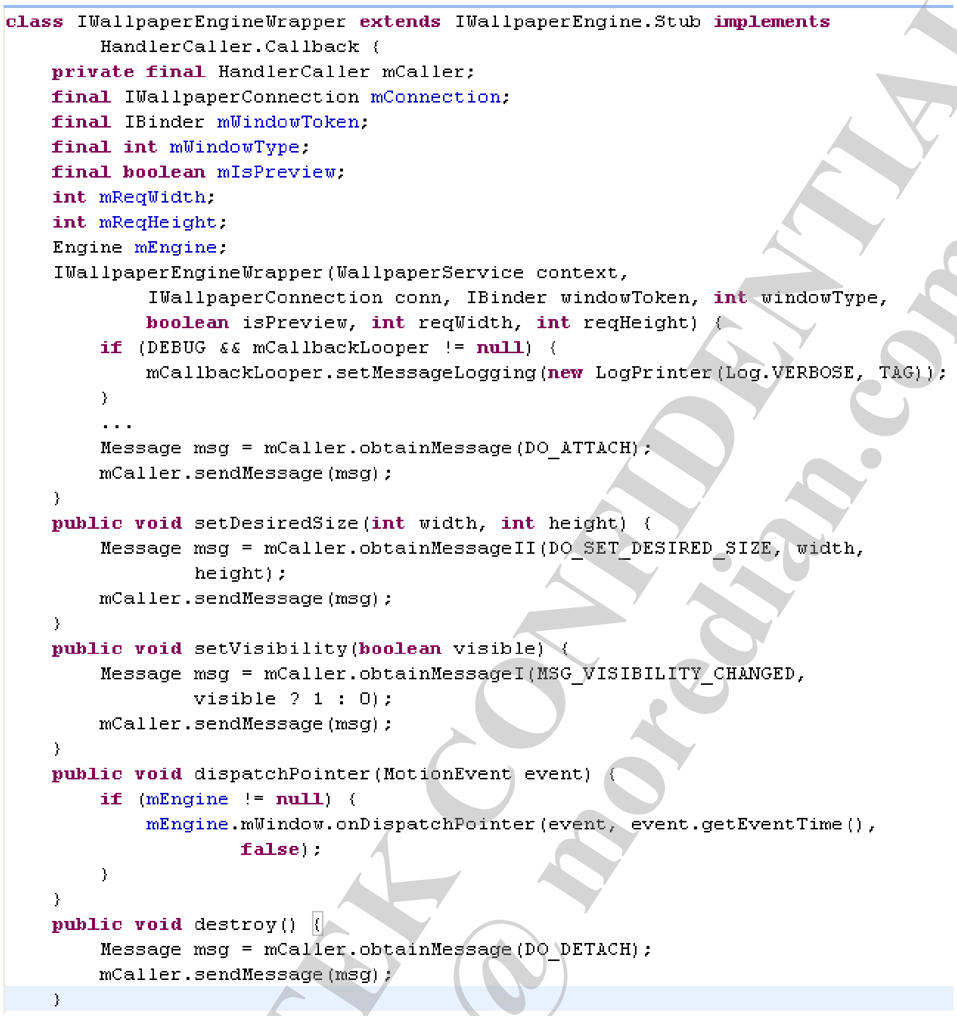
Used by IWallpaperEngineWrapper

figure2

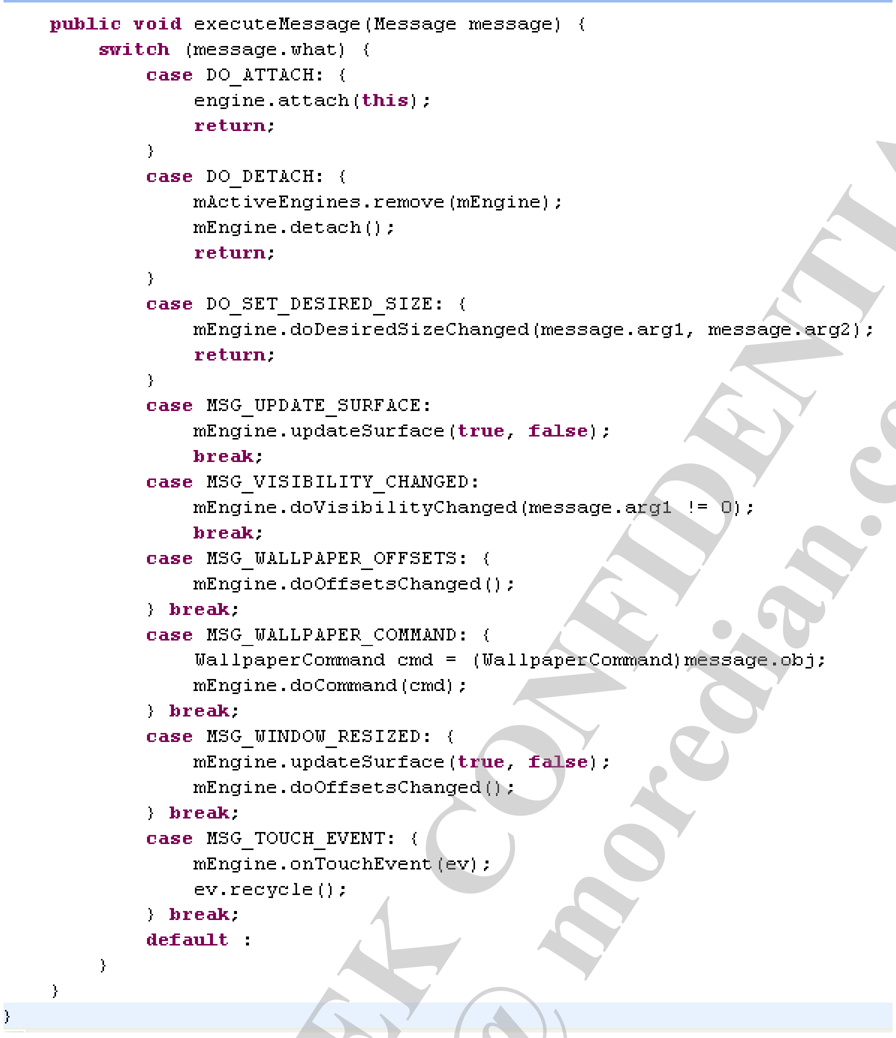


```
    void doOffsetsChanged() {
        if (mDestroyed) {
            return;
        }
        ...
        if (mSurfaceCreated) {
            ...
            onOffsetsChanged(xOffset, yOffset, xOffsetStep, yOffsetStep,
                    xPixels, yPixels);
        }
    }
    void doCommand(WallpaperCommand cmd) {
        Bundle result;
        if (!mDestroyed) {
            result = onCommand(cmd.action, cmd.x, cmd.y, cmd.z, cmd.extras,
                    cmd.sync);
        } else {
            result = null;
        }
        ...
    }
    void reportSurfaceDestroyed() {
        if (mSurfaceCreated) {
            ...
            onSurfaceDestroyed(mSurfaceHolder);
        }
    }
    void detach() {
        ...
         if (mVisible) {
             mVisible = false;
             if (DEBUG) Log.v(TAG, "onVisibilityChanged(false): " + this);
             onVisibilityChanged(false);
         }
        reportSurfaceDestroyed();
        unregisterReceiver(mReceiver);
    }
}
```

6.5 WallpaperService.IWallpaperEngineWrapper



```
class IWallpaperEngineWrapper extends IWallpaperEngine.Stub implements
        HandlerCaller.Callback {
    private final HandlerCaller mCaller;
    final IWallpaperConnection mConnection;
    final IBinder mWindowToken;
    final int mWindowType;
    final boolean mIsPreview;
    int mReqWidth;
    int mReqHeight;
    Engine mEngine;
    IWallpaperEngineWrapper(WallpaperService context,
            IWallpaperConnection conn, IBinder windowToken, int windowType,
            boolean isPreview, int reqWidth, int reqHeight) {
        if (DEBUG && mCallbackLooper != null) {
            mCallbackLooper.setMessageLogging(new LogPrinter(Log.VERBOSE, TAG));
        }
        ...
        Message msg = mCaller.obtainMessage(DO_ATTACH);
        mCaller.sendMessage(msg);
    }
    public void setDesiredSize(int width, int height) {
        Message msg = mCaller.obtainMessageII(DO_SET_DESIRED_SIZE, width,
                height);
        mCaller.sendMessage(msg);
    }
    public void setVisibility(boolean visible) {
        Message msg = mCaller.obtainMessageI(MSG_VISIBILITY_CHANGED,
                visible ? 1 : 0);
        mCaller.sendMessage(msg);
    }
    public void dispatchPointer(MotionEvent event) {
        if (mEngine != null) {
            mEngine.mWindow.onDispatchPointer(event, event.getEventTime(),
                    false);
        }
    }
    public void destroy() {
        Message msg = mCaller.obtainMessage(DO_DETACH);
        mCaller.sendMessage(msg);
    }
```



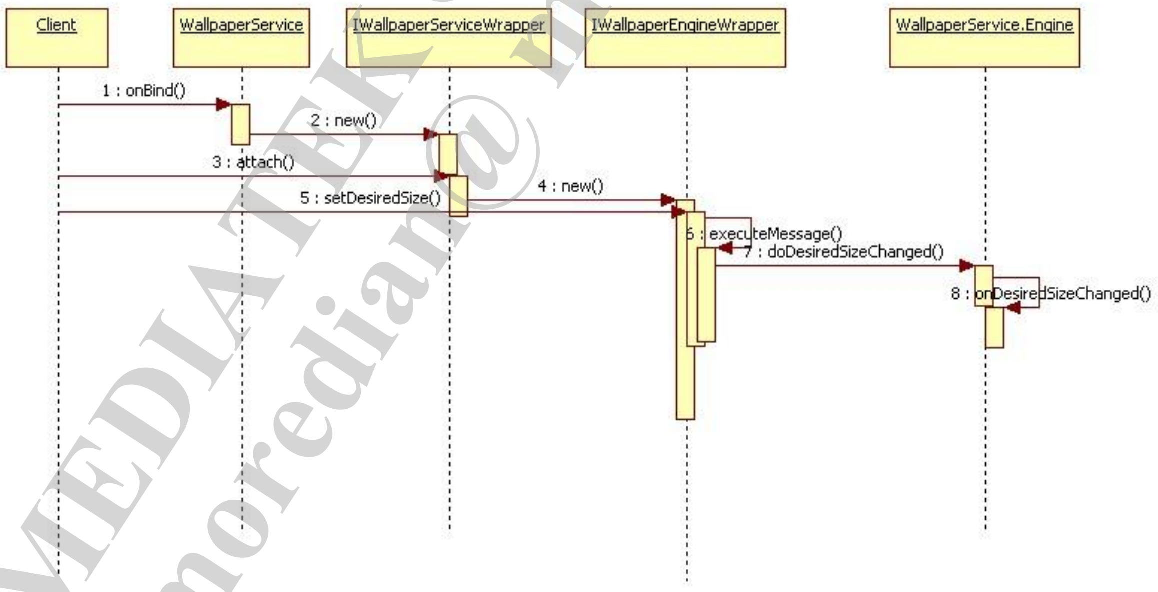
```
    public void executeMessage(Message message) {
        switch (message.what) {
            case DO_ATTACH: {
                engine.attach(this);
                return;
            }
            case DO_DETACH: {
                mActiveEngines.remove(mEngine);
                mEngine.detach();
                return;
            }
            case DO_SET_DESIRED_SIZE: {
                mEngine.doDesiredSizeChanged(message.arg1, message.arg2);
                return;
            }
            case MSG_UPDATE_SURFACE:
                mEngine.updateSurface(true, false);
                break;
            case MSG_VISIBILITY_CHANGED:
                mEngine.doVisibilityChanged(message.arg1 != 0);
                break;
            case MSG_WALLPAPER_OFFSETS: {
                mEngine.doOffsetsChanged();
            } break;
            case MSG_WALLPAPER_COMMAND: {
                WallpaperCommand cmd = (WallpaperCommand)message.obj;
                mEngine.doCommand(cmd);
            } break;
            case MSG_WINDOW_RESIZED: {
                mEngine.updateSurface(true, false);
                mEngine.doOffsetsChanged();
            } break;
            case MSG_TOUCH_EVENT: {
                mEngine.onTouchEvent(ev);
                ev.recycle();
            } break;
            default :
        }
    }
}
```

6.6 WallpaperService.IWallpaperServiceWrapper

- Implement IWallpaperService interface
- New IWallpaperEngineWrapper()

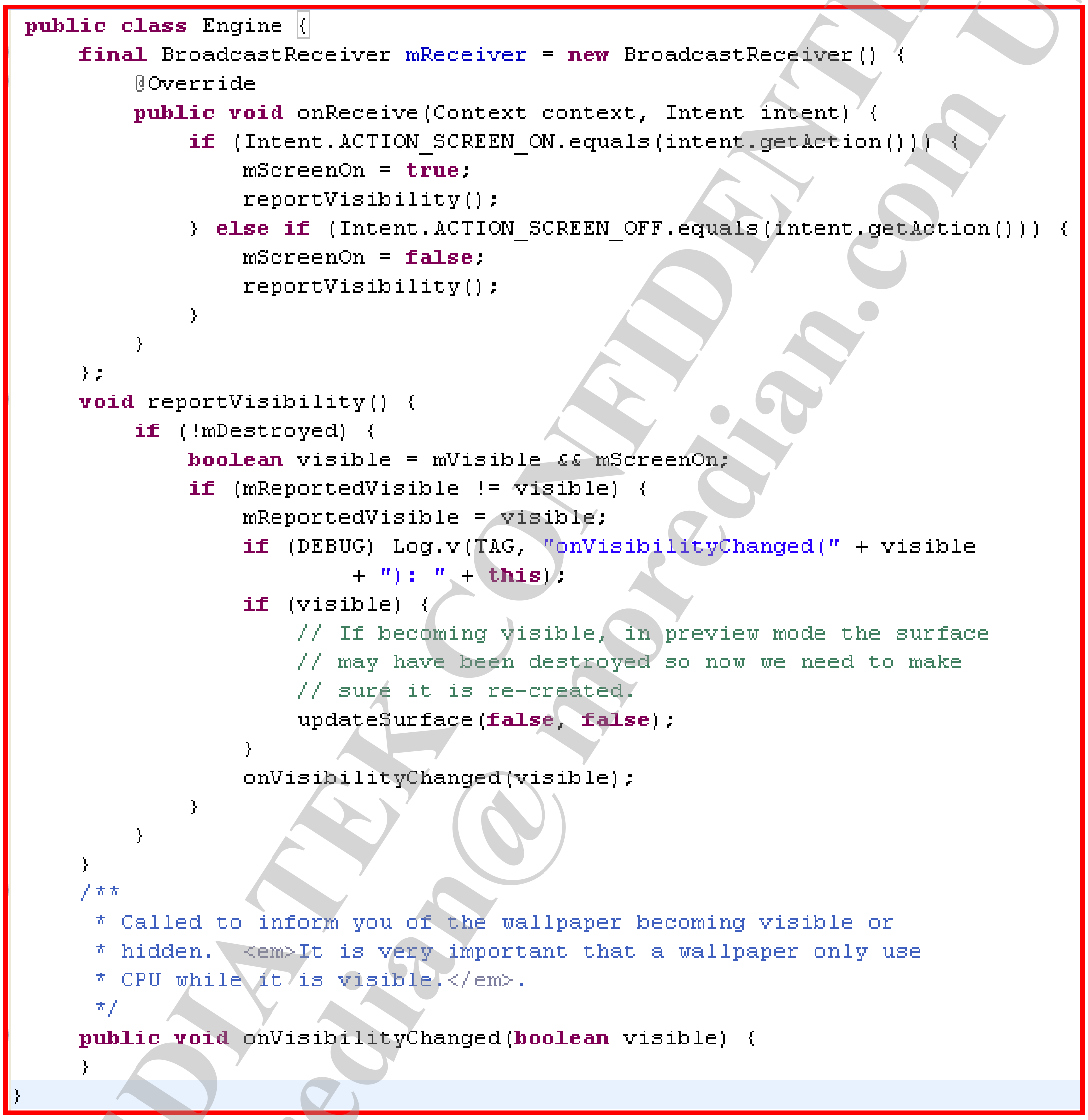
```
class IWallpaperServiceWrapper extends IWallpaperService.Stub {
        private final WallpaperService mTarget;

        public IWallpaperServiceWrapper(WallpaperService context) {
            mTarget = context;
        }
        public void attach(IWallpaperConnection conn, IBinder windowToken,
                int windowType, boolean isPreview, int reqWidth, int reqHeight) {
            new IWallpaperEngineWrapper(mTarget, conn, windowToken,
                    windowType, isPreview, reqWidth, reqHeight);
        }
    }
```

6.7 WallpaperService Sequence Diagram



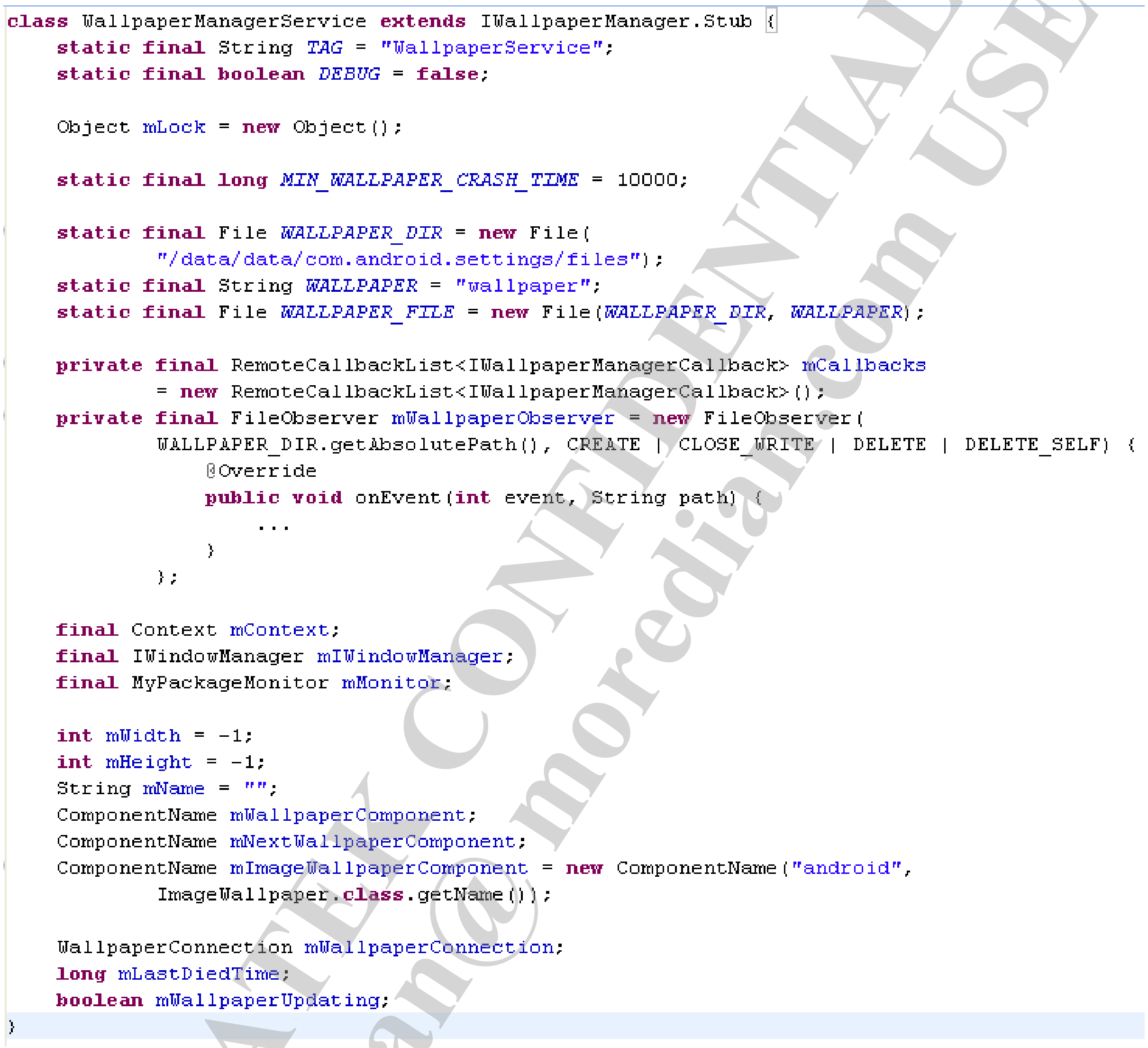
6.8 Wallpaper's visibility

Stop Live wallpaper when screen off or not visible



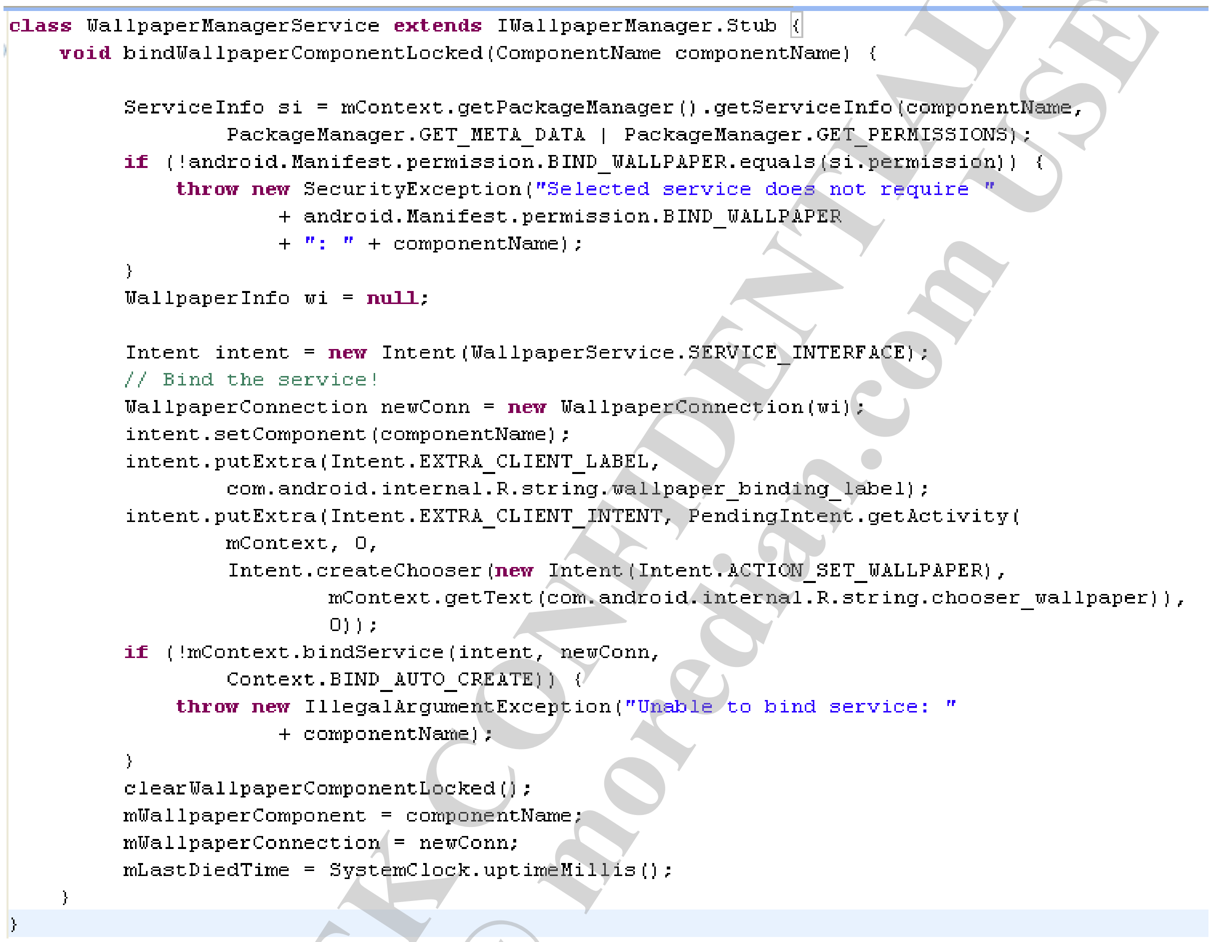
```
public class Engine {
    final BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            if (Intent.ACTION_SCREEN_ON.equals(intent.getAction())) {
                mScreenOn = true;
                reportVisibility();
            } else if (Intent.ACTION_SCREEN_OFF.equals(intent.getAction())) {
                mScreenOn = false;
                reportVisibility();
            }
        }
    };
    void reportVisibility() {
        if (!mDestroyed) {
            boolean visible = mVisible && mScreenOn;
            if (mReportedVisible != visible) {
                mReportedVisible = visible;
                if (DEBUG) Log.v(TAG, "onVisibilityChanged(" + visible
                        + "): " + this);
                if (visible) {
                    // If becoming visible, in preview mode the surface
                    // may have been destroyed so now we need to make
                    // sure it is re-created.
                    updateSurface(false, false);
                }
                onVisibilityChanged(visible);
            }
        }
    }
    /**
     * Called to inform you of the wallpaper becoming visible or
     * hidden. <em>It is very important that a wallpaper only use
     * CPU while it is visible.</em>.
     */
    public void onVisibilityChanged(boolean visible) {
    }
}
```

6.9 WallpaperManagerService:



```
class WallpaperManagerService extends IWallpaperManager.Stub {
    static final String TAG = "WallpaperService";
    static final boolean DEBUG = false;

    Object mLock = new Object();

    static final long MIN_WALLPAPER_CRASH_TIME = 10000;

    static final File WALLPAPER_DIR = new File(
            "/data/data/com.android.settings/files");
    static final String WALLPAPER = "wallpaper";
    static final File WALLPAPER_FILE = new File(WALLPAPER_DIR, WALLPAPER);

    private final RemoteCallbackList<IWallpaperManagerCallback> mCallbacks
            = new RemoteCallbackList<IWallpaperManagerCallback>();
    private final FileObserver mWallpaperObserver = new FileObserver(
            WALLPAPER_DIR.getAbsolutePath(), CREATE | CLOSE_WRITE | DELETE | DELETE_SELF) {
                @Override
                public void onEvent(int event, String path) {
                    ...
                }
            };

    final Context mContext;
    final IWindowManager mIWindowManager;
    final MyPackageMonitor mMonitor;

    int mWidth = -1;
    int mHeight = -1;
    String mName = "";
    ComponentName mWallpaperComponent;
    ComponentName mNextWallpaperComponent;
    ComponentName mImageWallpaperComponent = new ComponentName("android",
            ImageWallpaper.class.getName());

    WallpaperConnection mWallpaperConnection;
    long mLastDiedTime;
    boolean mWallpaperUpdating;
}
```

6.10 WallpaperManagerService



```
class WallpaperManagerService extends IWallpaperManager.Stub {
    void bindWallpaperComponentLocked(ComponentName componentName) {

        ServiceInfo si = mContext.getPackageManager().getServiceInfo(componentName,
                PackageManager.GET_META_DATA | PackageManager.GET_PERMISSIONS);
        if (!android.Manifest.permission.BIND_WALLPAPER.equals(si.permission)) {
            throw new SecurityException("Selected service does not require "
                    + android.Manifest.permission.BIND_WALLPAPER
                    + ": " + componentName);
        }
        WallpaperInfo wi = null;

        Intent intent = new Intent(WallpaperService.SERVICE_INTERFACE);
        // Bind the service!
        WallpaperConnection newConn = new WallpaperConnection(wi);
        intent.setComponent(componentName);
        intent.putExtra(Intent.EXTRA_CLIENT_LABEL,
                com.android.internal.R.string.wallpaper_binding_label);
        intent.putExtra(Intent.EXTRA_CLIENT_INTENT, PendingIntent.getActivity(
                mContext, 0,
                Intent.createChooser(new Intent(Intent.ACTION_SET_WALLPAPER),
                        mContext.getText(com.android.internal.R.string.chooser_wallpaper)),
                        0));
        if (!mContext.bindService(intent, newConn,
                Context.BIND_AUTO_CREATE)) {
            throw new IllegalArgumentException("Unable to bind service: "
                    + componentName);
        }
        clearWallpaperComponentLocked();
        mWallpaperComponent = componentName;
        mWallpaperConnection = newConn;
        mLastDiedTime = SystemClock.uptimeMillis();
    }
}
```

```
class WallpaperConnection extends IWallpaperConnection.Stub implements ServiceConnection {
       final WallpaperInfo mInfo;
       final Binder mToken = new Binder();
       IWallpaperService mService;
       IWallpaperEngine mEngine;
       public WallpaperConnection(WallpaperInfo info) {
           mInfo = info;
       }
       public void onServiceConnected(ComponentName name, IBinder service) {
           synchronized (mLock) {
               if (mWallpaperConnection == this) {
                   mLastDiedTime = SystemClock.uptimeMillis();
                   mService = IWallpaperService.Stub.asInterface(service);
                   attachServiceLocked(this);
                   saveSettingsLocked();
               }
           }
       }
       public void onServiceDisconnected(ComponentName name) {
           synchronized (mLock) {
               mService = null;
               mEngine = null;
               if (mWallpaperConnection == this) {
                   if (!mWallpaperUpdating && (mLastDiedTime+MIN_WALLPAPER_CRASH_TIME)
                               > SystemClock.uptimeMillis()) {
                       bindWallpaperComponentLocked(null);
                   }
               }
           }
       }
       public void attachEngine(IWallpaperEngine engine) {
           mEngine = engine;
       }
       public ParcelFileDescriptor setWallpaper(String name) {
           synchronized (mLock) {
               if (mWallpaperConnection == this) {
                   return updateWallpaperBitmapLocked(name);
               }
               return null;
           }
```

6.11 WallpaperManager

Provides access to the system wallpaper.

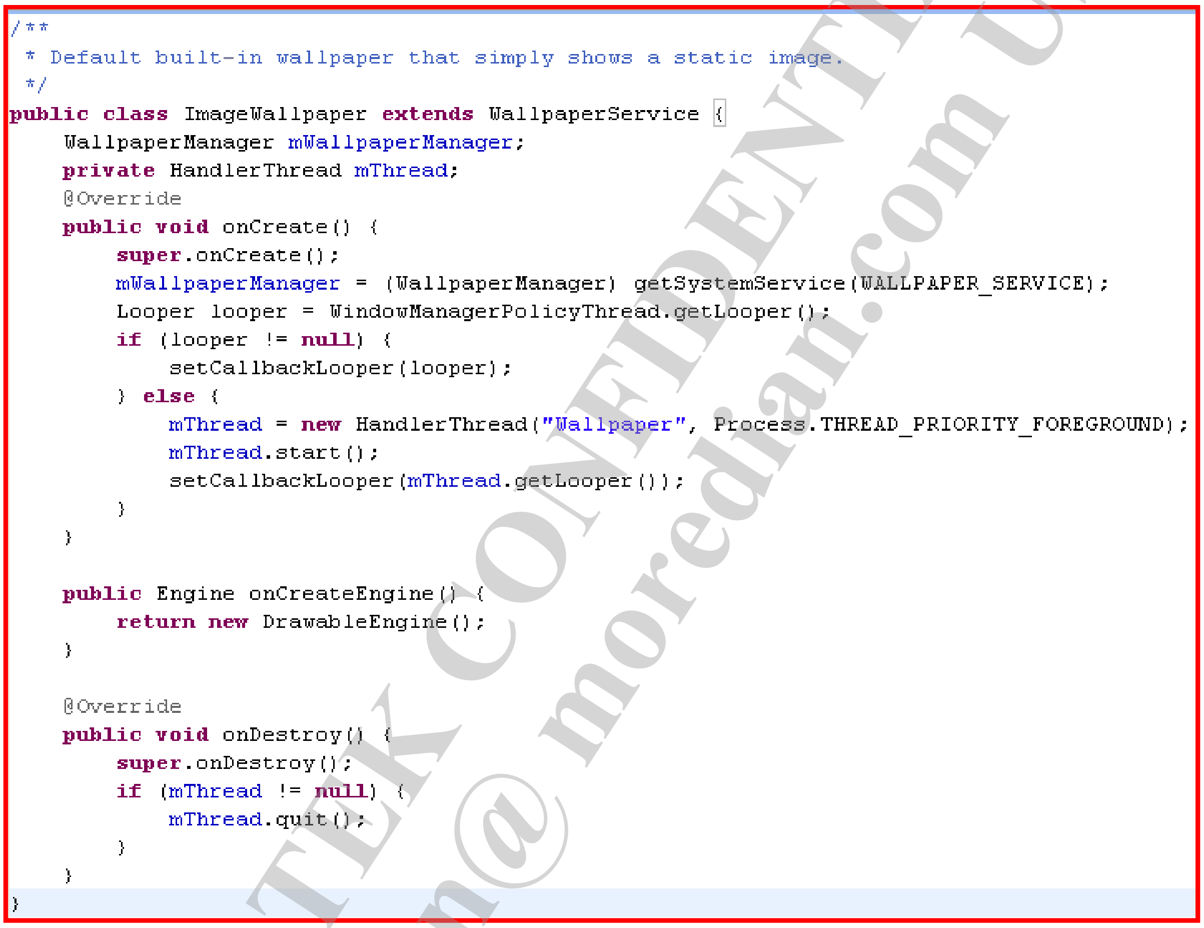
get the current wallpaper, get the desired dimensions for the wallpaper, set the wallpaper

```
 * Provides access to the system wallpaper. With WallpaperManager, you can
 * get the current wallpaper, get the desired dimensions for the wallpaper, set
 * the wallpaper, and more. Get an instance of WallpaperManager with
 * {@link #getInstance(android.content.Context) getInstance()}.
 */
public class WallpaperManager {
    static class Globals extends IWallpaperManagerCallback.Stub {
        private IWallpaperManager mService;
        private Bitmap mWallpaper;
        private Bitmap mDefaultWallpaper;

        private static final int MSG_CLEAR_WALLPAPER = 1;
        private final Handler mHandler;

        Globals(Looper looper) {
            IBinder b = ServiceManager.getService(Context.WALLPAPER_SERVICE);
            mService = IWallpaperManager.Stub.asInterface(b);
            mHandler = new Handler(looper) {
                @Override
                public void handleMessage(Message msg) {
                    switch (msg.what) {
                        case MSG_CLEAR_WALLPAPER:
                            synchronized (this) {
                                mWallpaper = null;
                                mDefaultWallpaper = null;
                            }
                            break;
                    }
                }
            };
        }
        public void onWallpaperChanged() {
            /* The wallpaper has changed but we shouldn't eagerly load the
             * wallpaper as that would be inefficient. Reset the cached wallpaper
             * to null so if the user requests the wallpaper again then we'll
             * fetch it.
             */
            mHandler.sendEmptyMessage(MSG_CLEAR_WALLPAPER);
        }
}
```

6.12 ImageWallpaper:

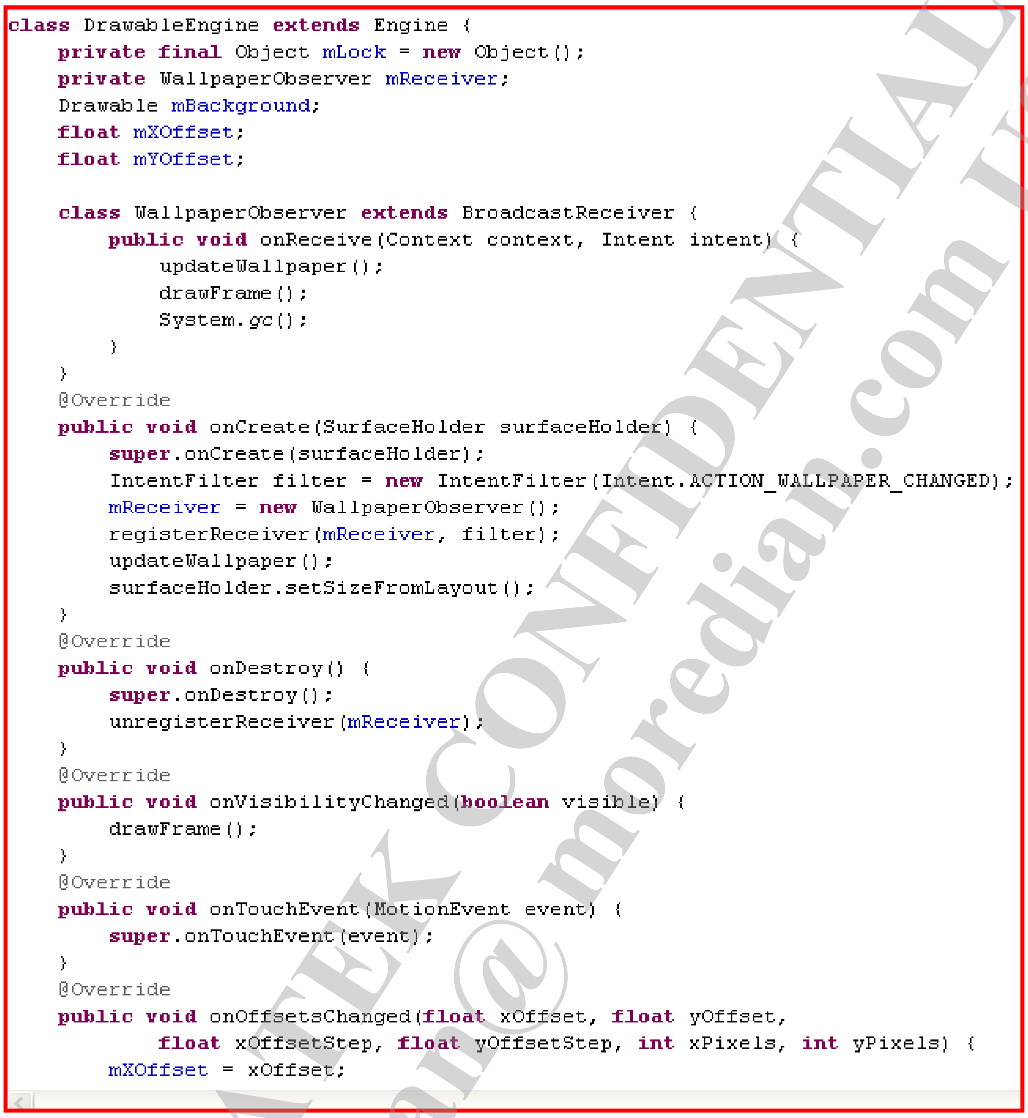
com.android.internal.services

Default built-in wallpaper



```
/**
 * Default built-in wallpaper that simply shows a static image.
 */
public class ImageWallpaper extends WallpaperService {
    WallpaperManager mWallpaperManager;
    private HandlerThread mThread;
    @Override
    public void onCreate() {
        super.onCreate();
        mWallpaperManager = (WallpaperManager) getSystemService(WALLPAPER_SERVICE);
        Looper looper = WindowManagerPolicyThread.getLooper();
        if (looper != null) {
            setCallbackLooper(looper);
        } else {
            mThread = new HandlerThread("Wallpaper", Process.THREAD_PRIORITY_FOREGROUND);
            mThread.start();
            setCallbackLooper(mThread.getLooper());
        }
    }

    public Engine onCreateEngine() {
        return new DrawableEngine();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if (mThread != null) {
            mThread.quit();
        }
    }
}
```

6.13 ImageWallpaper



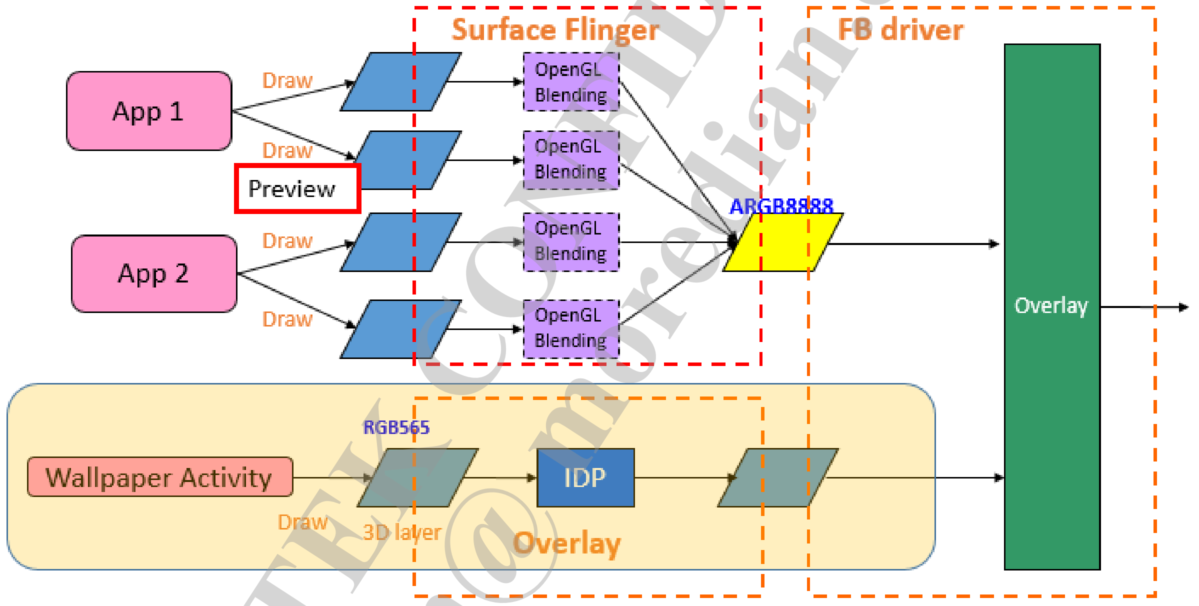
```
class DrawableEngine extends Engine {
    private final Object mLock = new Object();
    private WallpaperObserver mReceiver;
    Drawable mBackground;
    float mXOffset;
    float mYOffset;

    class WallpaperObserver extends BroadcastReceiver {
        public void onReceive(Context context, Intent intent) {
            updateWallpaper();
            drawFrame();
            System.gc();
        }
    }
    @Override
    public void onCreate(SurfaceHolder surfaceHolder) {
        super.onCreate(surfaceHolder);
        IntentFilter filter = new IntentFilter(Intent.ACTION_WALLPAPER_CHANGED);
        mReceiver = new WallpaperObserver();
        registerReceiver(mReceiver, filter);
        updateWallpaper();
        surfaceHolder.setSizeFromLayout();
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        unregisterReceiver(mReceiver);
    }
    @Override
    public void onVisibilityChanged(boolean visible) {
        drawFrame();
    }
    @Override
    public void onTouchEvent(MotionEvent event) {
        super.onTouchEvent(event);
    }
    @Override
    public void onOffsetsChanged(float xOffset, float yOffset,
            float xOffsetStep, float yOffsetStep, int xPixels, int yPixels) {
        mXOffset = xOffset;
```

7 Modification of LiveWallpaper

Use some modifications to enhance the live wallpaper implementation.

7.1 Using HW oevrlay

- **Architecture**
  - Use HW Overlay



7.2 Using S/Wor H/W Gles:

- *Architecture*

    Use Hw Overlay

        Root Cause

            The update rate of overlay layer is faster than compositing multiple normal layer

        Modification

            When first onOffsetsChanged() be called, set a notification to RS to change output layer from normal layer to overlay layer

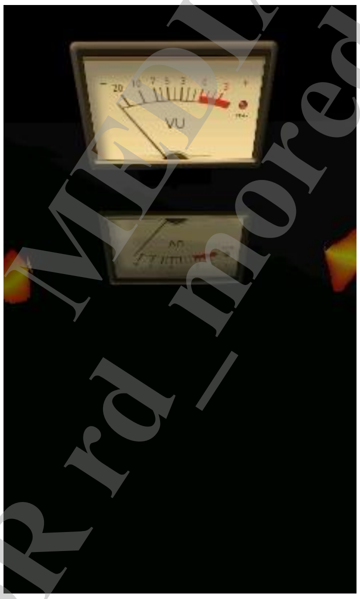
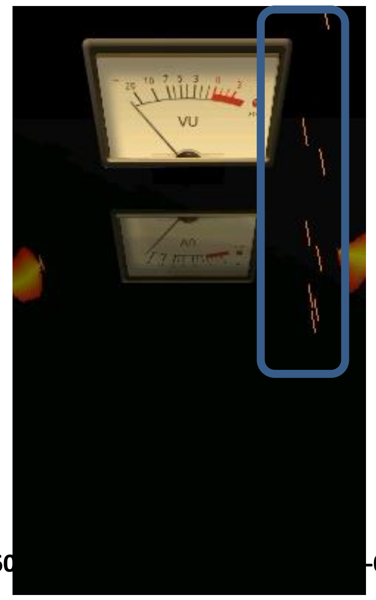
    Use SW / HW GLES

        Root Cause

            Some of LiveWallpaper meet GLES HW limitation / bug

        Modification

            When constructor of LiveWallpaper, use a new API of render script to notify render script create SW or HW GLES.

7.3 Modification of LiveWallpaper Components:

Layout

Root Cause:

The original design of LiveWallpaper is only for WVGA (800 x 480), so in different resolution of display size, it looks some abnormal

Modification List

Name	***Modification***
Grass	***Blade Length***
Water	***Ripple position***
Nexus	***Touch event position***
Music Visualization	***Wave Length***
MagicSmoke	***Texture size fixed to 256***

7.4 Modification of LiveWallpaper variables to reduce CPU usage:

Performance

Root Cause:

Some of LiveWallpaper have performance issue so need to modify some variable to reduce CPU usage

Modification

Fall	*Galaxy*
• ***Mesh Resolution (45 >>> 25)*** • ***Amplitude Threshold*** • ***Drop Gravity*** • ***Min Gravity*** • ***Leaf Drop Speed*** • ***Spread Speed*** • ***Offset Range*** • ***Random Ripple Gravity*** • ***Leaf Drop Gravity*** • ***Initial Drop Gravity***	• ***Particles Count (12000 >> 3000)*** • ***Particle Size***