Andriod Component Development Documentation

Group 3.1

Sophia Wilberscheid & Aleksandar Colic

Table of Contents

Introduction

On the onset of the project, both Alek and I agreed, that it seemed overwhelming due to the scope of Java knowledge and experienced needed to create this type of Developer Tool. It is my belief that with more time to research and test ideas, we could of developed an end product that would be very usable but in the haste of development a lot of bridges were crossed that in hindsight could have been avoided and cost us precious time for the true development of this tool. This was also due to the large learning curve to learn Java, research its libraries, and the many 3rd party tools available.

While writing this documentation, it is my hope that we will have the beginnings of a viable tool that can be developed further with reusability.

Research - Droid Draw & Other Libraries

Code license:

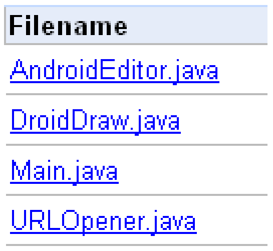
GNU General Public License v2

Last updated August 10, 2009.

Research led to studying DroidDraw since the code was available on-line and it could also be decompiled easily (if necessary). After completing and understanding how to design the Java interface, the main focus shifted to the actual code development. At first, it seemed it might go quite easily with pre-written code to look at but became pretty intense because of the amount of code and the desire to understand what the developer did.

DroidDraw consisted of eight folders, of which the ui folder, that contained all the *.png files, and the org folder which contained all the classes, where of the most interest since it contained what were the inner workings of DroidDraw.

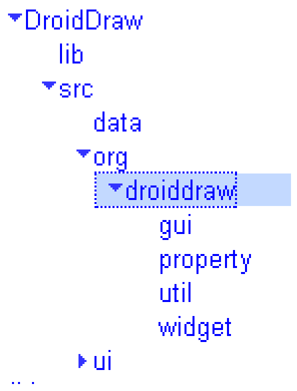
The “main” files in DroidDraw were:



Filename
AndroidEditor.java
DroidDraw.java
Main.java
URLOpener.java

Main.jave contained “main” , as expected and loaded all the components. What was strange was that DroidDraw was almost (minus main) a repeat of main.java in the sense it loaded all the image files, etc. AndroidEditor was used to set up the GUI interface with the correct “phone” background choosen for JPanel. Originally the main focus was in studying main.java to learn how the interface was setup, how the developer loaded his images, and set his listeners up. But in the end this was a great waste of time since he had broken up his code so much that it took many hours to connect all the code and understand what he did.

For example within DroidDraw there were four folders with several classes that were developed for each application.



```
▾DroidDraw
    lib
  ▾src
      data
    ▾org
      ▾droiddraw
          gui
          property
          util
          widget
    ▸ui
```

Within the gui folder there were approximately twenty-five classes which defined the different gui interfaces for DroidDraw. In widget, which defined all the classes to create a widget, there were about thirty classes defined for each type of widget. So it is easy to see where many hours were spent.

In the end, it was difficult to use the pre-existing code because of the time constraint of trying to understand all the code and the fact that we went with using SWF components because it offered several “widgets” that we were interested in using. The developer of Android used JPanel, JLabels, etc. from swing.

It would have been much better to just start from scratch and develop the tool from online snippets as it is what we ended up doing. So, in all, almost two and half weeks was wasted studying DroidDraw; o.k., not a total waste of time because it generated a lot of ideas and also an understanding of what was available in the Java libraries. We also learned that you could not “directly” drag and drop (dnd) a component but had to create your own classes to create this affect. All the dnd documentation always dragged and dropped textual content.

While researching libraries, we came across the nebula library which had a nice looking gallery which was used for the development of the interface.

http://www.eclipse.org/nebula/widgets/gallery/gallery.php

This is a great source for sample codes and was very useful for development of the interface.

Developing the Interface

The one element of DroidDraw that we did not like was its “archaic” look and feel as opposed to the MobiOne Interface Designer which seemed more “modern” and had a cleaner look and feel. DroidDraw was chosen for the main basis of the project, based on its functionality (it worked) and the fact that the source code was available. We choose to try and create the same look and feel we had with MobiOne.

The first major task was to make a better user interface with the correct components. Using many available components online the interface was made using the following components:

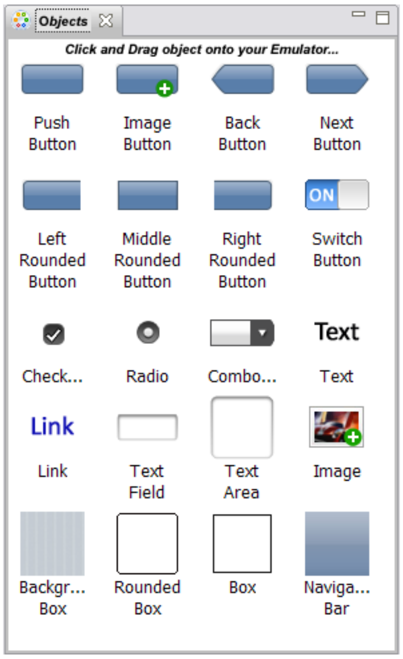
DroidDraw Package - http://code.google.com/p/droiddraw/

Nebula Gallery Widget-
http://www.eclipse.org/nebula/widgets/gallery/gallery.php

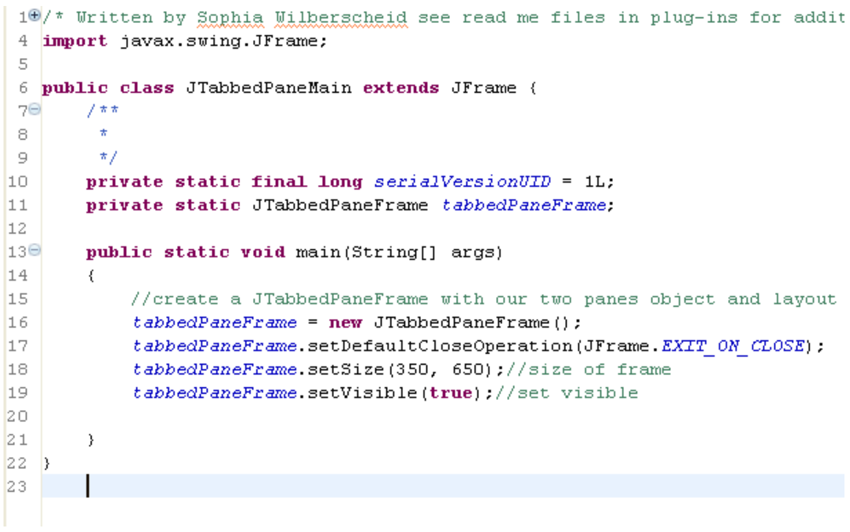
Java.swt - http://www.eclipse.org/swt/

The user interface went through three revisions as the first try did not produce the look and feel that was desired. The original design was based off of the MobiOne Interface.

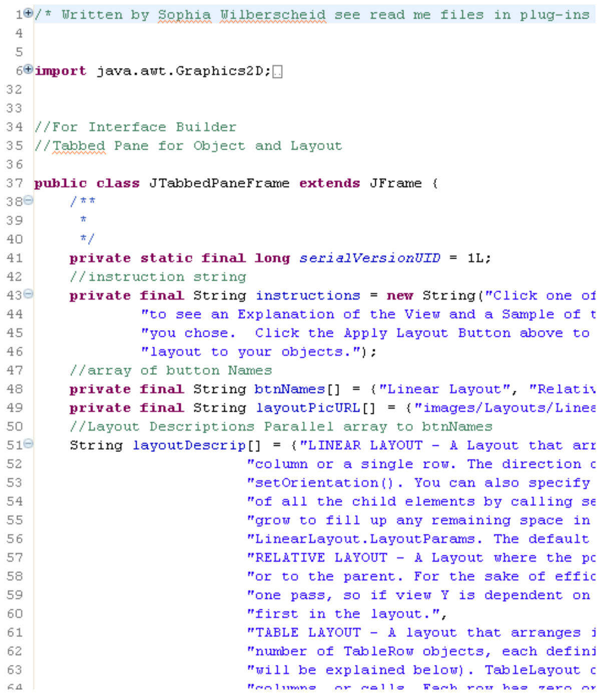
The Object Panel



In the original design developed two classes, JTabbedPaneMain and JTabbedPaneFrame. JTabbedPaneMain contained the code to run the Java Application.



```
/* Written by Sophia Wilberscheid see read me files in plug-ins for addit
import javax.swing.JFrame;

public class JTabbedPaneMain extends JFrame {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private static JTabbedPaneFrame tabbedPaneFrame;

    public static void main(String[] args)
    {
        //create a JTabbedPaneFrame with our two panes object and layout
        tabbedPaneFrame = new JTabbedPaneFrame();
        tabbedPaneFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        tabbedPaneFrame.setSize(350, 650);//size of frame
        tabbedPaneFrame.setVisible(true);//set visible

    }
}

```

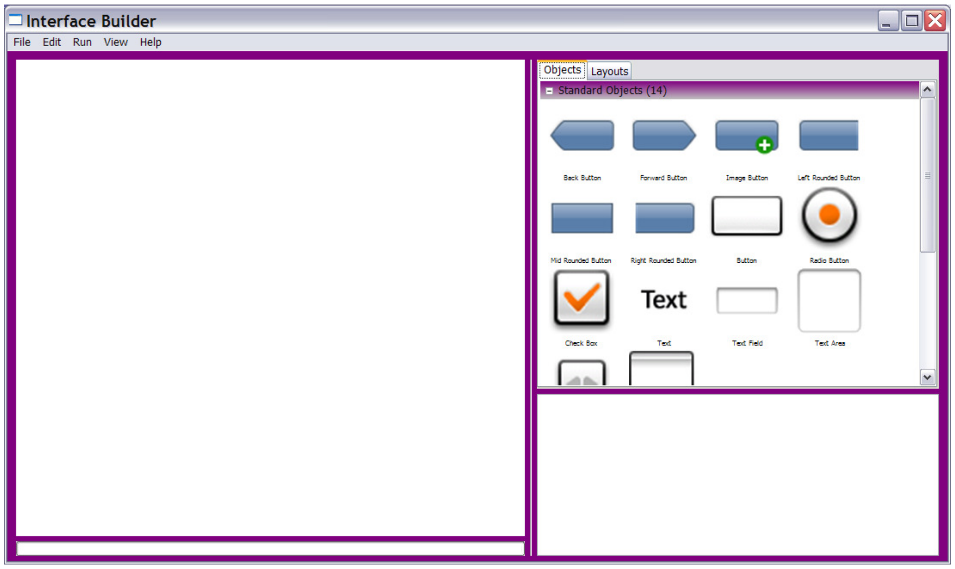


```
/* Written by Sophia Wilberscheid see read me files in plug-ins


import java.awt.Graphics2D;


//For Interface Builder
//Tabbed Pane for Object and Layout

public class JTabbedPaneFrame extends JFrame {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    //instruction string
    private final String instructions = new String("Click one of
            "to see an Explanation of the View and a Sample of t
            "you chose.  Click the Apply Layout Button above to
            "layout to your objects.");
    //array of button Names
    private final String btnNames[] = {"Linear Layout", "Relativ
    private final String layoutPicURL[] = {"images/Layouts/Linea
    //Layout Descriptions Parallel array to btnNames
    String layoutDescrip[] = {"LINEAR LAYOUT - A Layout that arr
                        "column or a single row. The direction c
                        "setOrientation(). You can also specify
                        "of all the child elements by calling se
                        "grow to fill up any remaining space in
                        "LinearLayout.LayoutParams. The default
                        "RELATIVE LAYOUT - A Layout where the pc
                        "or to the parent. For the sake of effic
                        "one pass, so if view Y is dependent on
                        "first in the layout.",
                        "TABLE LAYOUT - A layout that arranges i
                        "number of TableRow objects, each defini
                        "will be explained below). TableLayout c
                        "columns, or cells. Each row has zero or
```

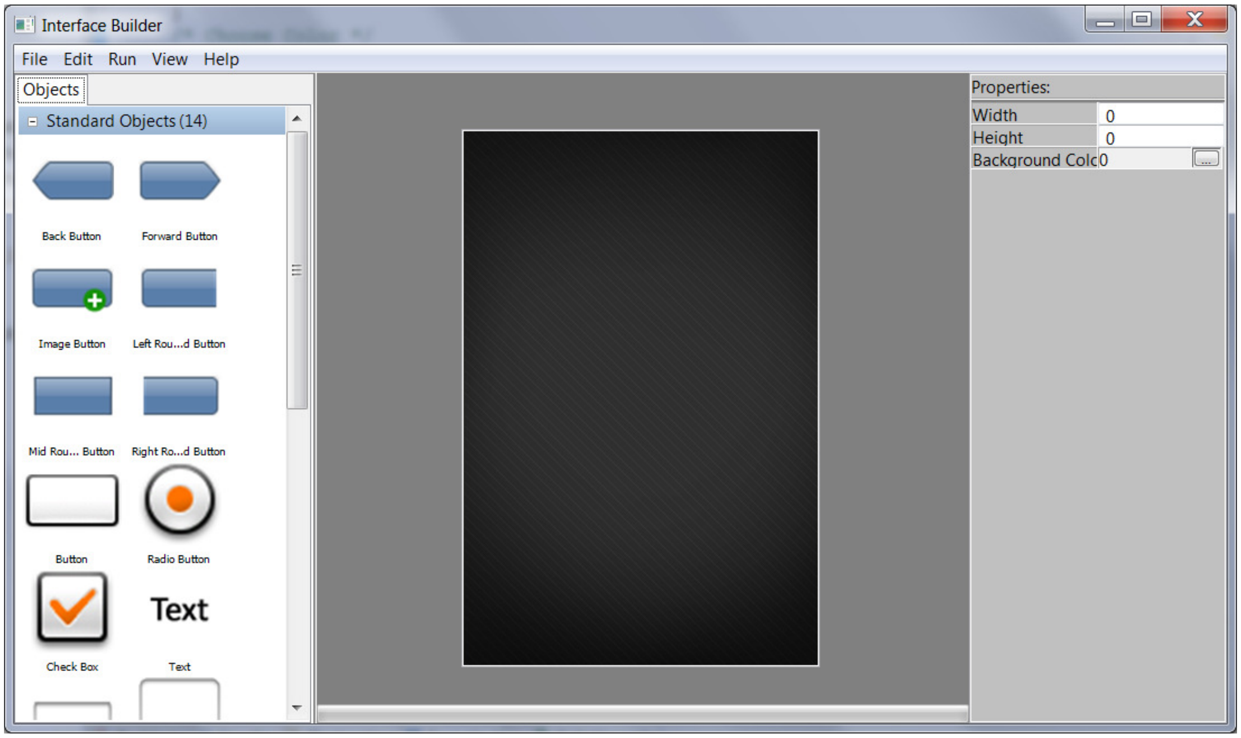
In the end the object panel morphed into the same look as the original design but the objects were split into two categories to group the most common objects together. Many of the objects shown on the Object Panel were not available on DroidDraw and it was the intent to develop the code to include these components.

The main shell for the components was developed using SWT and the Nebula widgets. Both Aleksandar and I developed the interface, exchanged code, and ended up with the final product. The second phase of the IDE looked like the figure below:

Second Design of Interface Builder

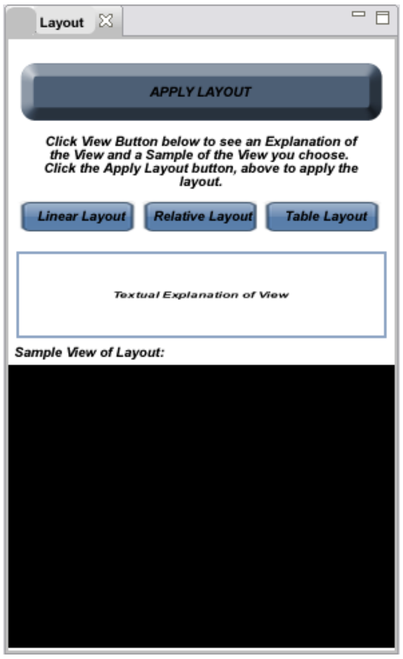


FINAL Design of Interface Builder



Properties:	
Width	0
Height	0
Background Colc	0

The Layout Panel

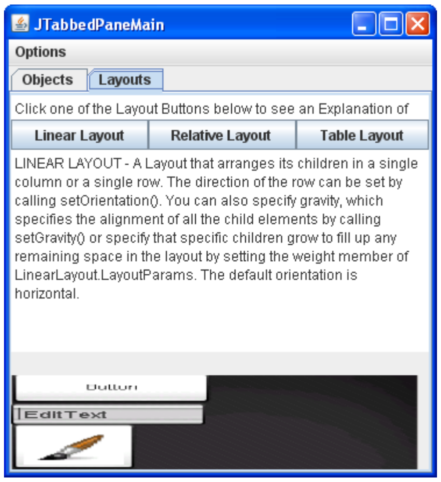


Click View Button below to see an Explanation of the View and a Sample of the View you choose. Click the Apply Layout button, above to apply the layout.

Sample View of Layout:

The original idea behind the layout panel was to allow the developer to not only choose the main layout for the objects but to also show a sample of the layout in "action" (a sample picture) and a written explanation of the layout to make it more user friendly. At this present time it is not as "user friendly" as we would like but it serves its purpose. We both agreed that we would like to improve the samples.

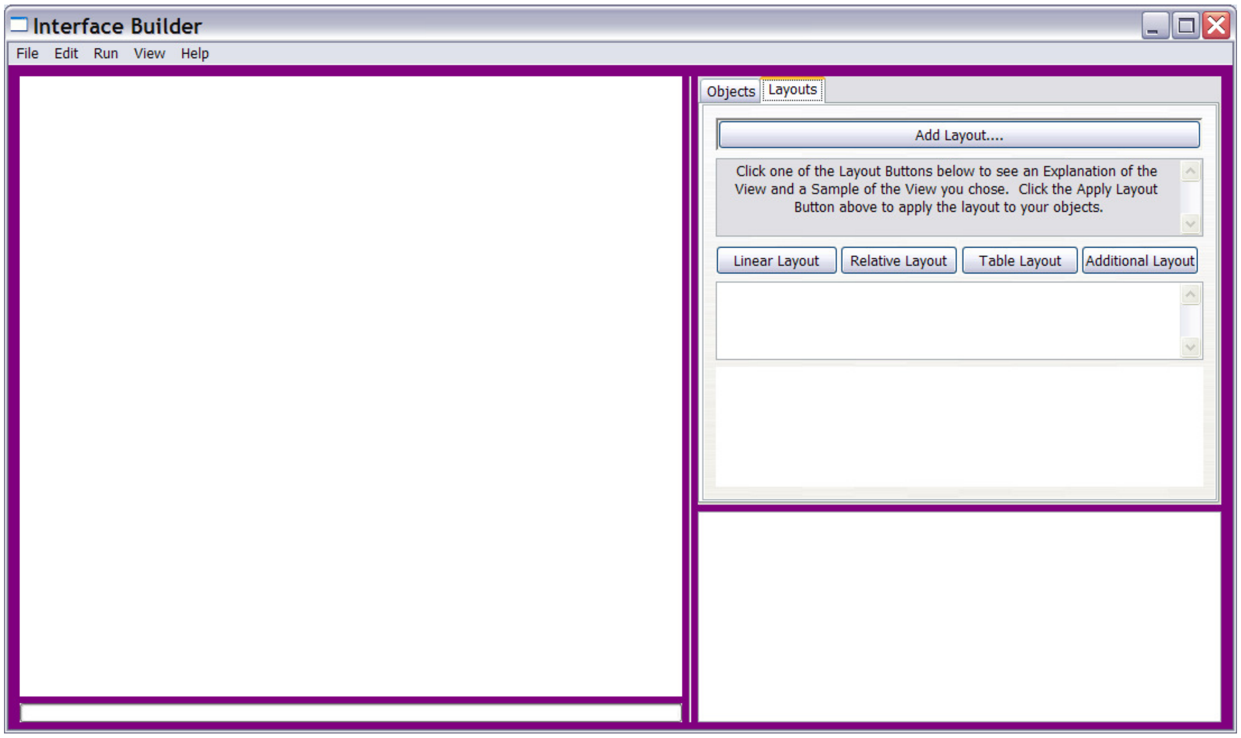
First Try:



Click one of the Layout Buttons below to see an Explanation of

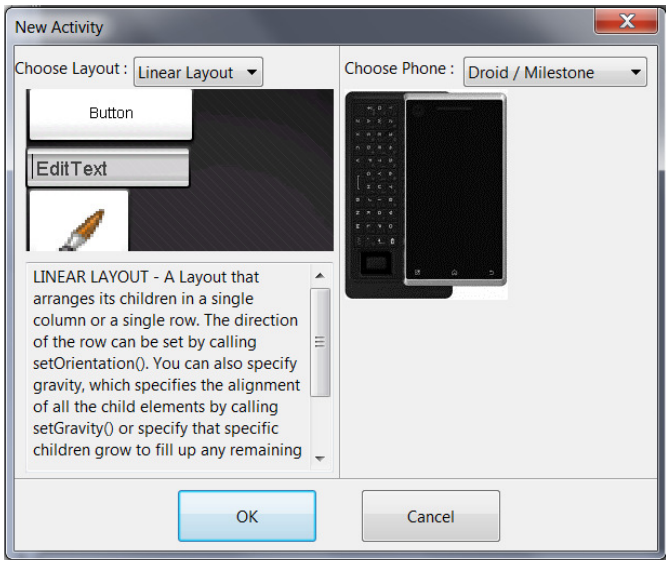
LINEAR LAYOUT - A Layout that arranges its children in a single column or a single row. The direction of the row can be set by calling setOrientation(). You can also specify gravity, which specifies the alignment of all the child elements by calling setGravity() or specify that specific children grow to fill up any remaining space in the layout by setting the weight member of LinearLayout.LayoutParams. The default orientation is horizontal.

Layout Panel in Interface Builder (second build of interface)



Click one of the Layout Buttons below to see an Explanation of the View and a Sample of the View you chose. Click the Apply Layout Button above to apply the layout to your objects.

Aleksandar felt that the properties window should execute on the opening of a new file, this made a lot sense, so the properties tab folder was moved out into its own window. The user is then allowed to choose the Main Layout for the objects and what type of device they wish to emulate.

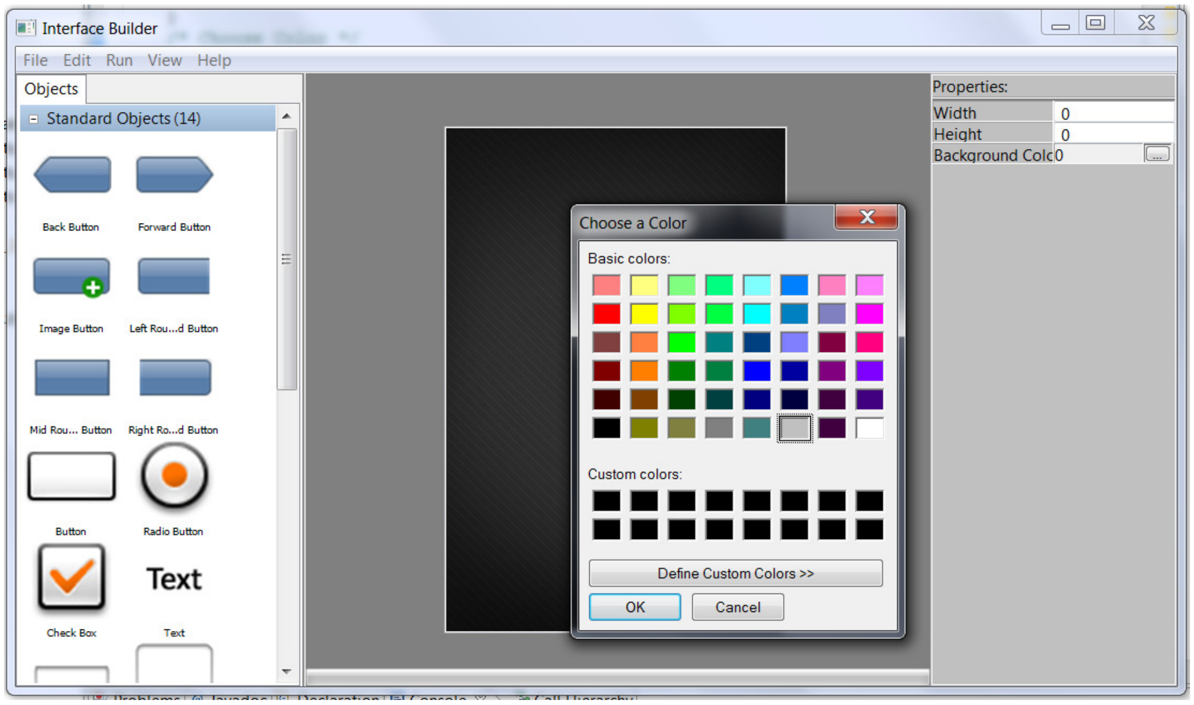


Choose Layout : Linear Layout

Choose Phone : Droid / Milestone

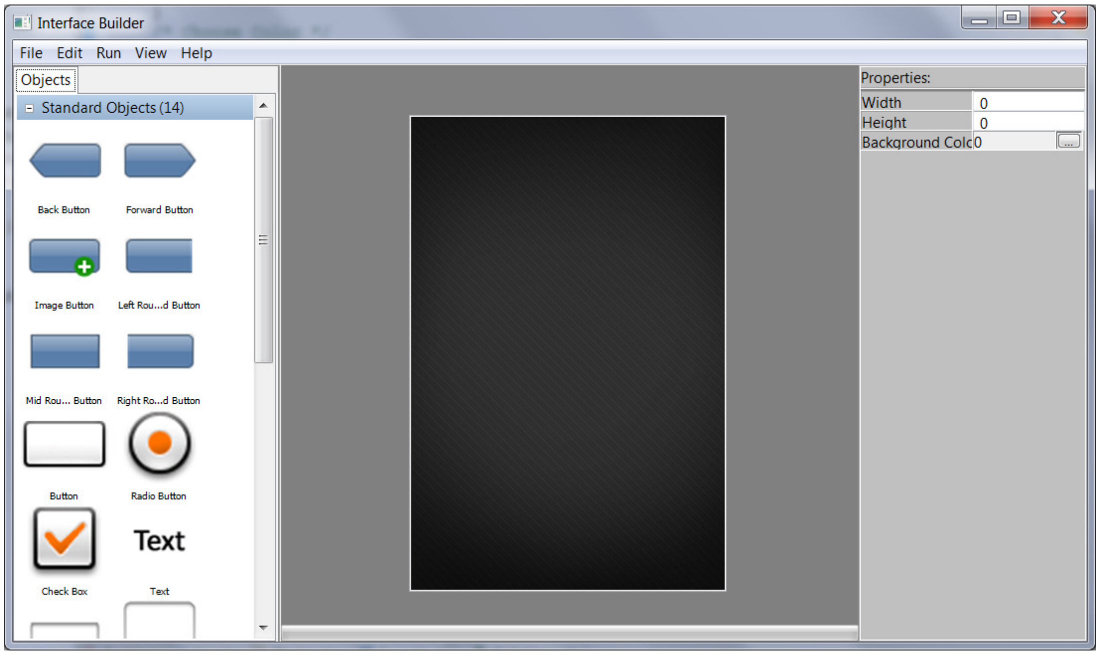
LINEAR LAYOUT - A Layout that arranges its children in a single column or a single row. The direction of the row can be set by calling setOrientation(). You can also specify gravity, which specifies the alignment of all the child elements by calling setGravity() or specify that specific children grow to fill up any remaining

In addition we added a color chooser panel to change the colors for the components.



Width	0
Height	0
Background Colc	0

Screenshot of Interface Builder with New File Opened



Width	0
Height	0
Background Colc	0

Developing the Code

To start this next step, an analysis of the existing DroidDraw code was done. What each class accomplished, how each piece was written and how we could use this to develop our own code. This step was and is by far the most tedious step. Although, the code itself was readily available it was extremely time consuming to read through all the code itself and connect all the pieces together. In all, it was what was expected (as far as how it was built structurally) and much of the code still needs to be examined. This route was too tedious and it was decided to develop our own code and use what was learned from DroidDraw for reference.

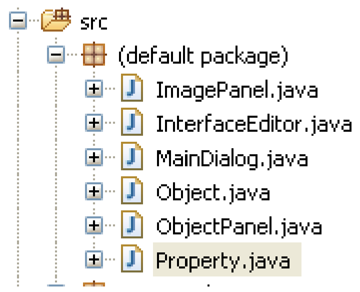
To date the following has been coded:

- New Interface
- Menu Items
  - File -> New
  - File -> Open
  - File -> Save
  - File -> Save As
  - File -> Close
  - Help -> About
- Object Panel
- Properties Window
- Emulator
- Add Object/Delete Object
- Tracking of Objects

We could develop the XML using a hierarchy structure (binary tree) which would allow us to implement the generation of any interface with its associated layouts; this would also include containers or components that had their own associated layouts. We still need to code all the classes for each object and create the correct associations with the XML. The properties class needs to also be created. At this time we have partial properties list and the objects are tracked by using a fixed array.

Class Definitions

The following Class Definitions have been created:



```
src
  (default package)
    ImagePanel.java
    InterfaceEditor.java
    MainDialog.java
    Object.java
    ObjectPanel.java
    Property.java
```

We still need to clean up code using a packages, class definitions and functions.

Objectives Accomplished:

Original Objective:

To develop an Interface Builder for Android that is easy for developers to use with little or no knowledge of xml.

The highlighted objectives have been met or are currently a work in progress:

- Review current Designs for Interfaces and decide what our main design will be (Storyboard)
- Develop Basic Interface with Associated Windows for Tool and Properties
  - Main window
  - Window with Emulated Phone
  - Properties/Tools Windows
  - Window for Objects (Graphical Components)
- Develop a Library of Objects (Graphical Components: buttons, textboxes, etc)

- Develop the Code behind the Interface (Java) – PARTIAL COMPLETION
- Generation of XML code
- Testing - TESTING ON CURRENT COMPONENTS HAS BEEN DONE

Task Outline

We were right on target with our original Task Outline even with the loss of Ulysses at the beginning of the project. The amount of time we took trying to study the different libraries and associated classes delayed our progress greatly; in addition, we also decided to incorporate Ulysses part of the project into the overall project between the both of us. The following is an outline of our progress:

February 18 until March 4 - ONTIME

March 4 until March 18 - ONTIME

March 18 until April 1 - ONTIME

April 1 until April 15 – FALLING BEHIND

April 15 until April 27 – FELL BEHIND

Needs for Future Development

To DATE: Wednesday, April 28, 2010

- Code Clean-up
- Fix Layout Popup Window (better sample pictures)
- Add a Horizontal Layout Option for Phone Emulator
- Further development of available objects in Andriod
- Development of classes for each object
- Generation of XML

- Completion of the Properties Class
- Complete Testing of Code