ENCOURAGING COLLABORATION THROUGH APP INVENTOR

Thesis
Submitted in Partial Fulfillment
of the Requirements for the

Degree of
Master of Arts in Interdisciplinary Computer Science

Mills College
Fall 2012

By

Katherine Kyle Feeney

Approved by:

Interdisciplinary Advisor _____
Professor Harold Abelson
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge

Computer Science Advisor _____
Professor Ellen Spertus
Department of Mathematics and Computer Science
Mills College, Oakland

Director of ICS Program _____
Professor Susan Wang
Department of Mathematics and Computer Science
Mills College, Oakland

ABSTRACT

App Inventor is a free, open source application that permits people with any level of programming background to create software applications for the Android operating system. At the time of its open source release, it lacked important resources for supporting collaboration: documentation for source code contributors and technological support for users developing apps in a group environment. To address these issues, I added a property to an existing component and then created detailed documentation about the process for future developers. I also created a tool for merging multiple projects, which will encourage teamwork by allowing multiple users to code separately and then combine their work. These contributions will increase collaboration among users as well as developers of App Inventor.

CONTENTS

FIGURES

TABLES

1. INTRODUCTION

App Inventor is a free, open source application that permits people with any level of programming background to create software applications for the Android operating system. App Inventor is a project that I am excited to be working on/ with and in doing so found that it was lacking in two important areas: sufficient documentation for contributors to the source code and supporting users who were developing in a group environment.

I contributed to improving the available documentation by creating a paper that details the steps I took to add the shape property to the Button component. This contribution will inform future developers who would like to add a property to a component.

I created the App Inventor Merger and wrote the associated documentation to make it easier for users of App Inventor to develop as a group. The App Inventor Merger allows users to develop different screens of a project independently and then merge them together into one project. The documentation I created around the App Inventor Merger describes not only how to use the tool but also how to develop in a team environment.

Both of these contributions to App Inventor encourage collaboration amongst developers, whether they are developers of App Inventor or developers with App Inventor.

1.1 BACKGROUND

1.1.1 APP INVENTOR

App Inventor is a free, open source application that permits people with any level of programming background to create software applications for the Android operating system. App Inventor uses a graphical user interface that allows users to drag and drop blocks (puzzle-shaped objects) to build their application without ever having to write traditional code. A simple example of this interface is shown below in Figure 1.

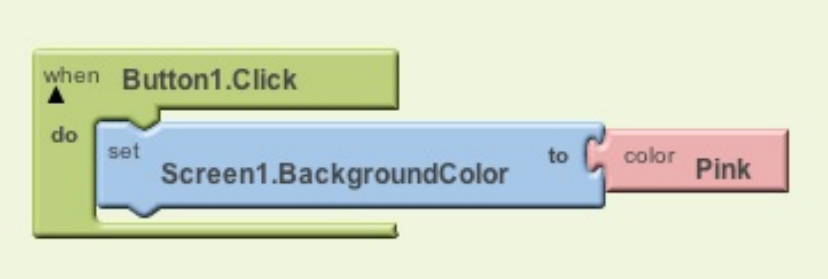


Figure 1: Blocks interface example.

1.1.2 GOOGLE TO MIT

Google originally launched App Inventor in July 2010. At that time App Inventor was a free web service that was provided to the public as a part of the Google Labs suite. In August 2011, Google announced that App Inventor would be released as an open source project.

In a response to the end of Google App Inventor, the Center for Mobile Learning was established at the MIT Media Lab to continue providing App Inventor to the public. In the fourth quarter of 2011 the Center of Mobile Learning started

working on App Inventor and in March of 2012 a beta version of MIT App Inventor was released to the public.

1.1.3 App Inventor Uses

Many different organizations and individuals with dramatically different goals are currently using App Inventor. This is possible partially because of how accessible App Inventor is both in the sense of programming skills and in the sense of having physical access to technology.

App Inventor is currently being used in a variety of educational settings, including classrooms ranging from elementary school to college and after-school programs.. While many do teach computer science using App Inventor, there are also a number of educators using it as a tool to engage students while teaching any subject. Many after school programs are being developed around App Inventor. These programs often focus on encouraging groups that are typically underrepresented in technology to feel empowered to be creators of technology.

App Inventor is also being used to foster entrepreneurship [MIT Center for Mobile Learning, 2012] [Iridescent, 2012]. Entrepreneurs are taking advantage of the ability to quickly and cheaply put together prototypes of their ideas, which they can use to pitch to venture capitalists. Using App Inventor eliminates a lot of time and work that is often done before there is even any funding for a project. App Inventor also broadens the pool of individuals who can be entrepreneurs in the tech industry. It is no longer necessary to know how to program in order to pitch an idea with a working prototype.

1.1.4 Experience with App Inventor

My first experience with App Inventor was in February of 2011 when I worked as a lead teaching assistant for an after school program called The Technovation Challenge. The Technovation Challenge is a program run by the non-profit Iridescent, which focuses on science, technology, engineering and math (STEM) education for underserved and underrepresented youth. The goal of the Technovation Challenge is to encourage women in technology and entrepreneurship and they strive to reach this goal using App Inventor. As a lead teaching assistant I spent ten weeks working with a group of high school girls to teach them App Inventor and help them develop their own original app.

In January 2012, I again worked with the Technovation Challenge, this time as an instructor. As an instructor I taught approximately fifty girls computer science concepts through App Inventor and assisted ten teams in the development of apps. Both as an instructor and a lead teaching assistant I was able to observe firsthand the users' experience learning and using App Inventor.

1.2 Motivation

I was drawn to this project for a variety of reasons, including App Inventor's mission and the desire to improve the user experience within App Inventor. In this section, I will provide a detailed description of my motivations.

1.2.1 Interest in App Inventor

One of my main goals while working in the computer science industry is to work on projects that make both the creation and use of computer technology more

accessible to the masses. Computers are infiltrating all aspects of our lives and currently it is a limited, elite group that has the privilege to create and use that technology. It is important that this begins to change because all people, particularly children, have a right to learn the technological skills necessary to engage fully in today's increasingly computer-dependent society.

Creating tools and training that are accessible to users of all levels of experience is an essential element of democratizing computing skills. Another element is keeping those resources affordable. App Inventor tackles both of these challenges, as it is designed for people without coding experience and is free to use. If an individual or organization has access to a computer with Internet access, then they can run App Inventor at no additional cost. Not only is App Inventor a free, open-source software, but the platform for which it makes apps (Android) is also open source. Since, Android is open source any manufacturer can create an Android device without having to pay to use the operating system. This means that there are more, inexpensive options for devices that can run apps created with App Inventor.

Another reason that I decided to work on App Inventor is that one of their goals coincides perfectly with one of mine, which is to encourage all people to become producers, rather than only consumers, of technology. App Inventor actively reaches for this goal by providing several free online tutorials to help new users get started. This allowed many outside organizations to deploy App Inventor to empower populations that are traditionally underrepresented in computing. The intuitive, forgiving interface has allowed organizations to open up computing

classes for inexperienced coders. For example, the University of San Francisco offers a mobile apps class using App Inventor as a general education class and attracts over 50% women. According to a Business Week story, this class has actually encouraged women to consider majoring in computer science [King, 2012].

1.2.2 App Inventor Changes

In my project, I addressed two characteristics of App Inventor that made it unnecessarily difficult to use: collaborative programming and customizing the end-user interface. I encountered the need for both features while working with The Technovation Challenge. I observed high school students as they used App Inventor and I noticed that users had two major frustrations: they wanted more control over the user interface and they wanted it to be easier to develop an app as a team.

As the groups developed their app, it became clear that App Inventor was very restrictive when it came to working as a group. A major part of this issue was that only one member of the group could be programming the app at once. When working in a group of five (as The Technovation Challenge teams were) it was often frustrating and boring for the team members who were not programming. A number of teams approached me and asked if there was any way for different team members to work on different screens at the same time and then combine them at the end. At the time there was not a solution to this problem so I decided to make it part of my thesis.

1.2.3 Documentation For New Developers

A key to building a community of developers for an open source project is to provide support to new contributors. Having detailed, accessible documentation is a major way of providing this support.

When I began exploring App Inventor's source code and setting up a local version on my machine I became painfully aware of how little documentation was available for new developers. Since App Inventor had just been released as an open source project much of the documentation that would typically be available had not been created yet. Because of this I kept detailed notes on my work and made a goal to create documentation that would allow new contributors to replicate my process. With these notes, I created and contributed to a number of different documents that are being used by current App Inventor developers. The major document I created was How to Add a Property to a Component but I also contributed to documents that detailed how to develop App Inventor on a Mac and how to develop with Eclipse.

In the rest of this document I describes in detail the solutions I developed as well as the process I went through to reach these solutions. I first discuss adding the button shape property to App Inventor and creating documentation about the process to benefit future developers. I then describe creating the project merger tool and associated documentation as well as how this tool is currently being utilized.

2. BUTTON SHAPE PROPERTY

The first feature I added to App Inventor was the ability to change the shape of Button components. Originally the Button component had one default shape, which used the system's defaults and varied from device to device. Now the user has the four options (default, rounded, rectangular, oval) to customize their interface.

When starting to work on the problem of allowing users to change the shape of their buttons, it was not my intention that it would be such a major aspect of my overall thesis. At that time I thought it would be a small, simple change that would help me learn about the source code and prepare me to do a much larger user interface change. This began to change relatively quickly after I began researching the problem.

Right away I realized how little documentation was available to support me in making this change and that in order to implement my changes I would need to become very familiar with a number of different areas of the code. This caused my focus to began to change from adding the property to give the user more control to documenting the process of adding a property so that it would be faster and simpler for future developers to add properties.

2.1 SOLUTION

2.1.1 SYSTEM SET UP

Before I could even start working on adding the button shape feature I needed to set up my system to be able to build and deploy my own version of App Inventor. This process took me about a week to complete partially because it was my first time working with a large open source code and partially because of the lack of documentation.

The first step was getting a copy of the source code. The source code was housed at Google code and could be cloned using the distributed source control management tool called Mercurial. While I had previously used similar tools I had never used Mercurial before and needed to download it and set it up on my computer. This process was simple and once complete I simply had to use the following command to copy the code to my computer.

hg clone https://code.google.com/p/app-inventor/

Following installing Mercurial and cloning the source code I went through the Mercurial start up guide and familiarized myself with its basic commands. I also installed Mercurialeclipse[1], a plug in for the Eclipse software to integrate Mercurial into Eclipse.

The next step was installing and setting up Apache Ant. Ant is a Java library and command line tool that is needed to build App Inventor as a Java application.

[1] http://javaforge.com/project/HGE

While downloading Apache Ant was simple, I had difficultly setting it up on my computer. The set up requires that the Ant folder be added to my computer's path, which I had difficulty doing this until I found a very helpful website[2]. Once Ant was installed I was able to generate javadocs for the App Inventor code and start to explore the classes that would be relevant to adding the button shape feature.

App Inventor runs using Google App Engine (a platform for developing and hosting web applications), therefore the next step was to create an App Engine account and familiarize myself with it. It took me about a day and a half to go through the tutorials and deploy two example applications. The two example applications I completed I developed using Eclipse and the App Engine Eclipse plug in. This did not turn out to be that helpful because I later learned that I could not set the App Inventor project up as an App Engine project in Eclipse and therefore could not use the App Engine plug-in. Other than that, through the tutorials I learned the basics of App Engine and how to set up and maintain applications that are deployed to App Engine.

In general I used the development environment Eclipse in all of my development work and since it has both a Mercurial and an App Engine plug-in I wanted my next step to be importing the App Inventor project into Eclipse. This turned out to be one the most frustrating parts of the entire project. I started by using the Mercurialeclipse plug-in to import the project. This seemed be a logical step

---

[2] http://sakibulhasan.wordpress.com/2010/07/30/install-apache-ant-on-mac-os-x/

since this would set up the link to the remote repository and make connecting to the source code easy. Through this process the project imported to Eclipse characterized as a “general project”. My first attempt was to convert it to an App Engine project so that it could use the App Engine plug-in. I was unable to convert the project so I settled on using Eclipse as an editor and deploying to App Engine using the command line. While this decision negated the need for the App Engine plug-in I still had the problem that the project was set up as a general project and not a Java project. Since the project was not a Java project, many of Eclipse’s built-in tools and the main reasons I find Eclipse helpful to use were unavailable. After many different attempts and much research I figured out how to import App Inventor as a Java project and could take advantage of many of the editing tools Eclipse has to offer.

The next and final step to setting up the project was to build and deploy my local version of App Inventor to App Engine. As stated above, I was not able to use Eclipse’s App Engine plug-in to deploy it so I needed to learn how to do it using the command line. Luckily, there was already some existing documentation that explained the necessary steps. Using this documentation, I created two shell scripts: one for building the project and the second for deploying the project to App Engine. These scripts are shown in Appendix A – Shell Scripts. Once the project was deployed to App Engine I was able to visit the url I specified when setting up App Engine and see my local version of App Engine. At this point I had my own version of App Inventor running. I had not made any changes to the code and my version of App Inventor was the same as the most recent public

release. I had also created a detailed step-by-step document of the steps I took that would allow future developers could benefit from my experience.

2.1.2 LEARNING THE CODE

The App Inventor source code has almost eight hundred files and over one hundred thousand lines of code. Therefore, after completing the set up my main focus was figuring out which areas of the code I would be working on and familiarize myself with them. One major way I accomplished this was by starting with the Button class and creating static class diagrams of its related classes, shown in Appendix B – Static Class Diagrams. I also looked at the alignment property, another Button property similar to shape, and began stepping through its code.

2.1.3 CHANGE BUTTON SHAPE PROGRAMMATICALLY

Before beginning to change the source code I needed to research and determine how I was going to actually change the button's shape programmatically. I had experience with changing the shape of buttons in Android applications but only by changing the layout using an xml file. In this case the shape needed to be changed on the fly and could not be hard coded in as it would be in an xml file. This turned out to be a very difficult problem to solve because not only did the shape of the button need to change, but also the ability to change the color and background image needed to remain. I built an Android app named ButtonStyle to test all the different methods I tried. These methods and why they would not work are list in Table 1 below.

Table 1: Button Shape Methods

Method	Issue
RoundRectShape()	could not set background color
ShapeDrawable()	originally thought that could not set background color but later determined a way
ColorDrawable()	could not round corners
PaintDrawable()	could not work with the current setBackgroundColor()

Eventually, I found a way to change the color of a ShapeDrawable() which allowed me to programmatically change the shape of a button. Below, in Figure **2**, are a few lines of code from the ButtonStyle app that successfully change the button shape and color.

```
// create view
LinearLayout view = new LinearLayout(this);
// create button
android.widget.Button button = new android.widget.Button(this);
// set button text
button.setText("Test Button");
// create drawable
ShapeDrawable drawable = new ShapeDrawable();
// set the drawable's color
drawable.getPaint().setColor(Color.RED);
// set the drawable's shape
drawable.setShape(new RoundRectShape(ROUNDED_CORNERS_ARRAY, null, null));
// set the button's background to the drawable
button.setBackgroundDrawable(drawable);
// add button to view
view.addView(button);
// show view
setContentView(view);
```

Figure 2**: Code excerpt from ButtonStyle**

2.1.4 Implementing Changes to the Code

Once I had a local version of App Inventor running, familiarized myself with the code and determined how I was going to change the button shape, I was ready to start working on the source code.

In the sections below I describe the process of adding a shape property with four options (default, rounded, rectangular and oval), although during the actual process I started with just two options (default and rounded) and added the additional two options after the first two were working. I organized the write-up this way for simplicity and because adding the two additional options were formulaic. The following sections are general descriptions of my work. For specific code changes see the diff file in Appendix C – Diff file for CL1 or the documentation in Appendix D – How to Add a Property to A Component.

2.1.4.1 Properties Panel

The first area I worked on was adding the shape property to the Properties panel so that testing would be easier. This is the panel in the Design view where the new shape property is visible to users when a Button component is selected. See the outlined area in Figure 3 below.

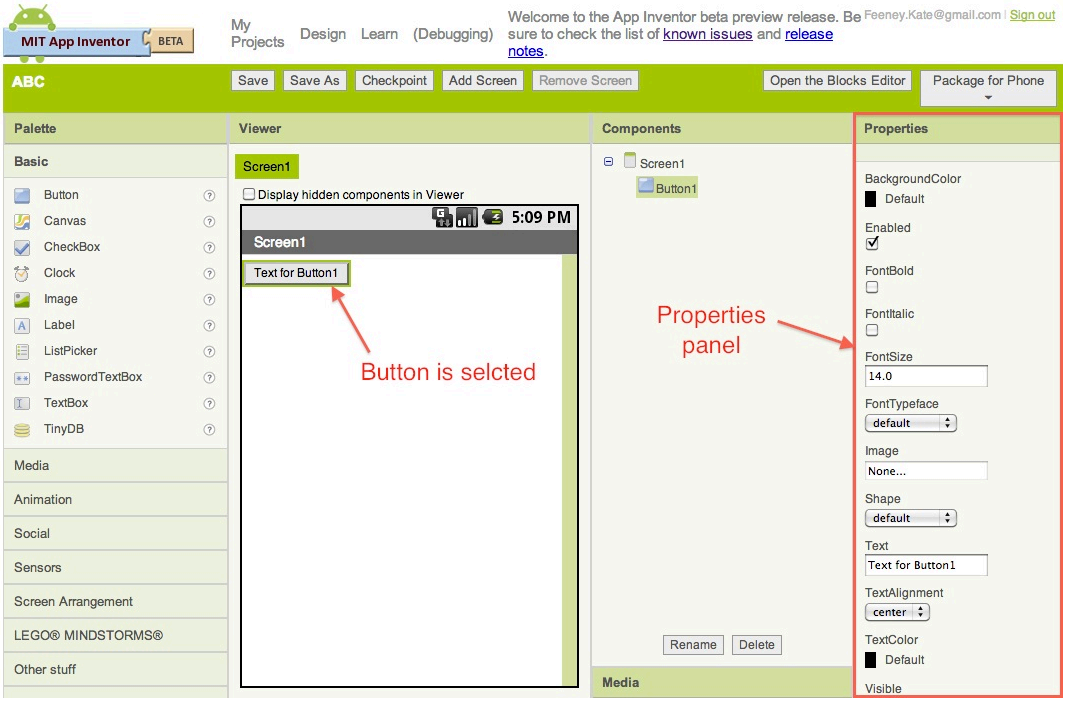


Figure 3: Properties Panel in Design view

The first step was to create a new property editor that offered a drop-down menu with the four legal shape values: default, rounded, rectangular and oval (as shown below in Figure 4).

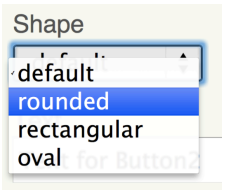


Figure 4: Shape PropertyEditor in the Designer

I did this by creating a new class, YoungAndroidButtonShapeEditor that extends the ChoicePropertyEditor. Since the new class extended the ChoicePropertyEditor all it needed to do was define an array of values to be

displayed in the drop-down menu and pass it to the ChoicePropertyEditor constructor.

After the property editor was created, I added logic to the YoungAndroidPalettePanel.createPropertyEditor() method so that any time a component with the shape property is selected in the Design view, the shape property editor is added to the Properties panel. I then associated the shape property to all ButtonBase components by adding a setter and getter for the shape property to the ButtonBase class. Both the setter and getter were marked with the SimpleProperty annotation and the setter specified that the shape property used the YoungAndroidButtonShapeEditor.

Once the steps described above were complete the Properties panel displayed the shape property when a Button component is selected in the Design view. This change is displayed in Figure 5 below.

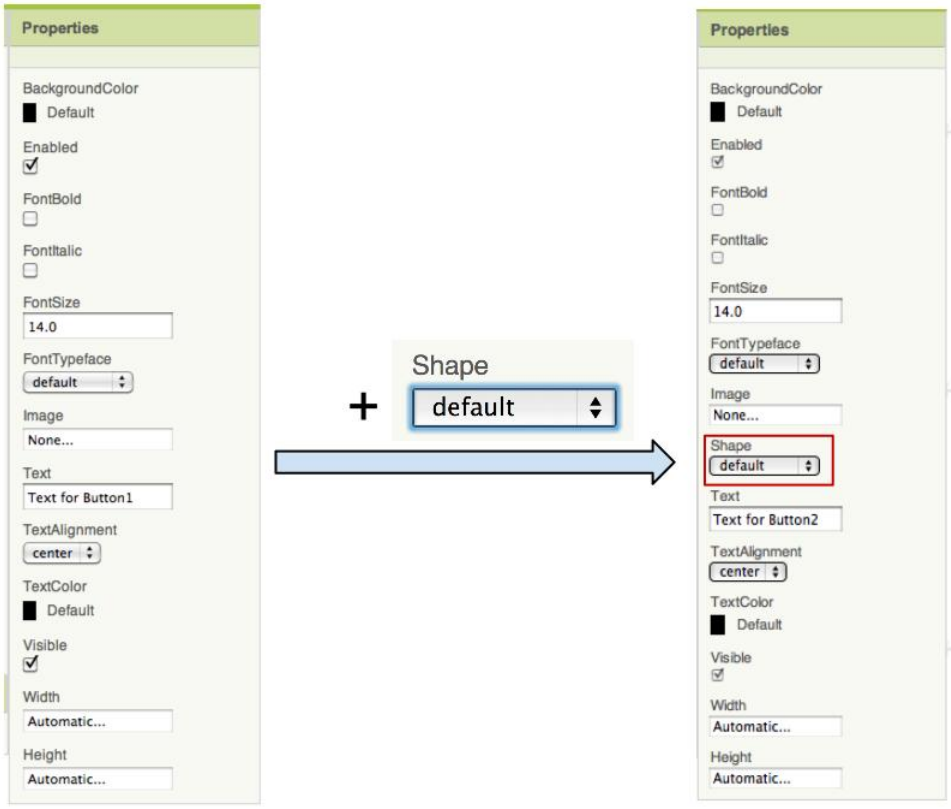


Figure 5: Button Property Panel

2.1.4.2 Viewer Panel

The next area I worked on was the Viewer panel. The Viewer panel is also part of the Design view and it shows what the running app would look like on the device. Figure 6: Viewer Panel in Design viewbelow shows the Viewer panel outlined in the Design view.

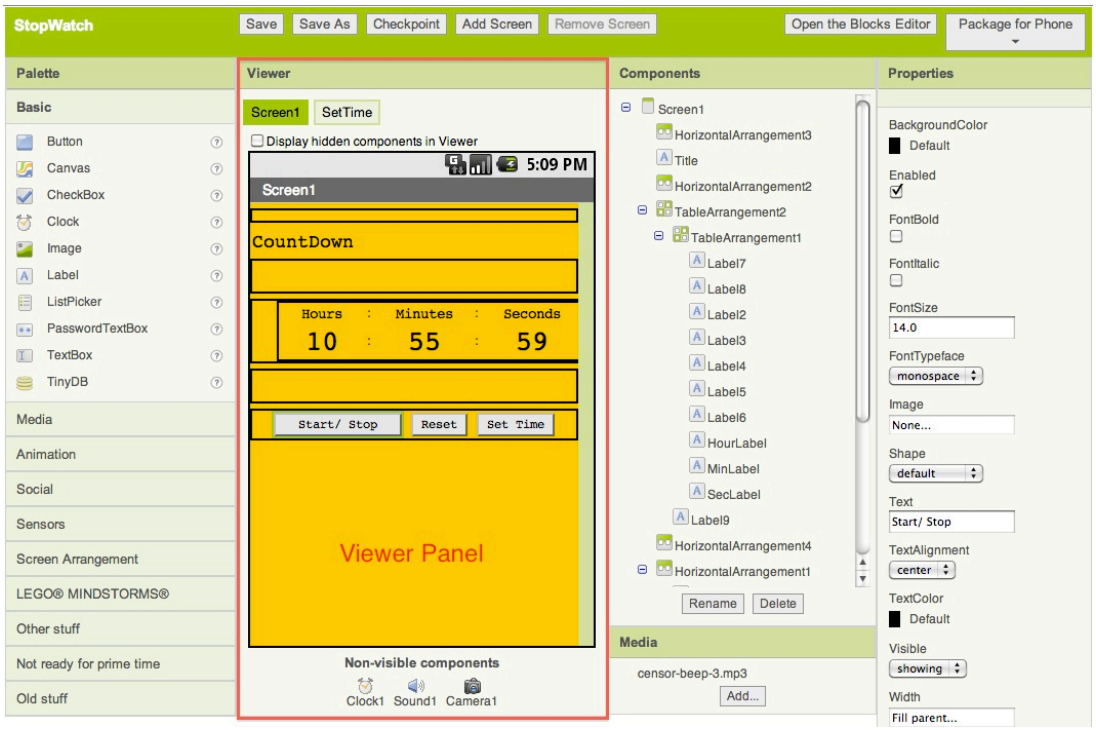


Figure 6: Viewer Panel in Design view

Now that the shape property would appear in the Properties panel, the Design view needed to update the appearance when the property changed.

To make this change I first needed to edit the MockButtonBase class. Every component has a corresponding mock class in the appengine project. For the ButtonBase component, the corresponding mock class is the MockButtonBase class. Mock classes are the visual representation of the component in the Designer, and therefore contain any code to change that visual representation.

All mock classes contain an onPropertyChange() method that is triggered when a property in the Properties panel is changed. The first step to changing the MockButtonBase class was to add the shape property to the onPropertyChange() method, so that when the shape property is changed and the

onPropertyChange() method is called, the onPropertyChange() method knows to call a new method setShapeProperty() with the shape option that was selected. The setShapeProperty() will actually change the button's shape.

The next step was to define the setShapeProperty() so that when it was called with the shapes listed in Table 2 below, the button in the Viewer panel would appear as the corresponding image shown in Table 2.

Table 2: Mock Buttons

Shape	**Image**	**Border Radius**
default		n/a
rounded		10 px
rectangular		0 px
oval		equal to the button's height

To change the images the setShapeProperty() method simply changed the border-radius attribute of the button widget that the MockButtonBase created based on which shape was chosen. For example, if the rounded shape was

chosen, the button widget's border-radius would be set to 10 px. The border radii are listed above in Table 2.

The last step was to update any other methods in MockButtonBase that were affected by the shape change. Since the button shape only changes if there is no background image, the Shape property and the Image property affect each other. Therefore, since the setShapeProperty() method was created, the setImageProperty() method needed to be updated. I updated the setImageProperty() method so that when a background image is added to a button, the border radii are set to 0 px and when a background image is removed the shape was reset to the selected option.

Now, when the shape property is changed in the Properties panel the shape of the button in the Viewer panel is updated.

2.1.4.3 Android Device

After the button shape was updated in the Viewer panel, the next step was to get it to update on the Android device. The first step to doing this was changing the updateAppearance() method in the ButtonBase class so that if the updateAppearance() method was called on a button that did not have a background image and that's chosen shape property was not the default then a new method setShape() would be called. The new setShape() method used the knowledge I gained about changing the shape of a button programmatically to create a drawable with the appropriate shape and color and then set the button's background drawable to that new drawable.

2.1.4.4 Version Numbers

The last step to adding the button shape property was to update all the necessary version numbers. The YaVersion, YoungAndroidFormUpgrader and BlockSaveFile classes all needed to be updated to reflect the new versions that were created by my changes.

2.1.4.5 Testing

Once all of the changes described in the previous sections were complete, I built a new version of App Inventor and deployed it to App Engine using the scripts described in section 2.1.1. This App Inventor server implemented my button shape changes and allowed me to test them before they were pushed to the master version of App Inventor.

First, I tested that when the shape property and other button properties were changed, the button in the viewer panel changed appropriately. To do this I created a new project with one Button component and completed the tests described in Table 3 for each of the shape properties.

Table 3: Button Shape Tests

Test	Expected Results
Changed background color to blue.	Button's background would change to blue and then back to the default color.
Changed background color to default	Button's background would change back to the default color.
Changed width and height to 90px	rectangular: 90px square rounded: 90px square with rounded corners oval: 90px circle default: depended on system

With background color set to blue and width and height set to automatic, added a background image and then removed it.	Button would take the shape and size of the image and then when the image is removed the button would return to the shape, color and size it was before the image was added.
With background color set to blue and width and height set to automatic, added a background image. Changed button shape and changed background color to red. Removed background image.	Button would take the shape and size of the image. When color and shape properties are changed the button would not change. When image is removed the button would have then new shape and will be red.

I then ran the project on an Android device emulator and repeated all tests. I did this to ensure that the changes would appear correctly on users' devices.

The tests described in Table 3 were chosen since the shape property changes the background drawable and therefore affects the background color and image. It was necessary to determine that different combinations of color, image and shape would result in the appropriate display.

2.2 Continuing Problems

After my code was reviewed and pushed live it was discover that the changes to the MockButtonBase did not work for all browsers. Tests showed that the button shape did not work in the following browsers:

- IE8
- IE7
- Firefox 12 (mac)
- Firefox 5 (PC)
- Firefox 3 (PC)

The shape has been tested and works properly in the following browsers:

- Chrome
- Safari 5
- IE9

The button shape property is changed by changing the CSS3 border-radius property. According to w3schools[3] this property should be supported in the following browsers:

- IE9+
- Firefox 4+
- Chrome
- Safari 5+
- Opera

Based on the above information, two things need to be determined. First, why the property change does not work appropriately in Firefox and if there is a workaround. Second, is it necessary to support IE versions before IE9 and if so what workarounds are possible. I have not completed these items.

2.3 DOCUMENTATION

As previously stated, a major focus of my work has been documenting the process of adding the ability to change the shape of a button. I used my notes from the processes and wrote a document that used adding the shape property as an example to instruct future developers how to add a property to a component. This document goes through each step I completed and explains how and why they were completed. Embedded in the document is also all of the code I added to the source code in order to make the change. This document can be viewed and edited by all members of the App Inventor Development team

[3] http://www.w3schools.com/

to insure that it remains up to date and accessible to those that would benefit from it. A copy of the How to Add a Property to a Component document, as of 9/1/2012, can be found in Appendix D – How to Add a Property to A Component.

2.4 Future Benefits

From this project I know first-hand how intimidating and daunting it can feel to start contributing to an open source project, especially one with as large and complicated source code as App Inventor. A major way of supporting new developers and keeping them interested in contributing to App Inventor is to provide documentation and instruction for tasks that they are likely to need to complete.

The documentation I developed provides clear instructions that a developer can follow to add a new property to a component. With these instructions developers will be able to implement changes much faster without having to figure out the whole process by themselves. This documentation will encourage developers to contribute to App Inventor and the more contributors there are, the faster App Inventor can grow and the more expansive the software can be.

3. PROJECT MERGING TOOL

The idea began as a simple way to merge two screens from two different App Inventor projects into one App Inventor project. This was a feature a number of my Technovation Challenge students requested to help improve their ability to work as a team. The students wanted to be able to divide up work between individuals or pairs of individuals so that more than one person could be coding at a time. This division of work brought up the issue of needing to know how to program as a team and made it clear that not only was a merger tool needed but also instructions on how to team program in App Inventor.

3.1 SOLUTION PROCESS

3.1.1 MANUAL MERGE

My first step was to merge two simple projects manually. It was necessary to start here so that I could learn all about the different parts of a project file and what files would need to be merged and which would not.

In App Inventor I created two new projects that I wanted to merge into one. The first project, Test1, had only a Screen1. Screen1 had a button that when pressed opened Screen3 (which did not exist in Test1). The second project, Test2, had a Screen1 and a Screen3. Screen1 was blank and had no block logic, it was only included because App Inventor requires that all projects have a Screen1. Screen3 had an image (goblue.png) and a button that when pressed closed Screen3 and went back to Screen1. The project created from the merge would be Test3. This project would have Screen1 from Test1 and Screen3 from Test2.

When the button on Screen1 is pressed it goes to Screen3 where the goblue asset is visible and when the button on Screen3 is pressed it will go back to Screen1.

App Inventor allows users to download the project files (or source file) for their projects, which can then be reloaded. I downloaded the project files for Test1 and Test2 and their structures are displayed below in Figure 7.

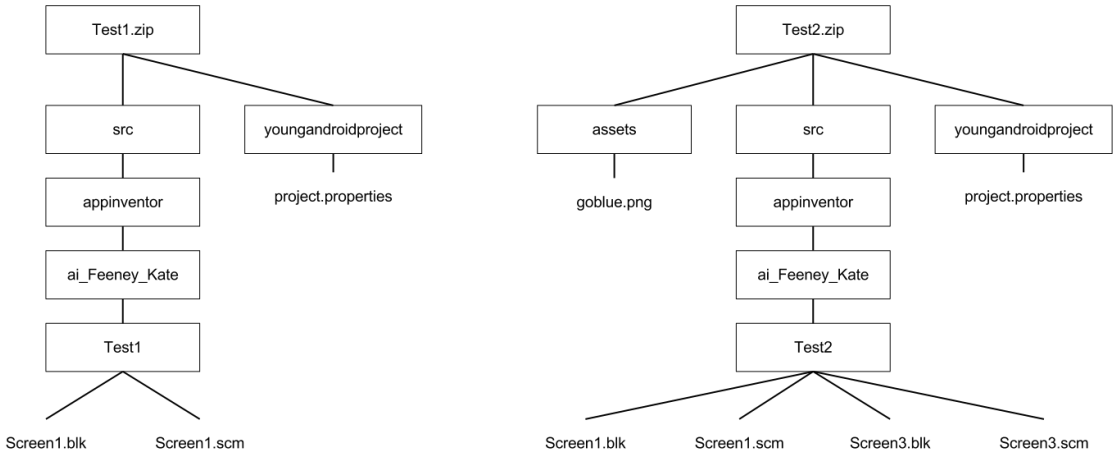


Figure 7: Test1 and Test2 project file structures

The assets folders hold all files that have been uploaded to the project as an asset. If no assets are associated with a project then there will be no assets folder. The src/appinventor/ai_[User_Name]/[project_name] folder holds a .blk file and a .scm file for every screen in that project. For example every project will have a Screen1.blk file and a Screen1.scm file. The Screen1.blk file holds all information about the blocks associated with Screen1 and the Screen1.scm file holds information about the layout of Screen1. The youngandroidproject folder

holds only one file, the project.properties file. This file holds basic information about the project, such as name, main screen and file structure.

From exploring the existing project files I determined that in order to merge these two projects I would need to create a new Test3.zip file with the same structure that is described above. This zip file would be a new project file that would hold the information that was needed from each of the original projects.

One question I still had was what project.properties file to put in the new project file or if a new file needed to be created. From talking to Liz Looney, a Google employee who was on the original of App Inventor team, I learned that the project.properties file was recreated when a project is loaded into App Inventor and therefore it did not matter which project.properties file was used. It didn't even matter if the file was blank as long as one existed in the file structure.

I created Test3.zip using the following steps:

1. Unzipped Test1.
2. Duplicated the unzipped Test1 folder and named it Test3.
3. Unzipped Test2.
4. Copied the assets folder from Test2 into Test3.
5. Copied the Screen3.scm and Screen3.blk in the Test3/src/appinventor/ai_Feeney_Kate/Test1 folder.
6. Compressed the Test3 folder.

I then attempted to upload Test3.zip to App Inventor as a new project but received an error stating, "The selected project is not a project source file! Project

source files are zip files." After doing a number of different tests it was determined that the error was caused because when the folder was zipped the file structure was being changed so that the first level in the directory was a Test3 folder instead of the assets, src and youndandroid folders. To fix this, instead of compressing the Test3 folder I compressed the three folders inside the Test3 folder into a zip file named Test3.zip

I tried again to upload Test3.zip to App Inventor and this time it was successful. A new project was created in App Inventor named Test3 and it had the Screen1 from Test1 and the Screen3 from Test3. The goblue assets from Test3 was listed in Test3's list of assets and appeared on Screen3.

The next step was to test that the blocks logic still worked and that when the button on Screen1 was pressed Screen3 appeared and vice versa. In order to test this the app must be packaged and installed on the phone. When I attempted to package the app for the phone it failed. Again after much testing and preparing the bug report provided in Appendix E – Bug Report, I determined the cause of the error to be the process of zipping the file. The software I was using to zip the file was adding extra data into the zip file, which was causing the error. I downloaded new software for zipping files, rezipped the folders with the new software, uploaded the project to App Inventor and then packaged the app for the phone. This time the app packaged correctly and I was able to confirm that the blocks logic was intact and that I had successfully manually merged to projects.

3.1.2 COMMAND LINE MERGE

The next step was to create a command based Java application that could be run through the command line. I started with this limited scope so that I could focus on the back end code before working on the user interface. The most difficult aspect of creating the application was learning how to work with zip files through the command line. After looking at some example code and javadocs I created a simple application, called Merge, that prompts the user for the names of the two project files they would like to merge and the name of the new project that will be created. The application then lists all the files in each of the original projects and prompts the user to enter each file they would like include in the new project. The application then creates a new project file with all the requested files. There is no error testing in this application it simply does the actual act of reading the original zip files and copying that data into a new zip file. The code for Merge is displayed in Appendix F – Merge Code.

3.1.3 AIMERGER

The command line merger completed the tasks I wanted but I also wanted the merger to follow the ideal of App Inventor, that non-technical people can use it. Because of this I wanted to create a merger with a simple graphical user interface (GUI). This became the final step of the merger creation and the AIMerger tool that is currently used by App Inventor users.

3.1.3.1 Scope

For version 1 of the AIMerger the scope was kept simple. The application needed to meet the following specifications:

- Allow the user to select two App Inventor projects that have previously been downloaded from App Inventor to merge.
- Display all assets and screens that are associated with each project selected.
- To deal with Screen1 conflicts the main project's Screen1 would automatically be the Screen1 in the new project. The second project's Screen1 would be grayed out and could not be selected to be merged.
- To deal with duplicate names the user could not select two assets or screens with the same name. They would be given an error and ask to deselect on of the items if they did select two items with the same name.
- The merged project could be saved under a third file name, preventing the user from over-writing one of the two original projects.

I also wanted to keep the GUI simple and similar to that of App Inventor. I used a similar color pallet as used for App Inventor and created a logo using the Android character and the puzzle piece theme. The AIMerger logo is shown below in Figure 8.

Figure 8: AIMerger Logo

3.1.3.2 Interface Implementation

First I sketched out a rough idea for the layout. This sketch is shown below in Figure 9 and is fairly close to the final layout.

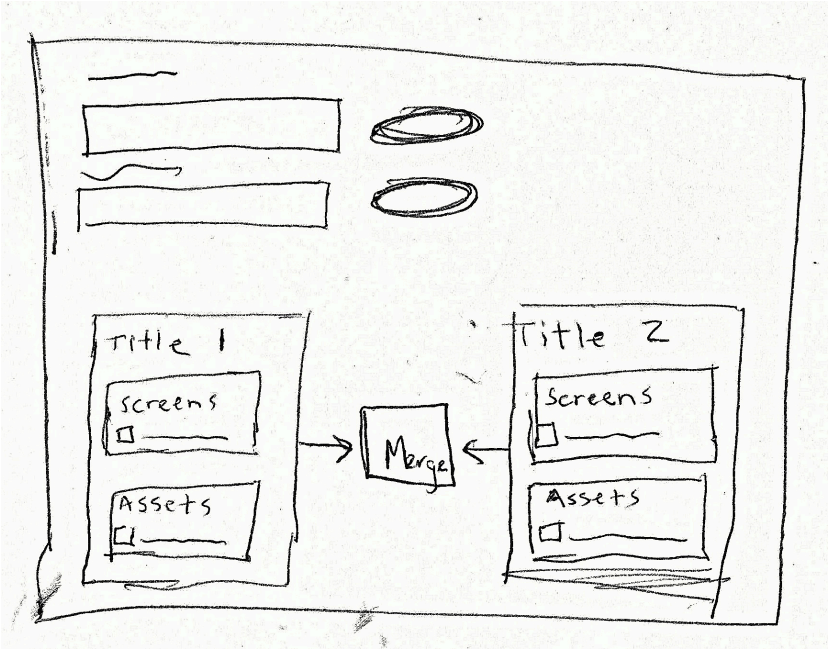


Figure 9: Layout Sketch

The top section would be for browsing for and loading the two original projects. The bottom area would be empty until a project was loaded. Once a project was loaded its name along with a list of assets and a list of screens would appear in the bottom. The main project would be on the left and the second project would be on the right. Each asset and screen listed would have a checkbox next to it. Once both projects were loaded and visible in the lower area, arrows and a merge button would appear between the two projects indicating that it was now possible to merge them.

I decided to build the application's user interface using Java's Swing toolkit. Swing is the primary Java toolkit for building GUIs and includes in its libraries many of the components I needed to implement. I had very limited previous experience with Swing and before I started creating the AIMerger interface I needed to do a number of tutorials [Trail: Graphical User Interfaces (The Java™ Tutorials), 2012] and look through a few example projects.

The code behind the interface is rather simple but implementing it was a very time consuming part of creating the AIMerger. The most difficult part was implementing the checkbox lists for the assets and screens. The Swing library dose not have a built in class that would allow me to create a scrollable list of checkboxes with the names of the screens and assets. I struggled to find a solution to this problem until I found some example code that solved a similar problem [Harmon, 2012]. This solution created a class that customized Swing's list component and I was able to use it as a base to create my own similar class with the required features. The final interface with two projects loaded is shown below in Figure 10.

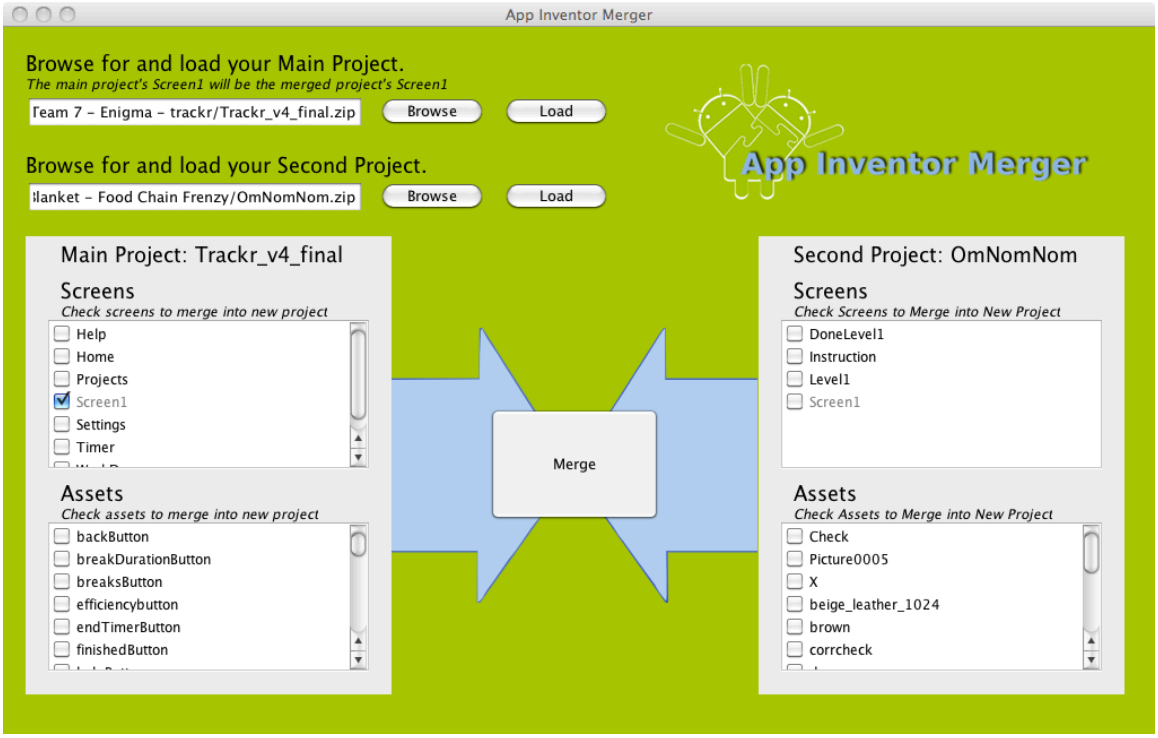


Figure 10: AIMerger Interface

3.1.3.3 Backend Implementation

The structure of the code consists of five classes: AIMerger, AIProject, AIAsset, AIScreen and CheckBoxList. The complete code is available in a public repository at github[4]. The static class diagram is shown in Figure 11.

[4] https://github.com/AIMerger/AIMerger.git

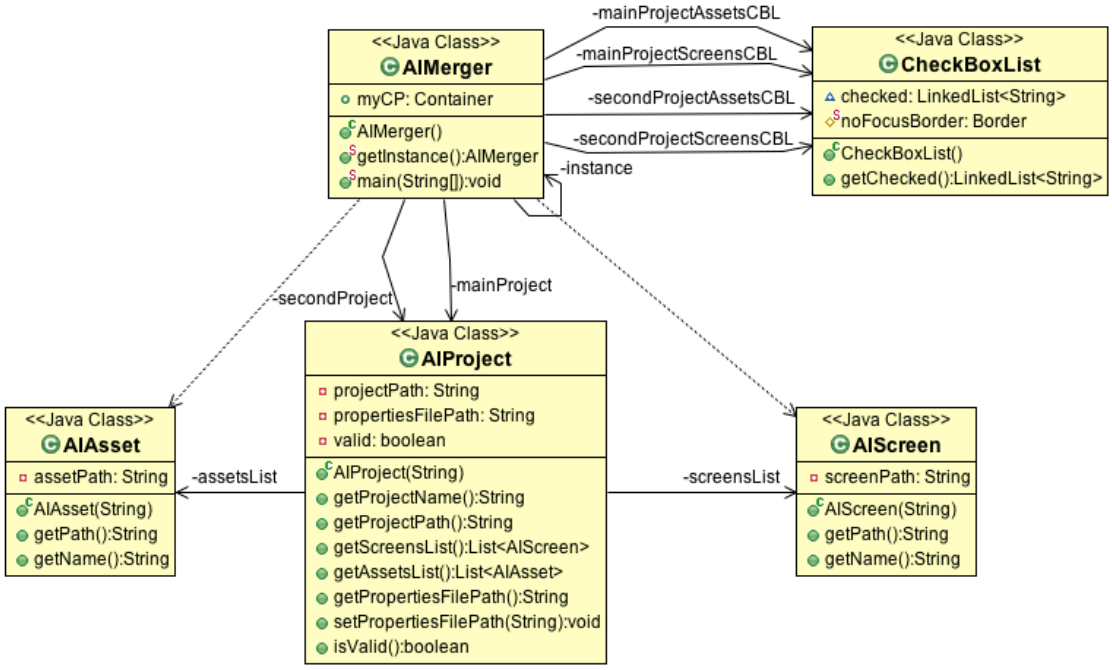


Figure 11: App Inventor Merger Package Diagram

AIMerger is the main class and when the application is launched a new instance of the AIMerger class is created, which displays the startup interface and allow the user to browse for the original projects.

Each time a project is loaded the AIMerger class creates a new instance of the AIProject class. The AIProject class' constructor cycles through all the file in the project zip file and for each file determines if it is a screen, an asset, a project.properties file, or some other file. If the file is a screen then a new instance of the AIScreen class is created with the file's path and it is added to the AIProject's list of screens. If the file is an asset then a new instance of the AIAsset class is created with the file's path and it is added to the AIProject's list of assets. If the file is a project.properties file then its path is assigned to the AIProject field, properitesFilePath. All other files are ignored. The AIMerger class

then displays the screens and assets lists form the AIProject in the lower half of the interface.

Once both original projects are loaded the merge button is made visible. When the merge button is pressed the AIMerger class creates two lists, one for each project, which contain the list of file paths for the files to be merged from the project. These lists are filled by going through the list of checked items from the CheckBoxList and adding each items path. The project.properties file from the main project is also added to the respective list.

The AIMerger class then prompts the user for the path of the new merged project and a new zip file is created using the returned path. Both list of file path are cycled through and each file is written to the new zip file resulting in a new project file with all the selected screens and assets.

The user is then informed that the projects were successfully merged and asked if they would like to merge another project. If the user clicks *no*, then the application is closed. If the user clicks *yes*, then they are asked if they would like to use the new project that was created by the previous merge in the next merge. If the user selects *no*, then the application is restarted as if it were just opening. If the user selects *yes*, then the application is restarted with the new project is displayed as the main project and an AIProject is created.

3.1.3.4 Testing

I tested the application as I finished each section. First, I tested the browse buttons by checking that the following statements were true:

- When the browse button is selected the file selector dialog box appears.
- If the associated text box is empty prior to the browse button being selected,
    - then when a file is selected and the Cancel button is clicked the text box remains empty.
    - then when a file is selected and the Open button is clicked the text box is filled with the absolute path of the file selected.
- If the associated text box is not empty prior to the browse button being selected,
    - then when a file is selected and the Cancel button is clicked the text box remains as it was before the browse button was selected.
    - then when a file is selected and the Open button is clicked the previous content of the text box is replaced with the absolute path of the file selected.

Then I tested the load buttons by checking that the following statements were true:

- If Load is clicked when the associated text box is empty, an error will be given.
- If Load is clicked when the associated text box contains the path to an invalid project (dose not exist, not a zip file, or no properties file), an error will be given.
- If Load is clicked when the associated text box contains the path to a valid project, the project will appear in the lower half of the screen.

- If two valid projects are loaded, the merge button will appear.

I then tested that the projects appeared in the lower section correctly. I did this by downloading the source code for four projects from App Inventor and then loaded these projects into the merger. Once the projects were loaded into the merger I confirmed that all the screens and assets appeared. The four projects were as follows: only screen1 no assets, only screen1 with assets, multiple screens no assets, and multiple screens with assets.

After the user interface appeared to be working properly I tested that the merge process was also working correctly. I did this by creating two projects using the merger and confirmed that they worked properly.

The first project merged together three projects to create one three-screen project. This app's main screen had two buttons and a textbox. When the first button was clicked, screen2 appeared which had an image and a back button. When the back button was clicked the main screen appeared again. When the second button was clicked, screen3 appeared with a textbox and an Ok button. When text was entered in the textbox and Ok was clicked, screen1 reappeared and the textbox on screen1 would then be filled with what was entered in the textbox from screen3. Each screen was originally its own project. This test showed that logic to call a new screen and to pass a value to a screen was still valid after the merge process.

The second project I tested confirmed that merged screens with a shared tinydB would still work properly after the merge. To test this I merged two projects into

one. The first project had a screen1 that had two textboxes and two buttons. The first textbox was for entering the key for the tinydB and the second textbox was for entering the value. The first button submitted the data to the tinydB and the second button switched to screen2. The second project contained screen2. This screen had one textbox, one label and one button. When a key value was entered into the textbox and the button was clicked, the associated value would be retrieved. After merging the two projects, I had one project with two screens. I could submit a value to the tinydB on screen1 then switch to screen2 and retrieve that value. This showed that merging projects that used a shared tinydB was possible with the AIMerger.

I completed all of the above tests on my personal machine running Mac OS X version 10.6.8 and Java 1.6.0_33. Then I opened the merger application and loaded projects using a Windows setup and an Ubuntu setup to confirm that the user interface appeared correctly for both operating systems. Three other members of the App Inventor Developers group using both Mac and Ubuntu systems also tested the application and no issues were discovered.

3.2 DOCUMENTATION

After completing the AIMerger the next step was to create instructions for both using the AIMerger and for developing with App Inventor as a team. This document used an example of two developers working on one app with two screens. It details discussions that need to be had in the team before any

programing is started, how to set up each individual project and then how to merge those projects using the AIMerger. The complete documentation is in Appendix G – AIMerger Documentation.

3.3 BUGS

After releasing the first version of the AIMerger to a limited group of users, I received feedback from one user. He reported two different bugs he discovered while using the AIMerger. The first bug was that when merging a third project, he would sometimes get an error that duplicate assets or screens were selected when it was actually not the case. The other bug was that he was unable to save a new project when running the AIMerger on a PC.

Based on the information provided by the user, I was able to duplicate issues and determine their causes. For the first bug, the issue was that when the AIMerger was reset to merge the third project the lists that held the selected assets and screens were not cleared. The second bug was caused because Windows machines return a path separated by '\' and the merger code expected paths to be separated by '/'. Both of these fixes were simple and required very few changes to the code. I made the changes and released a new version of the AIMerger

3.4 PRESENTATION

During the 2012 App Inventor Summit at MIT I was given the opportunity to present the work I had completed on the AIMerger. I showed a demo of the application and talked about the documentation I had put together. My audience

was mostly comprised of educators who were using App Inventor in their classrooms or programs. The audience also included other developers and users of App Inventor. There was much excitement about the AIMerger and a number of audience members said that this tool would be very helpful in their use of App Inventor.

3.5 Future Improvements

Below is a list of improvements and features I would like to see added to the next version of the App Inventor Merger.

- Show which assets are associated with which screens.
- Not require assets be selected, have assets auto-merged based on selected screens.
- Integrate with App Inventor so that projects don't need to be downloaded from App Inventor, merged and then reloaded. Make it so that the user can select two projects from the projects list in App Inventor and be taken to a merger screen and then the new project is automatically added to the projects list.
- Deal with file name conflicts; allow the user to rename one of the items.
- Let the user select which Screen1 is used in the new project and verify that there is a Screen1.
- Allow user to overwrite one of the original projects with the new project.
- Make dividing projects an explicit option.

3.6 Future Benefits

The AIMerger and subsequent documentation is only the beginning of encouraging team development with App Inventor but it does make it a lot more possible. Most apps are created in teams and it is important that when learning to use App Inventor there are also tools to learn to program as a team. My work will make it easier for programs in which students are working with each other toward a common goal. When only one person in a group can be programming at a time it is easy to rely only on your strongest programmers to do the work, but this leaves behind the rest of the group. By making it possible for more developers to contribute it increases the amount that everyone in the group learns from the project.

4. CONCLUSION

App Inventor is an innovative product that has influenced many people's lives and has the potential to impact many more. It is a great tool for engaging groups that are typically underrepresented in the computer science field and encouraging them to feel as if they can be producers of technology instead of only consumers. I have seen firsthand the impact that this project has made on young women's lives and it is important to me that it continues to broaden its utility.

When App Inventor became an open-source project, it gained the potential to grow exponentially. In order for it to continue to reach its goals, it needs to continue to create and maintain a strong community base. A key to creating a community is to encourage collaboration and share knowledge amongst developers. It must be relatively simple for a developer to get involved with the project and make simple changes in order to relieve stress and allow core developers to make major changes. By developing documentation around the process of adding a property to an existing component, future developers will be able to complete this task and make small but important changes to the project. Having this documentation available to new developers helps them feel supported and motivates them to continue to contribute and to create their own documentation.

App Inventor is used to teach users how to develop their own Android apps as well as to give an introduction to basic computer science concepts. Part of

learning to develop apps is learning to develop as a team. Collaboration is necessary and expected in the computer science workplace and can bring together many different perspectives for the benefit of one project. App Inventor Merger creates a way to develop as a team using App Inventor and allows groups to share programming opportunities. Prior to the merger, only one member of the team could be programming on the project at one time. Now, all members of a team can be programming simultaneously. This keeps everyone engaged with the project and encourages discussion.

More tools are necessary to improve the ability of both developers of App Inventor and developers using App Inventor to collaborate and build community. This thesis has contributed two such changes in the hope that App Inventor will continue to be an engaging, innovative way to introduce computer science concepts to a wide population.

APPENDIX A – SHELL SCRIPTS

buildai.sh – script to build my local version of App Inventor

```
#!/bin/bash

cd /Users/KateFeeney/Documents/workspace/app-inventor/appinventor
ant
```

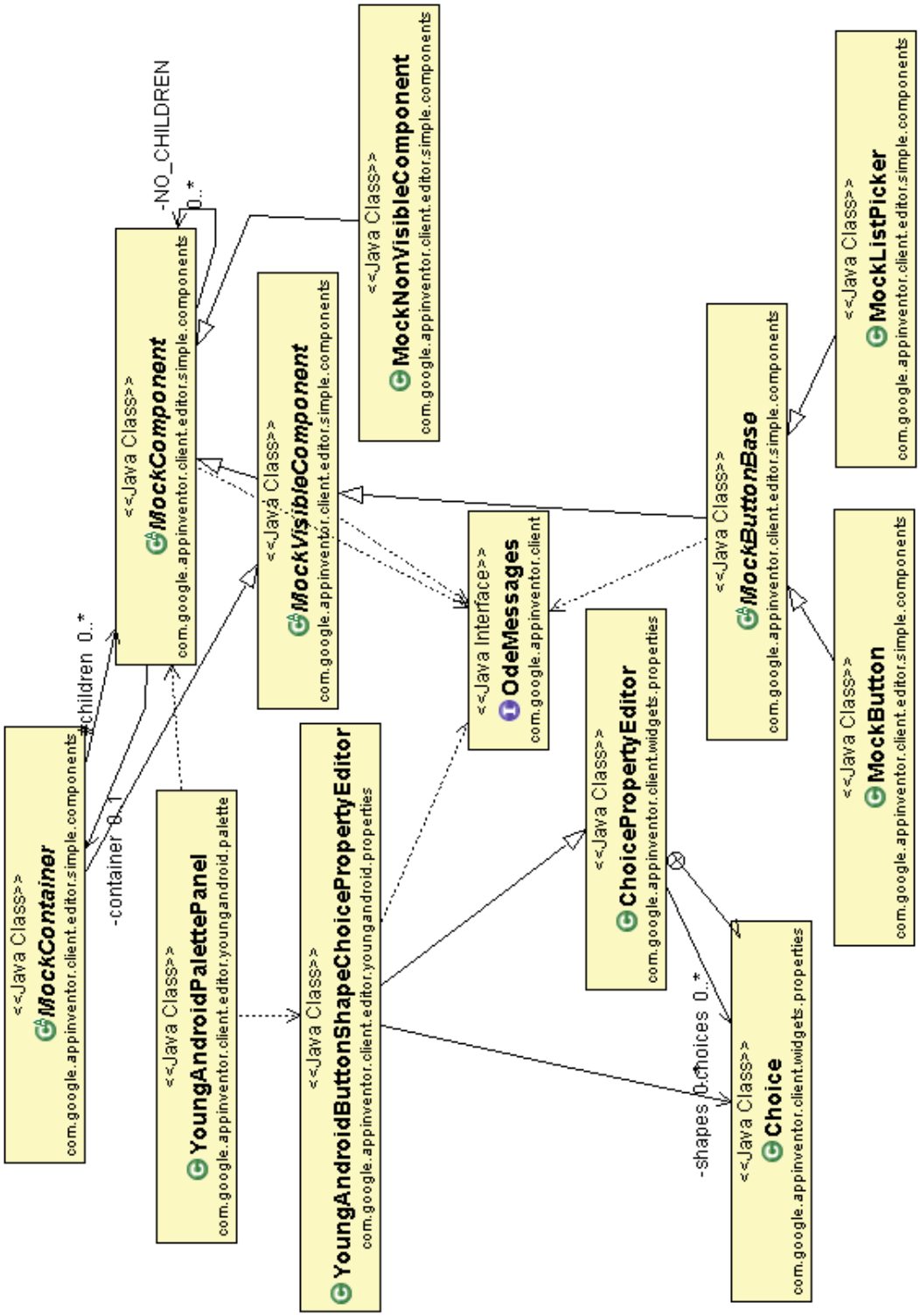
updateai.sh – script to deploy local App Inventor to App Engine

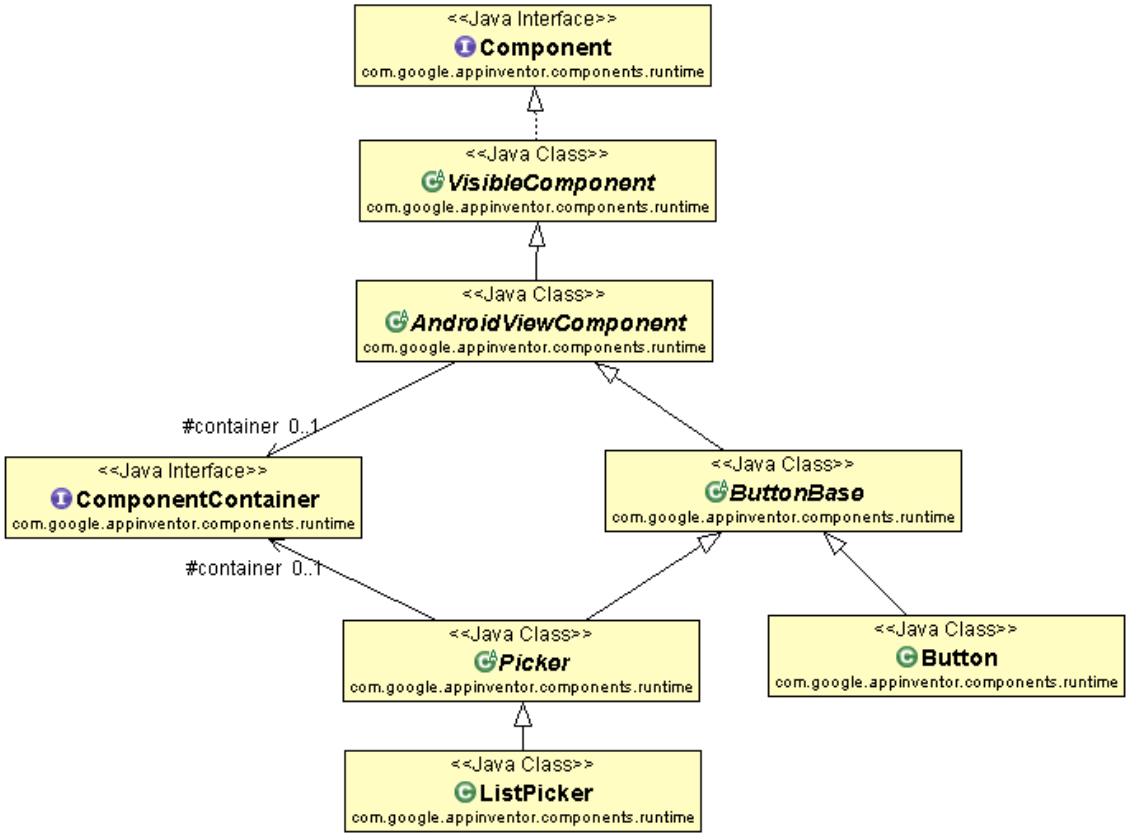
```
#!/bin/bash

cd /Users/KateFeeney/Documents/workspace/app-
inventor/appinventor/appengine/build

java -cp
/Applications/eclipse/plugins/com.google.appengine.eclipse.sdkbundle_1.
6.3.v201202290255r37/appengine-java-sdk-1.6.3/lib/appengine-tools-
api.jar com.google.appengine.tools.admin.AppCfg -A feeney-ka update war
```

rollbackai.sh – script to clear a previous update that did not successfully complete

```
#!/bin/bash

cd
/Users/KateFeeney/Documents/School/Mills/Thesis/AddingAProperty/ShowPro
p/app-inventor/appinventor/appengine/build
java -cp
/Applications/eclipse/plugins/com.google.appengine.eclipse.sdkbundle_1.
6.3.v201202290255r37/appengine-java-sdk-1.6.3/lib/appengine-tools-
api.jar com.google.appengine.tools.admin.AppCfg -A feeney-ka rollback
war
```

APPENDIX B – STATIC CLASS DIAGRAMS




<<Java Interface>>
Component
com.google.appinventor.components.runtime
<<Java Class>>
VisibleComponent
com.google.appinventor.components.runtime
<<Java Class>>
AndroidViewComponent
com.google.appinventor.components.runtime
#container 0..1
<<Java Interface>>
ComponentContainer
com.google.appinventor.components.runtime
<<Java Class>>
ButtonBase
com.google.appinventor.components.runtime
#container 0..1
<<Java Class>>
Picker
com.google.appinventor.components.runtime
<<Java Class>>
Button
com.google.appinventor.components.runtime
<<Java Class>>
ListPicker
com.google.appinventor.components.runtime

APPENDIX C – DIFF FILE FOR CL1

```
diff -r 6a6acbed906d
appinventor/appengine/src/com/google/appinventor/client/OdeMessages.java
--- a/appinventor/appengine/src/com/google/appinventor/client/OdeMessages.java
+++ b/appinventor/appengine/src/com/google/appinventor/client/OdeMessages.java
@@ -358,6 +358,24 @@
   @Description("Text for text alignment choice 'right'")
   String rightTextAlignment();

+  // Used in
editor/youngandroid/properties/YoungAndroidButtonShapeChoicePropertyEditor.java
+
+  @DefaultMessage("default")
+  @Description("Text for button shape choice 'default'")
+  String defaultButtonShape();
+
+  @DefaultMessage("rounded")
+  @Description("Text for button shape choice 'rounded'")
+  String roundedButtonShape();
+
+  @DefaultMessage("rectangular")
+  @Description("Text for button shape choice 'rectangular'")
+  String rectButtonShape();
+
+  @DefaultMessage("oval")
+  @Description("Text for button shape choice 'oval'")
+  String ovalButtonShape();
+

diff -r 6a6acbed906d
appinventor/appengine/src/com/google/appinventor/client/editor/simple/component
s/MockButtonBase.java
---
a/appinventor/appengine/src/com/google/appinventor/client/editor/simple/compone
nts/MockButtonBase.java
+++
b/appinventor/appengine/src/com/google/appinventor/client/editor/simple/compone
nts/MockButtonBase.java
@@ -31,9 +31,12 @@
   private String imagePropValue;
   private boolean hasImage;

-  // We need to maintain this so we can show color only when
+  // We need to maintain these so we can show color and shape only when
   // there is no image.
   private String backgroundColor;
+  // Legal values for shape are defined in
+  // com.google.appinventor.components.runtime.Component.java.
+  private int shape;

   /**
    * Creates a new MockButtonBase component.
@@ -103,7 +106,45 @@
   private void setTextAlignmentProperty(String text) {
     MockComponentsUtil.setWidgetTextAlign(buttonWidget, text);
   }
+  /*
+   * Sets the button's Shape property to a new value.
+   */
+  private void setShapeProperty(String text) {
+    shape = Integer.parseInt(text);
+    // Android Buttons with images take the shape of the image and do not
```

```
+    // use one of the defined Shapes.
+    if (hasImage) {
+      return;
+    }
+    switch(shape) {
+      case 0:
+        // Default Button
+        DOM.setStyleAttribute(buttonWidget.getElement(), "border-radius",
"0px");
+        break;
+      case 1:
+        // Rounded Button.
+        // The corners of the Button are rounded by 10 px.
+        // The value 10 px was chosen strictly for style.
+        // 10 px is the same as ROUNDED_CORNERS_RADIUS defined in
+        // com.google.appinventor.components.runtime.ButtonBase.
+        DOM.setStyleAttribute(buttonWidget.getElement(), "border-radius",
"10px");
+        break;
+      case 2:
+        // Rectangular Button
+        DOM.setStyleAttribute(buttonWidget.getElement(), "border-radius",
"0px");
+        break;
+      case 3:
+        // Oval Button
+        String height = DOM.getStyleAttribute(buttonWidget.getElement(),
"height");
+        DOM.setStyleAttribute(buttonWidget.getElement(), "border-radius",
height);
+        break;
+      default:
+        // This should never happen
+        throw new IllegalArgumentException("shape:" + shape);
+    }
+  }
+
   /*
    * Sets the button's BackgroundColor property to a new value.
    */
@@ -169,6 +210,7 @@
       hasImage = false;
       url = "";
       setBackgroundColorProperty(backgroundColor);
+      setShapeProperty(Integer.toString(shape));
     } else {
       hasImage = true;
       // Android Buttons do not show a background color if they have an image.
@@ -177,6 +219,7 @@
       // setting the widget's background color to COLOR_NONE.
       MockComponentsUtil.setWidgetBackgroundColor(buttonWidget,
           "&H" + COLOR_NONE);
+      DOM.setStyleAttribute(buttonWidget.getElement(), "border-radius",
"0px");
     }
     MockComponentsUtil.setWidgetBackgroundImage(buttonWidget, url);
     image.setUrl(url);
@@ -269,6 +312,8 @@
       refreshForm();
     } else if (propertyName.equals(PROPERTY_NAME_TEXTCOLOR)) {
       setTextColorProperty(newValue);
+    } else if (propertyName.equals(PROPERTY_NAME_BUTTONSHAPE)){
+      setShapeProperty(newValue);
     }
   }
```

```
 }
diff -r 6a6acbed906d
appinventor/appengine/src/com/google/appinventor/client/editor/simple/component
s/MockVisibleComponent.java
---
a/appinventor/appengine/src/com/google/appinventor/client/editor/simple/compone
nts/MockVisibleComponent.java
+++
b/appinventor/appengine/src/com/google/appinventor/client/editor/simple/compone
nts/MockVisibleComponent.java
@@ -19,6 +19,7 @@

   // Common property names (not all components support all properties).
   protected static final String PROPERTY_NAME_TEXTALIGNMENT = "TextAlignment";
+  protected static final String PROPERTY_NAME_BUTTONSHAPE= "Shape";
   protected static final String PROPERTY_NAME_BACKGROUNDCOLOR =
"BackgroundColor";
   protected static final String PROPERTY_NAME_BACKGROUNDIMAGE =
"BackgroundImage";
   protected static final String PROPERTY_NAME_ENABLED = "Enabled";
diff -r 6a6acbed906d
appinventor/appengine/src/com/google/appinventor/client/editor/youngandroid/pal
ette/YoungAndroidPalettePanel.java
---
a/appinventor/appengine/src/com/google/appinventor/client/editor/youngandroid/p
alette/YoungAndroidPalettePanel.java
+++
b/appinventor/appengine/src/com/google/appinventor/client/editor/youngandroid/p
alette/YoungAndroidPalettePanel.java
@@ -13,6 +13,7 @@
 import
com.google.appinventor.client.editor.youngandroid.properties.YoungAndroidAlignm
entChoicePropertyEditor;
 import
com.google.appinventor.client.editor.youngandroid.properties.YoungAndroidAssetS
electorPropertyEditor;
 import
com.google.appinventor.client.editor.youngandroid.properties.YoungAndroidBoolea
nPropertyEditor;
+import
com.google.appinventor.client.editor.youngandroid.properties.YoungAndroidButton
ShapeChoicePropertyEditor;
 import
com.google.appinventor.client.editor.youngandroid.properties.YoungAndroidColorC
hoicePropertyEditor;
 import
com.google.appinventor.client.editor.youngandroid.properties.YoungAndroidCompon
entSelectorPropertyEditor;
 import
com.google.appinventor.client.editor.youngandroid.properties.YoungAndroidFontTy
pefaceChoicePropertyEditor;
@@ -169,6 +170,8 @@
       return new YoungAndroidAlignmentChoicePropertyEditor();
     } else if (editorType.equals("typeface")) {
       return new YoungAndroidFontTypefaceChoicePropertyEditor();
+    } else if (editorType.equals("buttonshape")){
+      return new YoungAndroidButtonShapeChoicePropertyEditor();
     } else {
       return new TextPropertyEditor();
     }
diff -r 6a6acbed906d
appinventor/appengine/src/com/google/appinventor/client/editor/youngandroid/pro
perties/YoungAndroidButtonShapeChoicePropertyEditor.java
--- /dev/null
```

```
+++
b/appinventor/appengine/src/com/google/appinventor/client/editor/youngandroid/p
roperties/YoungAndroidButtonShapeChoicePropertyEditor.java
@@ -0,0 +1,24 @@
+package com.google.appinventor.client.editor.youngandroid.properties;
+
+import static com.google.appinventor.client.Ode.MESSAGES;
+import com.google.appinventor.client.widgets.properties.ChoicePropertyEditor;
+
+/**
+ * Property editor for button shape.
+ *
+ * @author feeney.kate@gmail.com (Kate Feeney)
+ */
+public class YoungAndroidButtonShapeChoicePropertyEditor extends
ChoicePropertyEditor {
+
+  // Button shape choices
+  private static final Choice[] shapes = new Choice[] {
+    new Choice(MESSAGES.defaultButtonShape(), "0"),
+    new Choice(MESSAGES.roundedButtonShape(), "1"),
+    new Choice(MESSAGES.rectButtonShape(), "2"),
+    new Choice(MESSAGES.ovalButtonShape(), "3")
+  };
+
+  public YoungAndroidButtonShapeChoicePropertyEditor() {
+    super(shapes);
+  }
+}
diff -r 6a6acbed906d
appinventor/appengine/src/com/google/appinventor/client/youngandroid/YoungAndro
idFormUpgrader.java
---
a/appinventor/appengine/src/com/google/appinventor/client/youngandroid/YoungAnd
roidFormUpgrader.java
+++
b/appinventor/appengine/src/com/google/appinventor/client/youngandroid/YoungAnd
roidFormUpgrader.java
@@ -418,6 +418,11 @@
       // No properties need to be modified to upgrade to version 3.
       srcCompVersion = 3;
     }
+    if (srcCompVersion < 4) {
+      // The Shape property was added.
+      // No properties need to be modified to upgrade to version 4.
+      srcCompVersion = 4;
+    }
     return srcCompVersion;
   }

@@ -471,6 +476,11 @@
       // The Open method was added.  No changes are needed.
       srcCompVersion = 3;
     }
+    if (srcCompVersion < 4) {
+      // The Shape property was added.
+      // No properties need to be modified to upgrade to version 4.
+      srcCompVersion = 4;
+    }
     return srcCompVersion;
   }

@@ -551,6 +561,11 @@
       // The Open method was added.  No changes are needed.
       srcCompVersion = 3;
```

```
     }
+    if (srcCompVersion < 4) {
+      // The Shape property was added.
+      // No properties need to be modified to upgrade to version 4.
+      srcCompVersion = 4;
+    }
     return srcCompVersion;
   }

@@ -599,6 +614,11 @@
       // The Open method was added.  No changes are needed.
       srcCompVersion = 4;
     }
+    if (srcCompVersion < 5) {
+      // The Shape property was added.
+      // No properties need to be modified to upgrade to version 5.
+      srcCompVersion = 5;
+    }
     return srcCompVersion;
   }

@@ -636,6 +656,11 @@
       // The Open method was added.  No changes are needed.
       srcCompVersion = 3;
     }
+    if (srcCompVersion < 4) {
+      // The Shape property was added.
+      // No properties need to be modified to upgrade to version 4.
+      srcCompVersion = 4;
+    }
     return srcCompVersion;
   }

diff -r 6a6acbed906d
appinventor/blockslib/src/openblocks/yacodeblocks/BlockSaveFile.java
--- a/appinventor/blockslib/src/openblocks/yacodeblocks/BlockSaveFile.java
+++ b/appinventor/blockslib/src/openblocks/yacodeblocks/BlockSaveFile.java
@@ -604,6 +604,11 @@
       // No blocks need to be modified to upgrade to version 3.
       blkCompVersion = 3;
     }
+    if (blkCompVersion < 4) {
+      // The Shape property was added.
+      // No blocks need to be modified to upgrade to version 4.
+      blkCompVersion = 4;
+    }
     return blkCompVersion;
   }

@@ -654,6 +659,11 @@
       // The Open method was added, which does not require changes.
       blkCompVersion = 3;
     }
+    if (blkCompVersion < 4) {
+      // The Shape property was added.
+      // No blocks need to be modified to upgrade to version 4.
+      blkCompVersion = 4;
+    }
     return blkCompVersion;
   }

@@ -708,6 +718,11 @@
       // The Open method was added, which does not require changes.
       blkCompVersion = 3;
     }
```

```
+    if (blkCompVersion < 4) {
+      // The Shape property was added.
+      // No blocks need to be modified to upgrade to version 4.
+      blkCompVersion = 4;
+    }
     return blkCompVersion;
   }

@@ -755,6 +770,11 @@
       // The Open method was added, which does not require changes.
       blkCompVersion = 4;
     }
+    if (blkCompVersion < 5) {
+      // The Shape property was added.
+      // No blocks need to be modified to upgrade to version 5.
+      blkCompVersion = 5;
+    }
     return blkCompVersion;
   }

@@ -795,6 +815,11 @@
       // The Open method was added, which does not require changes.
       blkCompVersion = 3;
     }
+    if (blkCompVersion < 4) {
+      // The Shape property was added.
+      // No blocks need to be modified to upgrade to version 4.
+      blkCompVersion = 4;
+    }
     return blkCompVersion;
   }

diff -r 6a6acbed906d
appinventor/components/src/com/google/appinventor/components/annotations/Design
erProperty.java
---
a/appinventor/components/src/com/google/appinventor/components/annotations/Desi
gnerProperty.java
+++
b/appinventor/components/src/com/google/appinventor/components/annotations/Desi
gnerProperty.java
@@ -37,6 +37,7 @@
   static final String PROPERTY_TYPE_TEXT = "text";
   static final String PROPERTY_TYPE_TEXTALIGNMENT = "textalignment";
   static final String PROPERTY_TYPE_TYPEFACE = "typeface";
+  static final String PROPERTY_TYPE_BUTTON_SHAPE = "buttonshape";

   /**
    * Determines the property editor used in the designer.
diff -r 6a6acbed906d
appinventor/components/src/com/google/appinventor/components/common/YaVersion.j
ava
---
a/appinventor/components/src/com/google/appinventor/components/common/YaVersion
.java
+++
b/appinventor/components/src/com/google/appinventor/components/common/YaVersion
.java
@@ -162,8 +162,14 @@
   // For YOUNG_ANDROID_VERSION 53:
   // - BLUETOOTHCLIENT_COMPONENT_VERSION was incremented to 5.
   // - BLUETOOTHSERVER_COMPONENT_VERSION was incremented to 5.
+  // For YOUNG_ANDROID_VERSION 54:
+  // - BUTTON_COMPONENT_VERSION was incremented to 4.
+  // - CONTACTPICKER_COMPONENT_VERSION was incremented to 4.
```

```
+  // - IMAGEPICKER_COMPONENT_VERSION was incremented to 4.
+  // - LISTPICKER_COMPONENT_VERSION was incremented to 5.
+  // - PHONENUMBERPICKER_COMPONENT_VERSION was incremented to 4.

-  public static final int YOUNG_ANDROID_VERSION = 53;
+  public static final int YOUNG_ANDROID_VERSION = 54;

   // ............................... Blocks Language Version
Number ...............................

@@ -275,7 +281,9 @@
   // - The Alignment property was renamed to TextAlignment.
   // For BUTTON_COMPONENT_VERSION 3:
   // - The LongClick event was added.
-  public static final int BUTTON_COMPONENT_VERSION = 3;
+  // For BUTTON_COMPONENT_VERSION 4:
+  // - The Shape property was added.
+  public static final int BUTTON_COMPONENT_VERSION = 4;

   public static final int CAMERA_COMPONENT_VERSION = 1;

@@ -302,7 +310,9 @@
   // - The Alignment property was renamed to TextAlignment.
   // For CONTACTPICKER_COMPONENT_VERSION 3:
   // - The method Open was added.
-  public static final int CONTACTPICKER_COMPONENT_VERSION = 3;
+  // For CONTACTPICKER_COMPONENT_VERSION 4:
+  // - The Shape property was added.
+  public static final int CONTACTPICKER_COMPONENT_VERSION = 4;

   // For EMAILPICKER_COMPONENT_VERSION 2:
   // - The Alignment property was renamed to TextAlignment.
@@ -333,7 +343,9 @@
   // - The Alignment property was renamed to TextAlignment.
   // For IMAGEPICKER_COMPONENT_VERSION 3:
   // - The method Open was added.
-  public static final int IMAGEPICKER_COMPONENT_VERSION = 3;
+  // For IMAGEPICKER_COMPONENT_VERSION 4:
+  // - The Shape property was added.
+  public static final int IMAGEPICKER_COMPONENT_VERSION = 4;

   // For IMAGESPRITE_COMPONENT_VERSION 2:
   // - The Rotates property was added.
@@ -368,7 +380,9 @@
   // - The SelectionIndex read-write property was added.
   // For LISTPICKER_COMPONENT_VERSION 4:
   // - The method Open was added.
-  public static final int LISTPICKER_COMPONENT_VERSION = 4;
+  // For LISTPICKER_COMPONENT_VERSION 5:
+  // - The Shape property was added.
+  public static final int LISTPICKER_COMPONENT_VERSION = 5;

   public static final int LOCATIONSENSOR_COMPONENT_VERSION = 1;

@@ -391,7 +405,9 @@
   // - The Alignment property was renamed to TextAlignment.
   // For PHONENUMBERPICKER_COMPONENT_VERSION 3:
   // - The method Open was added.
-  public static final int PHONENUMBERPICKER_COMPONENT_VERSION = 3;
+  // For PHONENUMBERPICKER_COMPONENT_VERSION 4:
+  // - The Shape property was added.
+  public static final int PHONENUMBERPICKER_COMPONENT_VERSION = 4;

   // For PLAYER_COMPONENT_VERSION 2:
   // - The Player.PlayerError event was added.
```

```
diff -r 6a6acbed906d
appinventor/components/src/com/google/appinventor/components/runtime/ButtonBase
.java
---
a/appinventor/components/src/com/google/appinventor/components/runtime/ButtonBa
se.java
+++
b/appinventor/components/src/com/google/appinventor/components/runtime/ButtonBa
se.java
@@ -13,7 +13,12 @@
 import com.google.appinventor.components.runtime.util.ViewUtil;
 import android.content.res.ColorStateList;
+import android.graphics.Color;
 import android.graphics.drawable.Drawable;
+import android.graphics.drawable.ShapeDrawable;
+import android.graphics.drawable.shapes.OvalShape;
+import android.graphics.drawable.shapes.RectShape;
+import android.graphics.drawable.shapes.RoundRectShape;
 import android.util.Log;
 import android.view.View;
 import android.view.View.OnClickListener;
@@ -34,7 +39,19 @@
   private static final String LOG_TAG = "ButtonBase";

   private final android.widget.Button view;
+  // Constant for shape
+  // 10px is the radius of the rounded corners.
+  // 10px was chosen for esthetic reasons.
+  private static final float ROUNDED_CORNERS_RADIUS = 10f;
+  private static final float[] ROUNDED_CORNERS_ARRAY = new float[]
{ ROUNDED_CORNERS_RADIUS,
+      ROUNDED_CORNERS_RADIUS, ROUNDED_CORNERS_RADIUS, ROUNDED_CORNERS_RADIUS,
+      ROUNDED_CORNERS_RADIUS, ROUNDED_CORNERS_RADIUS, ROUNDED_CORNERS_RADIUS,
+      ROUNDED_CORNERS_RADIUS };
+
+  // Constant background color for buttons with a Shape other than default
+  private static final int SHAPED_DEFAULT_BACKGROUND_COLOR = Color.LTGRAY;
+
   // Backing for text alignment
   private int textAlignment;

@@ -53,6 +70,9 @@
   // Backing for text color
   private int textColor;

+  // Backing for button shape
+  private int shape;
+
   // Image path
   private String imagePath = "";

@@ -101,6 +121,7 @@
     FontSize(Component.FONT_DEFAULT_SIZE);
     Text("");
     TextColor(Component.COLOR_DEFAULT);
+    Shape(Component.BUTTON_SHAPE_DEFAULT);
   }

   @Override
@@ -160,6 +181,39 @@
   }

   /**
+   * Returns the shape of the button.
+   *
```

```
+   * @return  one of {@link Component#BUTTON_SHAPE_DEFAULT},
+   *          {@link Component#BUTTON_SHAPE_ROUNDED},
+   *          {@link Component#BUTTON_SHAPE_RECT} or
+   *          {@link Component#BUTTON_SHAPE_OVAL}
+   */
+  @SimpleProperty(
+      category = PropertyCategory.APPEARANCE,
+      userVisible = false)
+  public int Shape() {
+    return shape;
+  }
+
+  /**
+   * Specifies the shape the button. This does not check that the argument is
a legal value.
+   *
+   * @param shape one of {@link Component#BUTTON_SHAPE_DEFAULT},
+   *          {@link Component#BUTTON_SHAPE_ROUNDED},
+   *          {@link Component#BUTTON_SHAPE_RECT} or
+   *          {@link Component#BUTTON_SHAPE_OVAL}
+   *
+   * @throws IllegalArgumentException if shape is not a legal value.
+   */
+  @DesignerProperty(editorType = DesignerProperty.PROPERTY_TYPE_BUTTON_SHAPE,
+      defaultValue = Component.BUTTON_SHAPE_DEFAULT + "")
+  @SimpleProperty(userVisible = false)
+  public void Shape(int shape) {
+    this.shape = shape;
+    updateAppearance();
+  }
+
+  /**
    * Returns the path of the button's image.
    *
    * @return  the path of the button's image
@@ -240,25 +294,59 @@
     updateAppearance();
   }

-  // Update appearance based on values of backgroundImageDrawable and
backgroundColor.
+  // Update appearance based on values of backgroundImageDrawable,
backgroundColor and shape.
   // Images take precedence over background colors.
   private void updateAppearance() {
-    // If there is no background image, the appearance depends solely on the
background color.
+    // If there is no background image,
+    // the appearance depends solely on the background color and shape.
     if (backgroundImageDrawable == null) {
-      if (backgroundColor == Component.COLOR_DEFAULT) {
-        // Restore original 3D bevel appearance.
-        ViewUtil.setBackgroundDrawable(view, defaultButtonDrawable);
+      if (shape == Component.BUTTON_SHAPE_DEFAULT) {
+        if (backgroundColor == Component.COLOR_DEFAULT) {
+          // If there is no background image and color is default,
+          // restore original 3D bevel appearance.
+          ViewUtil.setBackgroundDrawable(view, defaultButtonDrawable);
+        } else {
+          // Clear the background image.
+          ViewUtil.setBackgroundDrawable(view, null);
+          // Set to the specified color (possibly COLOR_NONE for transparent).
+          TextViewUtil.setBackgroundColor(view, backgroundColor);
+        }
       } else {
```

```
-        // Clear the background image.
-        ViewUtil.setBackgroundDrawable(view, null);
-        // Set to the specified color (possibly COLOR_NONE for transparent).
-        TextViewUtil.setBackgroundColor(view, backgroundColor);
+        // If there is no background image and the shape is something other
than default,
+        // create a drawable with the appropriate shape and color.
+        setShape();
       }
-      return;
+    } else {
+      // If there is a background image
+      ViewUtil.setBackgroundImage(view, backgroundImageDrawable);
     }
-    ViewUtil.setBackgroundImage(view, backgroundImageDrawable);
   }
+  // Throw IllegalArgumentException if shape has illegal value.
+  private void setShape() {
+    ShapeDrawable drawable = new ShapeDrawable();
+    // Set color of drawable.
+    drawable.getPaint().setColor((backgroundColor == Component.COLOR_DEFAULT)
+                                 ? SHAPED_DEFAULT_BACKGROUND_COLOR :
backgroundColor);
+    // Set shape of drawable.
+    switch (shape) {
+      case Component.BUTTON_SHAPE_ROUNDED:
+        drawable.setShape(new RoundRectShape(ROUNDED_CORNERS_ARRAY, null,
null));
+        break;
+      case Component.BUTTON_SHAPE_RECT:
+        drawable.setShape(new RectShape());
+        break;
+      case Component.BUTTON_SHAPE_OVAL:
+        drawable.setShape(new OvalShape());
+        break;
+      default:
+        throw new IllegalArgumentException();
+    }
+    // Set drawable to the background of the button.
+    view.setBackgroundDrawable(drawable);
+    view.invalidate();
+  }
+
   /**
    * Returns true if the button is active and clickable.
    *
diff -r 6a6acbed906d
appinventor/components/src/com/google/appinventor/components/runtime/Component.
java
---
a/appinventor/components/src/com/google/appinventor/components/runtime/Componen
t.java
+++
b/appinventor/components/src/com/google/appinventor/components/runtime/Componen
t.java
@@ -23,7 +23,15 @@
   static final int ALIGNMENT_NORMAL = 0;
   static final int ALIGNMENT_CENTER = 1;
   static final int ALIGNMENT_OPPOSITE = 2;
-
+
+  /*
+   * Button shape.
+   */
+  static final int BUTTON_SHAPE_DEFAULT = 0;
```

```
+  static final int BUTTON_SHAPE_ROUNDED = 1;
+  static final int BUTTON_SHAPE_RECT = 2;
+  static final int BUTTON_SHAPE_OVAL = 3;
+
   /*
    * Color constants.
    */
```

APPENDIX D – HOW TO ADD A PROPERTY TO A COMPONENT

How to Add a Property to a Component

Created by Kate Feeney

This document describes how to add a new property to an existing App Inventor component.

As an example, we will show how the Shape property was added to the ButtonBase component. ButtonBase is an abstract superclass of the Button and the Picker components (ContactPicker, ImagePicker and ListPicker); therefore, all properties defined for ButtonBase are also defined for Button and Picker. Originally the ButtonBase component had no Shape property; it simply used the default shape, which uses the system's defaults and varies from device to device. When the Shape property was added, four choices (default, rounded, rectangular and oval) were included. All examples in this document show the Button component but would be the same for any of the Picker components.

The user will first see the new property in the Button component's Properties panel in the Designer. Because the choices should be restricted to the four legal values, we will create a PropertyEditor that limits the choices to these values and maps them to integers

for the internal representation of the property. When the user changes the value of the Shape property, the visual representation of the Button component in the Designer must change so that the user can preview the interface. To do this we will create a method that will change the attributes of the GWT button widget that represents the Button component in the Designer. Finally, since the user ultimately wants the property changed on their Android device we will add code that changes the Button view's BackgroundDrawable depending on which Shape value is selected.

1. Adding a Property to the Properties Panel

The first step is to add the property so that it appears in the Properties panel when a Button is selected. This change is shown in the image below. Every property is associated with a PropertyEditor to allow the user to choose among legal values. Often, an existing PropertyEditor can be used, but in some cases, such as the Shape property, it will be necessary to create a new one. Finally, we will associate the property with the component so that the property will appear in the Properties panel when the component is selected.

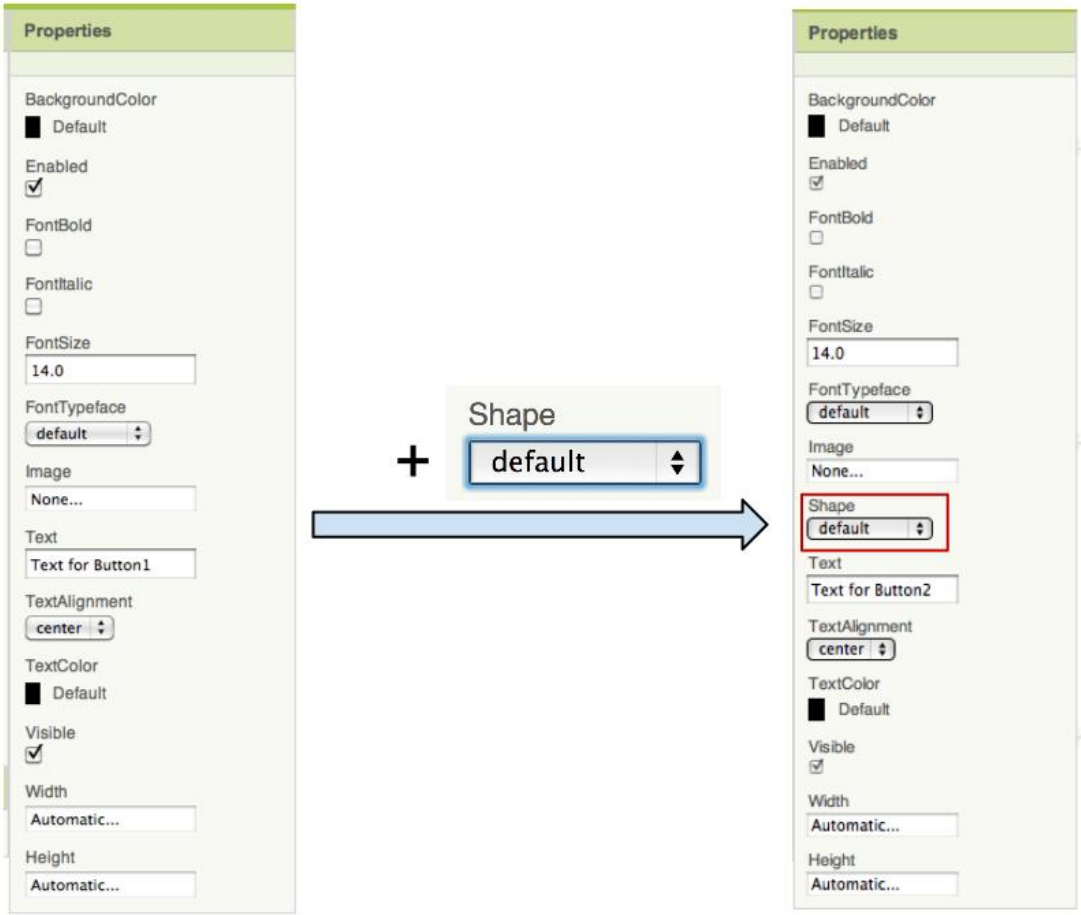


Figure 1: Button Property Panel

1.1 PropertyEditors

Each property has a PropertyEditor that controls what values can be specified in the Designer. Some existing PropertyEditors include the BooleanPropertyEditor (used by ButtonBase.Enabled), NonNegativeFloatPropertyEditor (used by ButtonBase.FontSize) and TextPropertyEditor (used by ButtonBase.Text). If a suitable PropertyEditor already exists (Check if an existing property will meet your needs.), then simply note its name and skip to Section 1.3. Otherwise a new PropertyEditor needs to be created as described in Section 1.2.

The PropertyEditor associated with the Shape property must offer a drop-down menu with the four legal shape values: default, rounded, rectangular and oval (as shown below). Since this does not already exist, a new PropertyEditor must be created.

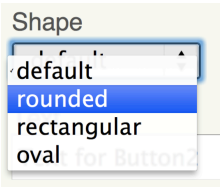


Figure 2: Shape PropertyEditor in the Designer

1.2 Creating a New PropertyEditor

1.2.1 Creating the new PropertyEditor class

The new PropertyEditor class must extend the PropertyEditor class and restrict the user inputs to only legal values. This class will also define how the PropertyEditor is displayed to the user (drop-down menu, text box, etc.).

For our example the new class will be called YoungAndroidButtonShapeChoicePropertyEditor and it will extend the ChoicePropertyEditor class, which itself extends PropertyEditor. This new class must define an array of Choice objects and pass the array to the ChoicePropertyEditor constructor, which will create the drop-down choice widget. A Choice is a static class defined in the ChoicePropertyEditor class and its constructor takes in two strings: `caption` and `value`. The `caption` string is text to be shown in the drop-down choice widget. The `value` string is the value assigned to the property if the choice is selected. The new class is defined in Figure 4.

The first step is to define the four descriptive strings (`caption` strings), which are displayed to the user and will be placed in the array passed to the ChoicePropertyEditor. This is done by adding the following code to the OdeMessages interface. The reason the strings are defined in a separate file rather than hard-coded is to support abstraction and internationalization.

```
//Used in
editor/youngandroid/properties/YoungAndroidButtonShapeChoicePropertyEditor.java
```

```
@DefaultMessage("default")                          ← this is what will be displayed to the user
@Description("Text for button shape choice 'default'")   ← this string is information for a translator
String defaultButtonShape();                        ← the name of the descriptive string is made up at this point

@DefaultMessage("rounded")
@Description("Text for button shape choice 'rounded'")
String roundedButtonShape();

@DefaultMessage("rectangular")
@Description("Text for button shape choice 'rectangular'")
String rectButtonShape();

@DefaultMessage("oval")
@Description("Text for button shape choice 'oval'")
String ovalButtonShape();
```

Figure 3: Define Strings in OdeMessages class

Now that the strings are defined, the YoungAndroidButtonShapeEditor class can be created as shown in Figure 4.

```
package com.google.appinventor.client.editor.youngandroid.properties;

import static com.google.appinventor.client.Ode.MESSAGES;
import com.google.appinventor.client.widgets.properties.ChoicePropertyEditor;

/**
 * Property editor for button shape.
 *
 * @author feeney.kate@gmail.com (Kate Feeney)
 */
public class YoungAndroidButtonShapeChoicePropertyEditor extends
ChoicePropertyEditor {                                          ← extend ChoicePropertyEditor
  // Button shape choices
  private static final Choice[] shapes = new Choice[] {
    new Choice(MESSAGES.defaultButtonShape(), "0"),
    new Choice(MESSAGES.roundedButtonShape(), "1")
    new Choice(MESSAGES.rectButtonShape(), "2"),        ← define an array of the choices
    new Choice(MESSAGES.ovalButtonShape(), "3")
  };

  public YoungAndroidButtonShapeChoicePropertyEditor() {
   super(shapes);                                       ← pass array to the ChoicePropertyEditor constructor
  }
}
```

Figure 4: YoungAndroidButtonShapeEditor class

The `value` strings (e.g., “0”, “1”, “2”, “3” in Figure 4) will be assigned to the property and can be accessed via the component’s getters and setters. When the string is retrieved by the component’s getter it will be retrieved as an int for which constants should be defined in the Component class. We will define the constants in the Component class since all components inherit from it. Therefore the following code (Figure 5) is to be added.

```
  /*
   * Button Shapes.
   */
  static final int BUTTON_SHAPE_DEFAULT = 0;
  static final int BUTTON_SHAPE_ROUNDED = 1;
  static final int BUTTON_SHAPE_RECT = 2;
  static final int BUTTON_SHAPE_OVAL = 3;
```

Figure 5: Define `value` Strings in the Component class

1.2.2 Adding the New PropertyEditor to the Properties Panel

The YoungAndroidPalettePanel class creates the component palette on the left side of the Designer and instantiates the components' PropertyEditors. We need to add the logic to do this in the createPropertyEditor() method. First, we need to define a constant for the property in the propertyTypeContants class. Add the following code.

```
  /**
  * Button shapes.  * @see
com.google.appinventor.client.editor.youngandroid.properties.

  * YoungAndroidButtonShapeChoicePropertyEditor
  */
  public static final String PROPERTY_TYPE_BUTTON_SHAPE = "button_shape";
```

Then add the following case to the createPropertyEditor() method inside the same class.

```
    } else if
(editorType.equals(PropertyTypeConstants.PROPERTY_TYPE_BUTTON_SHAPE)) {
      return new YoungAndroidButtonShapeChoicePropertyEditor();
```

Figure 6: Addition to YoungAndroidPalettePanel.createPropertyEditor()

1.3. Associate the Property with the Component

Create a setter and a getter method for the new property in the component's class. Both the setter and the getter methods must be marked with the SimpleProperty annotation. The SimpleProperty annotation consists of a description, the property's category and whether or not the property is visible (visible in the BlocksEditor). The setter method must also be marked with the DesignerProperty annotation. This annotation consists of the property's editor type, which is defined in the DesignerProperty annotation, and the default value of the property.

For this example the following code needs to be added to the ButtonBase class.

This is the getter:

```
  /**
   * Returns the shape of the button.
   *
   * @return  one of {@link Component#BUTTON_SHAPE_DEFAULT},
   *          {@link Component#BUTTON_SHAPE_ROUNDED},
   *          {@link Component#BUTTON_SHAPE_RECT} or
```

```
     *          {@link Component#BUTTON_SHAPE_OVAL}
     */
    @SimpleProperty(
        category = PropertyCategory.APPEARANCE,
        userVisible = false)
    public int Shape() {
      return shape;
    }
```

false since user can not edit in Blocks Editor

the name of the method will be displayed in the Properties panel

Figure 7: Shape Getter

This is the setter:

```
    /**
     * Specifies the shape of the button. This does not check that the argument
     * is a legal value.
     *
     * @param shape one of {@link Component#BUTTON_SHAPE_DEFAULT},
     *          {@link Component#BUTTON_SHAPE_ROUNDED},
     *          {@link Component#BUTTON_SHAPE_RECT} or
     *          {@link Component#BUTTON_SHAPE_OVAL}
     *
     * @throws IllegalArgumentException if shape is not a legal value.
     */
    @DesignerProperty(editorType = DesignerProperty.PROPERTY_TYPE_BUTTON_SHAPE,
        defaultValue = Component.BUTTON_SHAPE_DEFAULT + "")
    @SimpleProperty(description = "Specifies the button's shape (default, " +
        "rounded, rectangular, oval). The shape will not be visible if an " +
        "Image is being displayed.", userVisible = false)
    public void Shape(int shape) {
      this.shape = shape;
      updateAppearance();
    }
```

the category only needs to be specified in the either the setter or the getter

Figure 8: Shape Setter

Then define the variable shape inside the same file with the following code.

```
    // Backing for button shape
    private int shape;
```

And add the following line to the ButtonBase constructor.

```
Shape(Component.BUTTON_SHAPE_DEFAULT);
```

After implementing the code in this section, the Shape property will appear in the Properties panel when a Button is selected in the designer view; however, changing the property does not yet affect the appearance of the Button.

2. Changing the Designer View When a Property Changes

The app's user interface can be viewed in two locations. The first location is in the Designer and the second is on the Android device (i.e. phone, tablet or emulator). The Designer is the view shown in the browser window and is what will be changed in this section.

Every Component has a corresponding mock class in the appengine project. The mock class is the visual representation of the component in the Designer, and the mock classes generally follow the same hierarchy as the Component classes. If your property changes visual aspects of the component (such as color), then the mock class of that component must adjust the Designer to show these visual changes.

2.1 Change Component's Attributes

There are already a number of methods written to change a component's attributes in MockComponent, MockComponentsUtil and MockVisibleComponent; most component mock classes inherit from at least one of these classes.

To change the appearance in the Designer, find the mock class that corresponds to the component to which the property was added. Determine if the class inherits a method that changes the appropriate attribute. If such a method exists, then simply write a method that calls that method with the appropriate arguments, as done in the setEnabledProperty() method in the MockButtonBase class. If a method doesn't exist then follow this Shape property example.

For this example the mock class associated with the ButtonBase component is MockButtonBase. Figure 9 shows the appearances for each value of the Shape property.

Shape	Image
default	
rounded	
rectangular	
oval	

Figure 9: Mock Buttons

Mock components are built on top of GWT widgets. The MockButtonBase creates a button widget which uses the system's defaults and in order to make the above shapes the button's corner radii needs to be changed. There does not already exist a method in MockButtonBase or any of its superclasses that could change a button's corner radii, so the following method (Figure 10) needs to be added to MockButtonBase.

```
// Legal values for shape are defined in
// com.google.appinventor.components.runtime.Component.java.
private int shape;

/*
 * Sets the button's Shape property to a new value.
 */
private void setShapeProperty(String text) {
  shape = Integer.parseInt(text);
  // Android Buttons with images take the shape of the image and do not
  // use one of the defined Shapes.
  if (hasImage) {
    return;
  }
  switch(shape) {
    case 0:
      // Default Button
      DOM.setStyleAttribute(buttonWidget.getElement(), "border-radius",
          "0px");
      break;
    case 1:
      // Rounded Button.
      // The corners of the Button are rounded by 10 px.
      // The value 10 px was chosen strictly for style.
      // 10 px is the same as ROUNDED_CORNERS_RADIUS defined in
      // com.google.appinventor.components.runtime.ButtonBase.java.
      DOM.setStyleAttribute(buttonWidget.getElement(), "border-radius",
          "10px");
      break;
    case 2:
      // Rectangular Button
      DOM.setStyleAttribute(buttonWidget.getElement(), "border-radius",
          "0px");
      break;
    case 3:
      // Oval Button
      String height = DOM.getStyleAttribute(buttonWidget.getElement(),
          "height");
      DOM.setStyleAttribute(buttonWidget.getElement(), "border-radius",
          height);
      break;
    default:
      // This should never happen
      throw new IllegalArgumentException("shape:" + shape);
  }
}
```

Figure 10: Addition to MockButtonBase Class

2.2 Update the onPropertyChange Method

Mock component classes contain an onPropertyChange() method which is called by GWT when any of the component's properties are changed and it determines how to change the view in the Designer. The onPropertyChange() method has two string

parameters, the `propertyName` and the `newValue`. The `propertyName` is a string representation of the name of the method defined in the property's getter (also the string displayed in the Properties Panel). The `newValue` is the value of the property and is passed to the method that was just created. For this example the `propertyName` would be "Shape" and the `newValue` could be "0", "1", "2" or "3".

There is a list of all the possible `propertyName` values in the MockVisualComponent class. The new property needs to be added to the list. For this example add the following line.

```
protected static final String PROPERTY_NAME_BUTTONSHAPE= "Shape";
```

Now back in the mock component's class add logic so that if the new property is changed it will call the method that was just created.

For this example add the following statement to the onPropertyChange() method of the MockButtonBase class

```
    } else if
(propertyName.equals(PROPERTY_NAME_BUTTONSHAPE)){
      setShapeProperty(newValue);
```

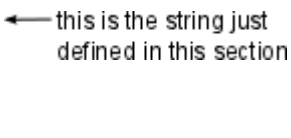


Figure 11: Additon to onPropertyChange()

2.3 Update Any Other Necessary Methods

If changing the new property doesn't affect any other properties then this section is complete. It might be a good idea to review the methods in the mock component class to confirm this. If the new property does affect other properties then update the methods called on those property changes as needed.

For this example, since the button shape only changes if there is no background image, the Shape property and the Image property affect each other. Therefore, since the setShapeProperty() method was created (or updated), the setImageProperty() method needs to be updated.

The following two gray lines need to be added to the setImageProperty() method.

```
/*
 * Sets the button's Image property to a new value.
 */
private void setImageProperty(String text) {
  imagePropValue = text;
  String url = convertImagePropertyValueToUrl(text);
  if (url == null) {
    hasImage = false;
    url = "";
    setBackgroundColorProperty(backgroundColor);
    setShapeProperty(Integer.toString(shape));
  } else {
    hasImage = true;
    // Android Buttons do not show a background color if they have an image.
```

```
      // The container's background color shows through any transparent
      // portions of the Image, an effect we can get in the browser by
      // setting the widget's background color to COLOR_NONE.
      MockComponentsUtil.setWidgetBackgroundColor(buttonWidget,
          "&H" + COLOR_NONE);
      DOM.setStyleAttribute(buttonWidget.getElement(), "border-radius", "0px");
    }
    MockComponentsUtil.setWidgetBackgroundImage(buttonWidget, url);
    image.setUrl(url);
  }
```

Figure 12: Addition to setImageProperty()

Now when the Shape property is changed the button shown in the designer view will also change to reflect the user's preference. There will still be no change to the button on the Android device.

3. Changing the Android Representation of the Component

Next decide how you would like to change the visual representation of the component on the Android device. Then implement the necessary code inside the class where you defined the property's getter and setters.

For the Shape property the component's BackgroundDrawable needs to be changed to change the shape. The table below describes how the ButtonBase component is changed for each Shape.

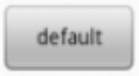
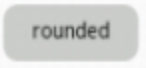
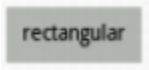
Shape	Image	Drawable
default		defaultButtonDrawable or no drawable and set the background color
rounded		RoundRectShape(CornerArray, null, null) where CornerArray is and Array of 8 floats each with the value 10f.
rectangular		RectShape() drawable
oval		OvalShape() drawable

Figure 13: Shape Drawables

The process of changing the component on the Android device is very specific to both the component being changed and the property being added. Review the methods currently available to the component and components with similar properties to determine what code needs to be added or changed.

The next section details the process followed when implementing the Shape property.

3.1. Button Shape Example

In the ButtonBase class add the following constants.

```
  // Constant for shape
  // 10px is the radius of the rounded corners.
  // 10px was chosen for esthetic reasons.
  private static final float ROUNDED_CORNERS_RADIUS = 10f;
  private static final float[] ROUNDED_CORNERS_ARRAY = new float[] {
      ROUNDED_CORNERS_RADIUS,ROUNDED_CORNERS_RADIUS, ROUNDED_CORNERS_RADIUS,
      ROUNDED_CORNERS_RADIUS,ROUNDED_CORNERS_RADIUS, ROUNDED_CORNERS_RADIUS,
      ROUNDED_CORNERS_RADIUS,ROUNDED_CORNERS_RADIUS };

  // Constant background color for buttons with a Shape other than default
  private static final int SHAPED_DEFAULT_BACKGROUND_COLOR = Color.LTGRAY;
```

Replace the updateAppearance() method with the following.

```
  // Update appearance based on values of backgroundImageDrawable,
backgroundColor and
  // shape.
  // Images take precedence over background colors.
  private void updateAppearance() {
    // If there is no background image,
    // the appearance depends solely on the background color and shape.
    if (backgroundImageDrawable == null) {
      if (shape == Component.BUTTON_SHAPE_DEFAULT) {
        if (backgroundColor == Component.COLOR_DEFAULT) {
        // If there is no background image and color is default,
        // restore original 3D bevel appearance.
        ViewUtil.setBackgroundDrawable(view, defaultButtonDrawable);
        } else {
          // Clear the background image.
          ViewUtil.setBackgroundDrawable(view, null);
          // Set to the specified color (possibly COLOR_NONE for transparent).
          TextViewUtil.setBackgroundColor(view, backgroundColor);
        }
      } else {
        // If there is no background image and the shape is something other
        // than default, create a drawable with the appropriate shape and
       // color.
        setShape();
      }
    } else {
      // If there is a background image
      ViewUtil.setBackgroundImage(view, backgroundImageDrawable);
    }
  }
```

Figure 14: Addition to updateAppearance()

Add the setShape() method with the following.

```
  // Throw IllegalArgumentException if shape has illegal value.
  private void setShape() {
    ShapeDrawable drawable = new ShapeDrawable();
    // Set color of drawable.
    drawable.getPaint().setColor((backgroundColor == Component.COLOR_DEFAULT)
                                 ? shapedDefaultBackgroundColor :
                                     backgroundColor);
    // Set shape of drawable.
    switch (shape) {
      case Component.BUTTON_SHAPE_ROUNDED:
        drawable.setShape(new RoundRectShape(ROUNDED_CORNERS_ARRAY, null,
            null));
        break;
      case Component.BUTTON_SHAPE_RECT:
        drawable.setShape(new RectShape());
```

```
        break;
      case Component.BUTTON_SHAPE_OVAL:
        drawable.setShape(new OvalShape());
        break;
      default:
        throw new IllegalArgumentException();
    }
    // Set drawable to the background of the button.
    view.setBackgroundDrawable(drawable);
    view.invalidate();
  }
```

Figure 15: Addition to setShape()

Now when the Shape of a Button is changed both the Mock button and the Button on the Android device should change.

4. Update Version Numbers

4.1 YaVersion

The YaVersion class defines the Young Android System version number, Blocks Language version number and Component version numbers. If the Blocks Language or any of the Components were updated in the previous sections then their version numbers and the Young Android System version number needs to be increased. There are also instructions, in the class, describing updating each of these values.

For the Button Shape example the ButtonBase component was updated, therefore the ButtonBase component version number needs to be increased along with the version numbers of components that are subclassed of ButtonBase and the Young Android System version number. Add the following gray code to the YaVersion class.

```
// ........................... Young Android System Version Number
// ...........................

  // YOUNG_ANDROID_VERSION must be incremented when either the blocks language
  // or a component changes.
  // TODO(lizlooney) - should this version number be generated so that it is
  // automatically incremented when the blocks language or a component changes?

  // For YOUNG_ANDROID_VERSION 2:
  // - The Logger component was removed. The Notifier component should be used
  // instead.
  // - TINYWEBDB_COMPONENT_VERSION was incremented to 2.
       :
       :
       :
  // For YOUNG_ANDROID_VERSION 54:
  // - BUTTON_COMPONENT_VERSION was incremented to 4.
  // - CONTACTPICKER_COMPONENT_VERSION was incremented to 4.
  // - IMAGEPICKER_COMPONENT_VERSION was incremented to 4.
  // - LISTPICKER_COMPONENT_VERSION was incremented to 5.
  // - PHONENUMBERPICKER_COMPONENT_VERSION was incremented to 4.
  public static final int YOUNG_ANDROID_VERSION = 54;
```

```
  // For BUTTON_COMPONENT_VERSION 2:
  // - The Alignment property was renamed to TextAlignment.
  // For BUTTON_COMPONENT_VERSION 3:
  // - The LongClick event was added.
  // For BUTTON_COMPONENT_VERSION 4:
  // - The Shape property was added.
  public static final int BUTTON_COMPONENT_VERSION = 4;

  // For CONTACTPICKER_COMPONENT_VERSION 2:
  // - The Alignment property was renamed to TextAlignment.
  // For CONTACTPICKER_COMPONENT_VERSION 3:
  // - The method Open was added.
  // For CONTACTPICKER_COMPONENT_VERSION 4:
  // - The Shape property was added.
  public static final int CONTACTPICKER_COMPONENT_VERSION = 4;

  // For IMAGEPICKER_COMPONENT_VERSION 2:
  // - The Alignment property was renamed to TextAlignment.
  // For IMAGEPICKER_COMPONENT_VERSION 3:
  // - The method Open was added.
  // For IMAGEPICKER_COMPONENT_VERSION 4:
  // - The Shape property was added.
  public static final int IMAGEPICKER_COMPONENT_VERSION = 4;

  // For LISTPICKER_COMPONENT_VERSION 2:
  // - The Alignment property was renamed to TextAlignment.
  // For LISTPICKER_COMPONENT_VERSION 3:
  // - The SelectionIndex read-write property was added.
  // For LISTPICKER_COMPONENT_VERSION 4:
  // - The method Open was added.
  // For LISTPICKER_COMPONENT_VERSION 5:
  // - The Shape property was added.
  public static final int LISTPICKER_COMPONENT_VERSION = 5;

  // For PHONENUMBERPICKER_COMPONENT_VERSION 2:
  // - The Alignment property was renamed to TextAlignment.
  // For PHONENUMBERPICKER_COMPONENT_VERSION 3:
  // - The method Open was added.
  // For PHONENUMBERPICKER_COMPONENT_VERSION 4:
  // - The Shape property was added.
  public static final int PHONENUMBERPICKER_COMPONENT_VERSION = 4;
```

Figure 16: Addition to YaVersion Class

4.2 YoungAndroidFormUpgrader

As stated in the instructions in YaVersion, if a component version is updated code must be added to the YoungAndroidFormUpgrader. For the Button Shape example the following gray code needs to be added to the YoungAndroidFormUpgrader class.

```
private static int upgradeButtonProperties(Map<String, JSONValue>
componentProperties,
    int srcCompVersion) {
  if (srcCompVersion < 2) {
    // The Alignment property was renamed to TextAlignment.
    handlePropertyRename(componentProperties, "Alignment", "TextAlignment");
    // Properties related to this component have now been upgraded to version
    // 2.
    srcCompVersion = 2;
  }
  if (srcCompVersion < 3) {
```

```
    // The LongClick event was added.
    // No properties need to be modified to upgrade to version 3.
    srcCompVersion = 3;
  }
  if (srcCompVersion < 4) {
    // The Shape property was added.
    // No properties need to be modified to upgrade to version 4.
    srcCompVersion = 4;
  }
  return srcCompVersion;
}

private static int upgradeContactPickerProperties(Map<String, JSONValue>
    componentProperties, int srcCompVersion) {
  if (srcCompVersion < 2) {
    // The Alignment property was renamed to TextAlignment.
    handlePropertyRename(componentProperties, "Alignment", "TextAlignment");
    // Properties related to this component have now been upgraded to version
    // 2.
    srcCompVersion = 2;
  }
  if (srcCompVersion < 3) {
    // The Open method was added.  No changes are needed.
    srcCompVersion = 3;
  }
  if (srcCompVersion < 4) {
    // The Shape property was added.
    // No properties need to be modified to upgrade to version 4.
    srcCompVersion = 4;
  }
  return srcCompVersion;
}

private static int upgradeImagePickerProperties(Map<String, JSONValue>
    componentProperties, int srcCompVersion) {
  if (srcCompVersion < 2) {
    // The Alignment property was renamed to TextAlignment.
    handlePropertyRename(componentProperties, "Alignment", "TextAlignment");
    // Properties related to this component have now been upgraded to version
    // 2.
    srcCompVersion = 2;
  }
  if (srcCompVersion < 3) {
    // The Open method was added.  No changes are needed.
    srcCompVersion = 3;
  }
  if (srcCompVersion < 4) {
    // The Shape property was added.
    // No properties need to be modified to upgrade to version 4.
    srcCompVersion = 4;
  }
  return srcCompVersion;
}

private static int upgradeListPickerProperties(Map<String, JSONValue>
```

```
    componentProperties, int srcCompVersion) {
  if (srcCompVersion < 2) {
    // The Alignment property was renamed to TextAlignment.
    handlePropertyRename(componentProperties, "Alignment", "TextAlignment");
    // Properties related to this component have now been upgraded to version
    // 2.
    srcCompVersion = 2;
  }
  if (srcCompVersion < 3) {
    // The SelectionIndex property was added.  No changes are needed.
    srcCompVersion = 3;
  }
  if (srcCompVersion < 4) {
    // The Open method was added.  No changes are needed.
    srcCompVersion = 4;
  }
  if (srcCompVersion < 5) {
    // The Shape property was added.
    // No properties need to be modified to upgrade to version 5.
    srcCompVersion = 5;
  }
  return srcCompVersion;
}

private static int upgradePhoneNumberPickerProperties(Map<String, JSONValue>
    componentProperties, int srcCompVersion) {
  if (srcCompVersion < 2) {
    // The Alignment property was renamed to TextAlignment.
    handlePropertyRename(componentProperties, "Alignment", "TextAlignment");
    // Properties related to this component have now been upgraded to version
2.
    srcCompVersion = 2;
  }
  if (srcCompVersion < 3) {
    // The Open method was added.  No changes are needed.
    srcCompVersion = 3;
  }f
  if (srcCompVersion < 4) {
    // The Shape property was added.
    // No properties need to be modified to upgrade to version 4.
    srcCompVersion = 4;
  }
  return srcCompVersion;
}
```

Figure 17: Addition to YoungAndroidFormUpgrader Class

4.3 BlockSaveFile

The BlockSaveFile class must also be updated. If any component version numbers were increased then add code to upgradeComponentBlocks().

Add the following gray code to upgradeComponentBlocks() for the Button Shape example.

```
private int upgradeButtonBlocks(int blkCompVersion, String componentName) {
  if (blkCompVersion < 2) {
    // The Alignment property was renamed to TextAlignment.
    handlePropertyRename(componentName, "Alignment", "TextAlignment");
    // Blocks related to this component have now been upgraded to version 2.
    blkCompVersion = 2;
  }
  if (blkCompVersion < 3) {
    // The LongClick event was added.
    // No blocks need to be modified to upgrade to version 3.
    blkCompVersion = 3;
  }
  if (blkCompVersion < 4) {
    // The Shape property was added.
    // No blocks need to be modified to upgrade to version 4.
    blkCompVersion = 4;
  }
  return blkCompVersion;
}

private int upgradeContactPickerBlocks(int blkCompVersion, String
componentName) {
  if (blkCompVersion < 2) {
    // The Alignment property was renamed to TextAlignment.
    handlePropertyRename(componentName, "Alignment", "TextAlignment");
    // Blocks related to this component have now been upgraded to version 2.
    blkCompVersion = 2;
  }
  if (blkCompVersion < 3) {
    // The Open method was added, which does not require changes.
    blkCompVersion = 3;
  }
  if (blkCompVersion < 4) {
    // The Shape property was added.
    // No blocks need to be modified to upgrade to version 4.
    blkCompVersion = 4;
  }
  return blkCompVersion;
}

private int upgradeImagePickerBlocks(int blkCompVersion, String componentName)
{
  if (blkCompVersion < 2) {
    // The Alignment property was renamed to TextAlignment.
    handlePropertyRename(componentName, "Alignment", "TextAlignment");
    // Blocks related to this component have now been upgraded to version 2.
    blkCompVersion = 2;
  }
  if (blkCompVersion < 3) {
    // The Open method was added, which does not require changes.
    blkCompVersion = 3;
  }
  if (blkCompVersion < 4) {
    // The Shape property was added.
    // No blocks need to be modified to upgrade to version 4.
```

```
    blkCompVersion = 4;
  }
  return blkCompVersion;
}

private int upgradeListPickerBlocks(int blkCompVersion, String componentName) {
  if (blkCompVersion < 2) {
    // The Alignment property was renamed to TextAlignment.
    handlePropertyRename(componentName, "Alignment", "TextAlignment");
    // Blocks related to this component have now been upgraded to version 2.
    blkCompVersion = 2;
  }
  if (blkCompVersion < 3) {
    // The SelectionIndex property was added, which does not require changes.
    blkCompVersion = 3;
  }
  if (blkCompVersion < 4) {
    // The Open method was added, which does not require changes.
    blkCompVersion = 4;
  }
  if (blkCompVersion < 5) {
    // The Shape property was added.
    // No blocks need to be modified to upgrade to version 5.
    blkCompVersion = 5;
  }
  return blkCompVersion;
}

private int upgradePhoneNumberPickerBlocks(int blkCompVersion, String
componentName) {
  if (blkCompVersion < 2) {
    // The Alignment property was renamed to TextAlignment.
    handlePropertyRename(componentName, "Alignment", "TextAlignment");
    // Blocks related to this component have now been upgraded to version 2.
    blkCompVersion = 2;
  }
  if (blkCompVersion < 3) {
    // The Open method was added, which does not require changes.
    blkCompVersion = 3;
  }
  if (blkCompVersion < 4) {
    // The Shape property was added.
    // No blocks need to be modified to upgrade to version 4.
    blkCompVersion = 4;
  }
  return blkCompVersion;
}
```

Figure 18: Addition to BlockSaveFile Class

APPENDIX E – BUG REPORT

I am getting a Build failed! Unexpected problems generating YAIL. error when I attempt to package a project that I have previously downloaded, unzipped, rezipped using a Mac, and then uploaded. This error does not occur if I don't unzip the downloaded project file before re-uploading it. The new zip file is significantly bigger than the original one (5,534 bytes vs. 855 bytes) but the uncompressed contents for the files appear to be the same.

Below are the detailed steps I followed to get this error. I have also attached the referenced files. Please let me know if you have any suggestions about why this is happening.

Thanks, Kate

OneScreenOrig - Success
- -Created very simple project with only one screen and no assets and no blocks. OneScreenOrig
- -Blocks Editor Opens correctly
- -Able to connect and run project on the phone
- -Application successfully downloads to phone
- -Application successfully downloads to computer. OneScreenOrig.apk
- -Source is successfully downloaded to computer. OneScreenOrig.zip

OneScreenOrig (uploaded, not unzipped) - Success
- -Delete OneScreenOrig project from MyProjects
- -Successfully Uploaded the source to MyProjects. Did not open the zip file.
- -Blocks Editor Opens correctly
- -Able to connect and run project on the phone.
- -Application successfully downloads to phone
- -Application successfully downloads to computer. OneScreenOrig(1).apk

OneScreenRezipped (uploaded, not unzipped) - Success
- -Copied OneScreenOrig.zip and renamed to copy to OneScreenRezipped.zip. Still have not opened the zip file
- -Successfully Uploaded OneScreenRezipped.zip to MyProjects.
- -Using OneScreenRezipped project Blocks Editor Opens correctly
- -Able to connect and run project on the phone
- -Application successfully downloads to phone
- -Application successfully downloads to computer. OneScreenRezipped.apk

OneScreenRezipped (uploaded, unzipped and rezipped) - FAIL
- -Delete OneScreenRezipped project from MyProjects
- -Renamed OneScreenReziped.zip to OneScreenRezipped1.zip
- -Opened the unziped folder, OneScreenRezipped.
- -In the folder was a src folder and a youngandroidproject folder.
- -Selected both folders, right clicked and selected Compress 2 Items.

-A new zip file, Archive.zip, is created inside the OneScreenRezipped folder.
-Moved Archive.zip outside the OneScreenRezipped folder and renamed it OneScreenRezipped.zip
-Successfully Uploaded OneScreenRezipped.zip.to MyProjects.
-Blocks Editor Opens correctly
-Able to connect and run project on the phone
-When attempting to download to the connected phone the following error is received after about 15 seconds.

Build failed! Unexpected problems generating YAIL.

-The debugging message is:

```
Build of OneScreenRezipped requested at 2012 Mar 26 14:04:21.
Waiting for 10 seconds.

Unexpected problems generating YAIL.
```

-When attempting to download to this computer the following error is received after about 15 seconds.

Build failed! Unexpected problems generating YAIL.

APPENDIX F – MERGE CODE

```
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipFile;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


public class Merge {
  static BufferedReader br = new BufferedReader(
      new InputStreamReader(System.in));
  public static final void main(String[] args) {

    try {
      System.out.print("First Project: ");
      String firstProject=readLine();

      System.out.print("Second Project: ");
      String secondProject=readLine();

      System.out.print("Name of Merged Project: ");
      String mergedProject=readLine();

      String outFileName = mergedProject;

      ZipFile inZipFile1 = new ZipFile(firstProject);
      ZipFile inZipFile2 = new ZipFile(secondProject);
      ZipOutputStream outZip = new ZipOutputStream(
          new FileOutputStream(outFileName));
      ZipInputStream inZip1 = new ZipInputStream(new BufferedInputStream(
          new FileInputStream(firstProject)));
      ZipInputStream inZip2 = new ZipInputStream(new BufferedInputStream(
          new FileInputStream(secondProject)));


      List<String> filenamesFromZip1 = new ArrayList<String>();
      List<String> filenamesFromZip2 = new ArrayList<String>();
      byte [] buf = new byte[1024];

      printZipContent(inZipFile1);

      Boolean more = true;
      while(more){
        System.out.print("Enter full name of the file to copy from the " +
                    "first Project");
        String file = readLine();
        filenamesFromZip1.add(file);
        System.out.print("Would you like to add another file (Y/N)");
        String input = readLine();
        if(input.equalsIgnoreCase("n")){
          more=false;
        }else if (input.equalsIgnoreCase("y")){
          more=true;
        }else{
```

```
        System.out.println(input);
      }
    }

    System.out.println("Now Project Two");
    printZipContent(inZipFile2);

    more=true;
    while(more){
      System.out.print("Enter full name of the file to copy from the " +
                  "second Project");
      String file = readLine();
      filenamesFromZip2.add(file);
      System.out.print("Would you like to add another file (Y/N)");
      String input = readLine();
      if(input.equalsIgnoreCase("n")){
        more=false;
      }else if (input.equalsIgnoreCase("y")){
        more=true;
      }else{
        System.out.println(input);
      }
    }

    ZipEntry curEntry;
    while((curEntry = inZip1.getNextEntry())!=null){
      if (filenamesFromZip1.contains(curEntry.getName())){
        //System.out.println(curEntry.getName());
        outZip.putNextEntry(curEntry);
        int len;
        while((len = inZip1.read(buf))>0){
          outZip.write(buf, 0, len);
        }
        outZip.closeEntry();
        inZip1.closeEntry();
      }
    }
    inZip1.close();

    while((curEntry = inZip2.getNextEntry())!=null){
      if (filenamesFromZip2.contains(curEntry.getName())){
        //System.out.println(curEntry.getName());
        outZip.putNextEntry(curEntry);
        int len;
        while((len = inZip2.read(buf))>0){
          outZip.write(buf, 0, len);
        }
        outZip.closeEntry();
        inZip2.closeEntry();
      }
    }
    inZip2.close();

    outZip.close();

  } catch (IOException ioe) {
    System.err.println("Unhandled exception:");
    ioe.printStackTrace();
    return;
  }
}
public static void printZipContent(ZipFile zFile){
  System.out.println("List of Files in "+ zFile.getName() +" :");
  Enumeration <? extends ZipEntry> e = zFile.entries();
  while(e.hasMoreElements()){
```

```
      System.out.println(e.nextElement().getName());
    }
  }

  public static String readLine(){
    try {
      String str = br.readLine();
      return str;
    } catch (IOException ioe) {
      System.out.println("IO error trying to read the project name!");
      System.exit(1);
      return null;
    }
  }

}
```

APPENDIX G – AIMERGER DOCUMENTATION

HOW TO USE APP INVENTOR'S AIMERGER FOR DEVELOPMENT AS A TEAM

The AIMerger tool can be very useful when developing an app within a team. The tool allows for multiple developers to work on different screens of the app and then merge them together. This document outlines the process of using the AIMerger to develop an app in a team environment. It will use the example of a simple two-screen app developed by two different developers to demonstrate this process.

Overview

Each developer will work on their own separate project file. These project files can be either under the same username or different usernames. There can only be one "Screen1" per project and "Screen1" can not be renamed; therefore only the developer designing the first screen to appear on the app should populate "Screen1." All other developers should leave "Screen1" blank and only develop additional screens.

One developer can write code to call a screen they are not developing but they must know the name that has been assigned to that screen by its developer. Also, two different developers/screens can use the same database or asset but they must be named the same (for more details on this see the Universal Databases and Assets

section below). Finally, no two different screens or assets can have the same name. For these reasons it is important to decide beforehand the name of each screen and a naming convention for assets that will insure no unwanted duplicates.

Once separate projects are complete they can be merged together using the AIMerger.

Dividing Work

Work should be divided by screens. Each screen should be assigned to a developer and there should be only one version of each screen to be merged into the final app. This document will follow an example for which there are two screens and two developers working on an app named “CountDown.” The CountDown app will work like a timer. The first screen shows the time counting down and allows the user to start, stop or reset the timer. From the first screen the user can also switch to a second screen to set the amount of time to count down.

Developer1 will work on the first screen and call it “Screen1”. Developer2 will work on the second screen and call it “SetTime”. The naming convention for assets will be the screen name followed by the asset name. There are no universal assets or databases. Screen1 will open the SetTime screen and when the SetTime screen is closed it will pass a number, representing the total number of seconds to countdown, to Screen1.

Developer 1 Work In App Inventor

Developer1 will log into App Inventor using their account and create a new project called “CountDownScreen1.”

Design view

The Design view for CountDownScreen1’s Screen1 is shown below in Figure 1. Note that the project name, the screen name and assets follow the predetermined naming convention.

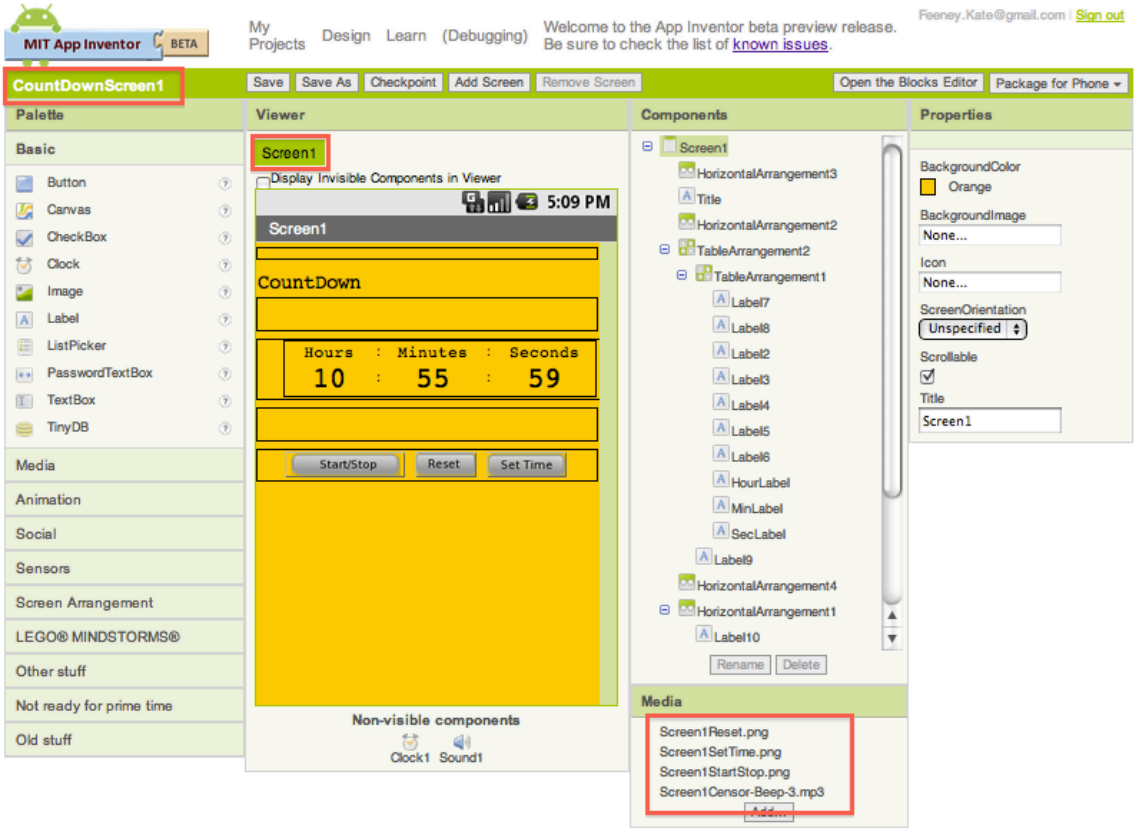


Figure 1: CountDownScreen1's Screen1 Design view

Blocks Editor

Developer1 then creates the blocks needed for Screen1. These blocks included a block to open the SetTime screen when the "Set Time" button is clicked and a block to handle when the SetTime screen is closed. Views of the SetTime.Click block and the Screen1.OtherScreenClosed block are shown in Figure 2 and 3 respectively. The complete set of blocks for Screen1 are shown in Appendix A.

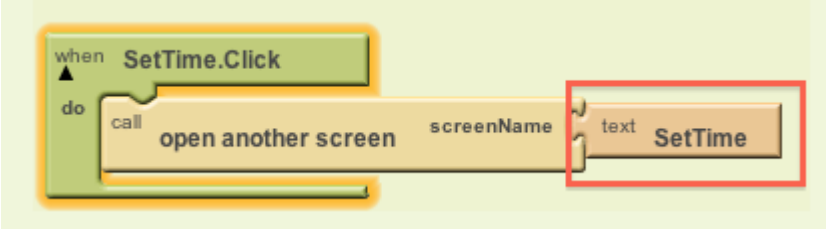


Figure 2: CountDownScreen1's Screen1 SetTime.Click block

In the SetTime.Click block, Screen1 opens the SetTime screen. Note that the string assigned to screenName must be exactly what was decided in advance as the name of the second screen.

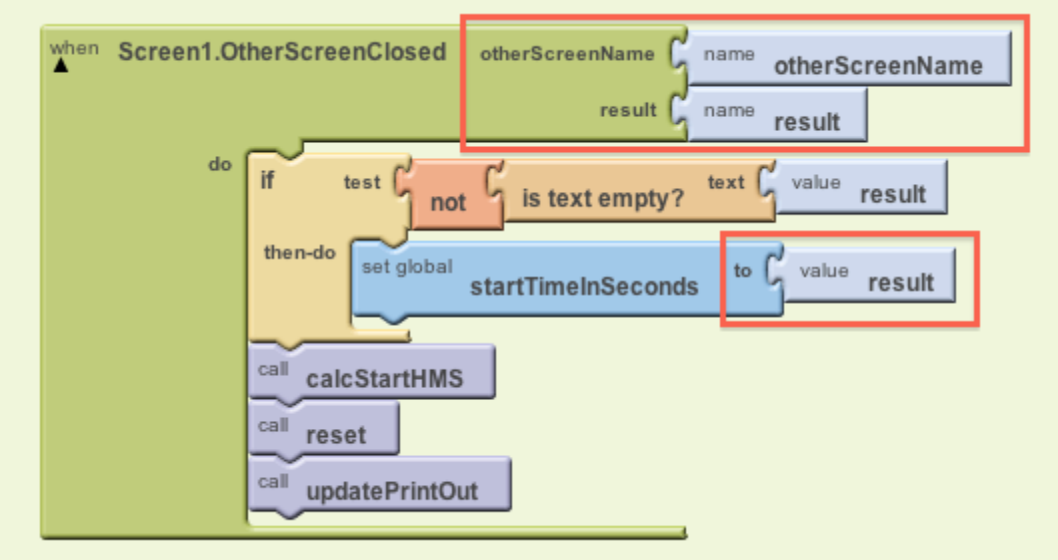


Figure 3: CountDownScreen1's Screen1 OtherScreenClosed block

The Screen1.OtherScreenClosed receives the number of seconds to countdown from the SetTime screen when the screen is closed and sets the startTimeInSeconds variable to it.

Download Source Code

Once Developer1 completes Screen1 they download the source code. This is done by going to the My Projects view and checking the checkbox next to the CountDownScreen1 project. Then clicking on the "More Actions" dropdown and selecting "Download Source". In Figure 4 red arrows show how to select a project and where to click to download the project's source.

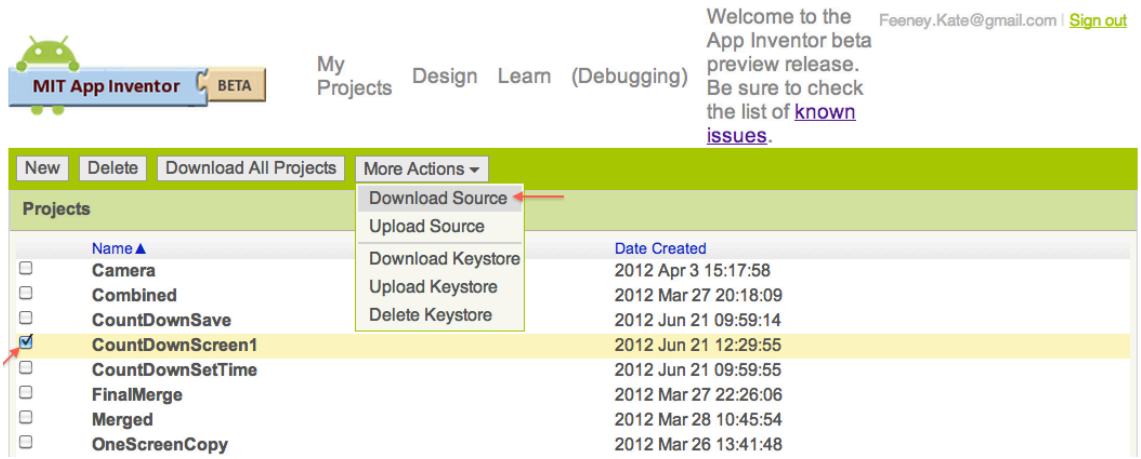


Figure 4: Steps to download CountDownScreen1 source

Developer 2 Work In App Inventor

Developer2 will log into App Inventor using their account and create a new project called "CountDownSetTime."

Design view

The Design view for CountDownSetTime's Screen1 is shown below in Figure 5. The Screen1 is empty except for a button that takes you to the SetTime screen. This button is only for Developer2 to get to the SetTime screen during testing and debuging. This Screen1 will not be merged into the final app.

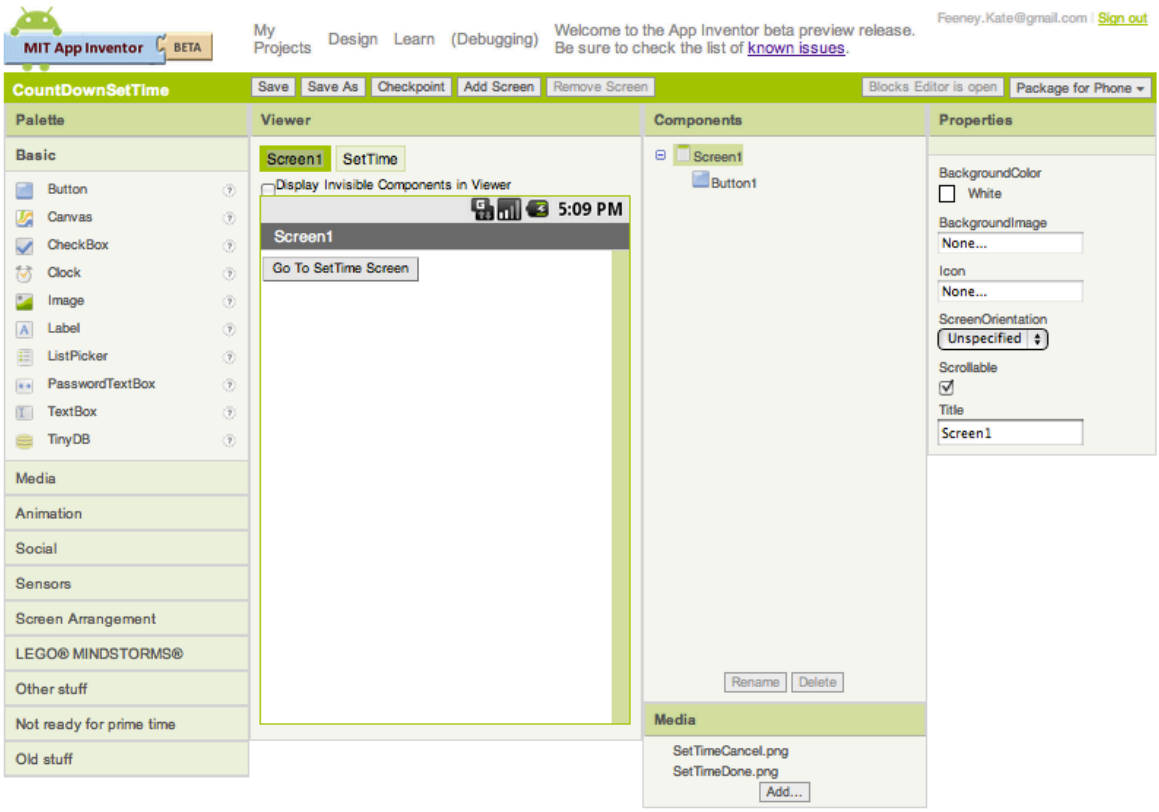


Figure 5: CountDownSetTime's Screen1 Design view

The Design view for CountDownSetTime's SetTime screen is shown below in Figure 6. Note that the project name, the screen name and assets follow the predetermined naming convention.

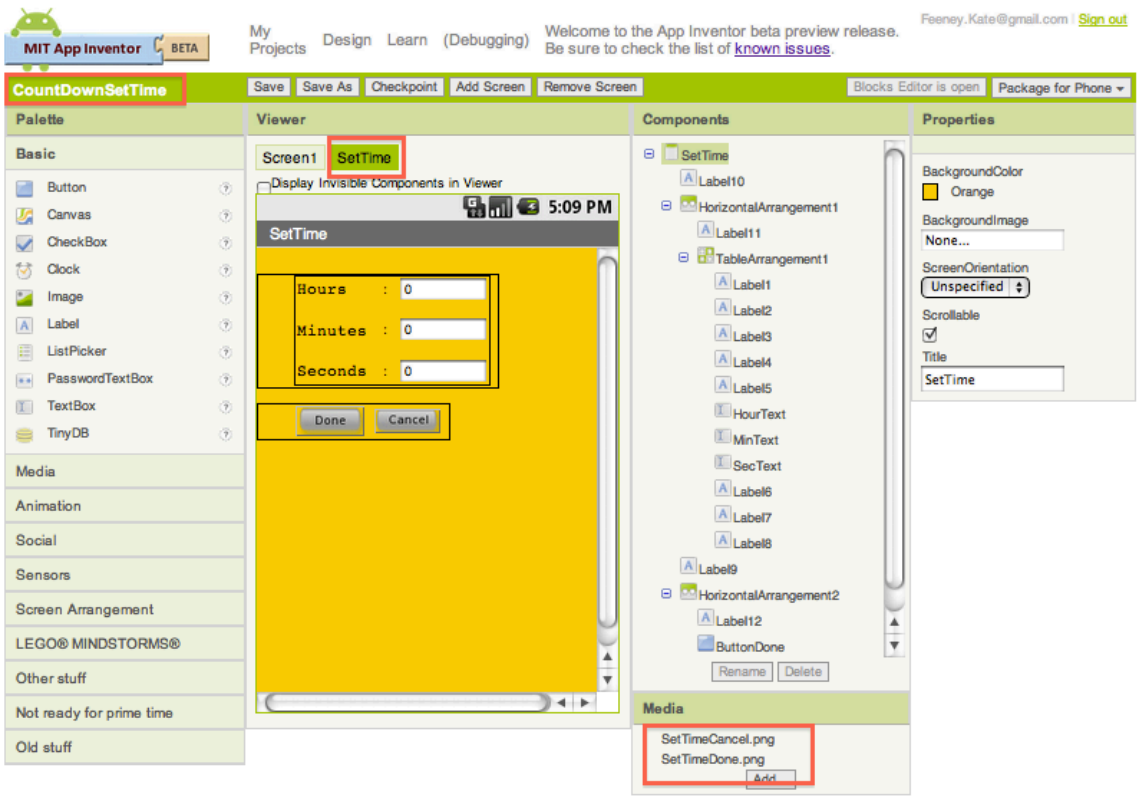


Figure 6: CountDownSetTime's SetTime screen Design view

Blocks Editor

Developer2 then creates the blocks needed for the SetTime screen. These blocks include a block to send the number of seconds to countdown to Screen1 when the SetTime screen is closed. This block is the ButtonDone.Click block shown in Figure 7. A complete set of blocks for the SetTime screen are shown in Appendix B.

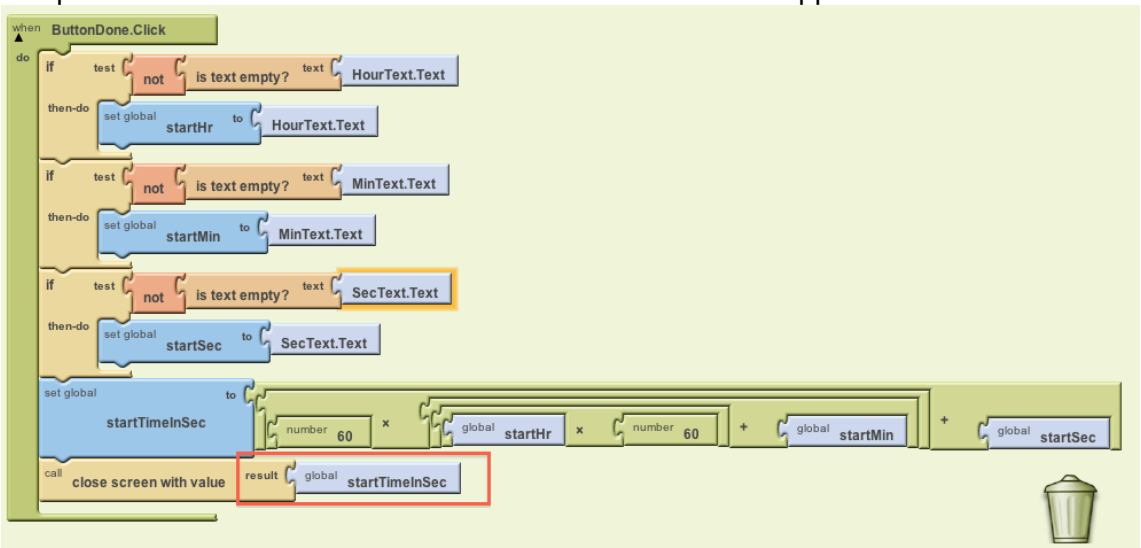


Figure 7: ButtonDone.Click block for CountDownSetTime's SetTime screen

Download Source Code

Developer2 downloads the source for CountDownSetTime following the same steps Developer1 followed to download the source for CountDownScreen1.

Merging into one Project

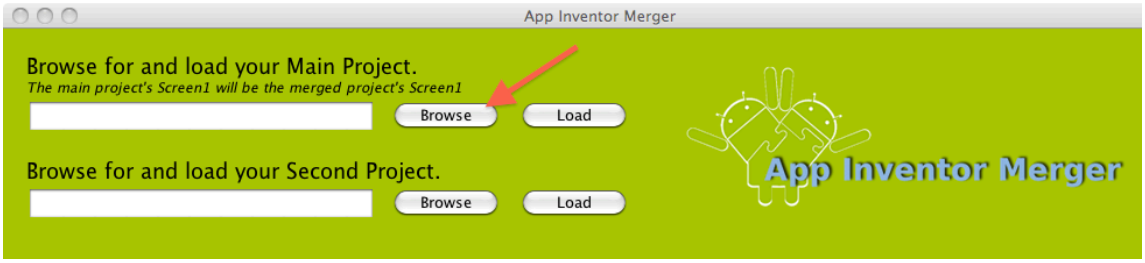
Once both developers have downloaded the source code for their respective projects, the two projects can be merged into the final app using the following steps.

Launch the AIMerger.

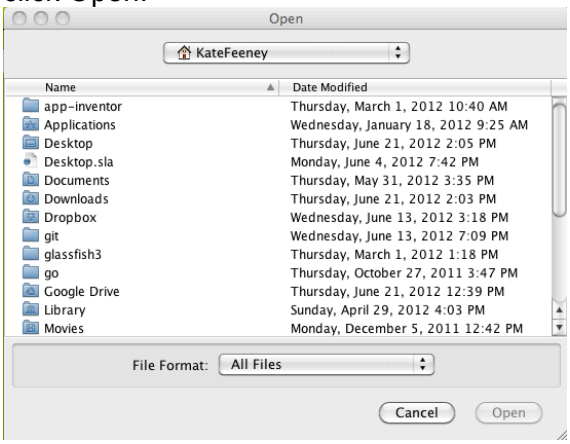
The main project will be CountDownScreen1 since Screen1 from this project will be the Screen1 for the final app and the second project will be CountDownSetTime.

Find and Load Both Projects

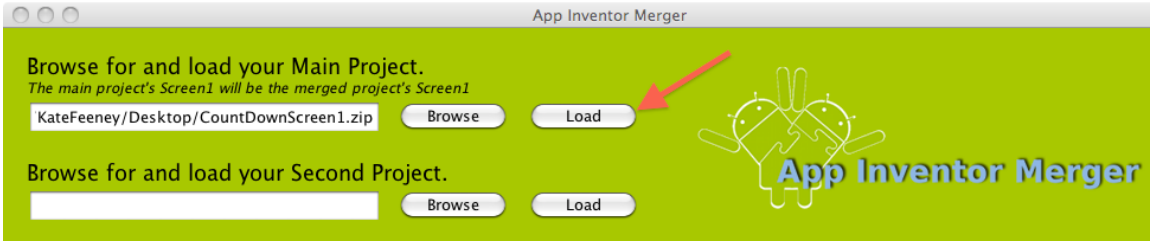
Select the browse button for the main project.



A file browser window will appear. Find and select the CountDownScreen1.zip file, then click Open.

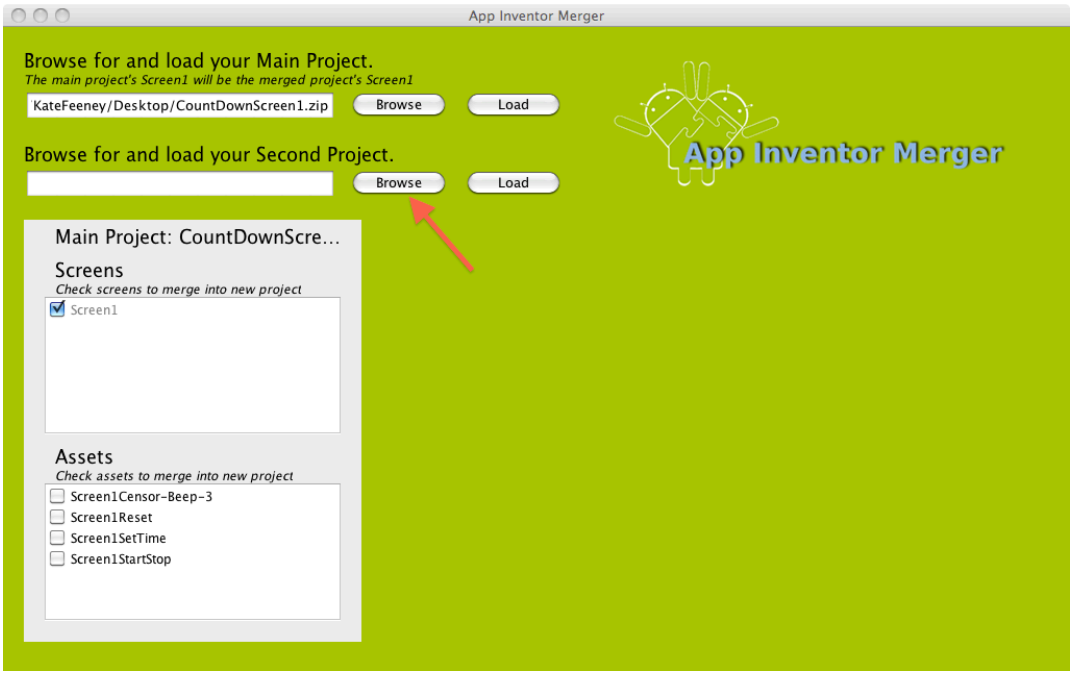


The path for the project file will appear in the main project text box. Click the main project's Load button to load the project into the AIMerger.

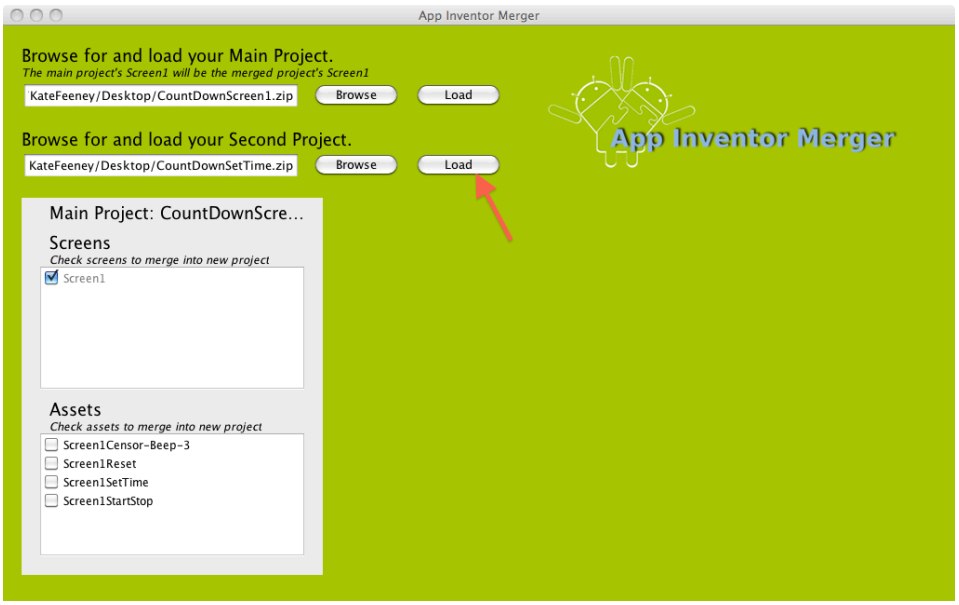


A list of the main project's assets and screens will appear in the lower left hand corner of the screen and this means that the main project has been loaded into the AIMerger.

Click the browse button for the second project.



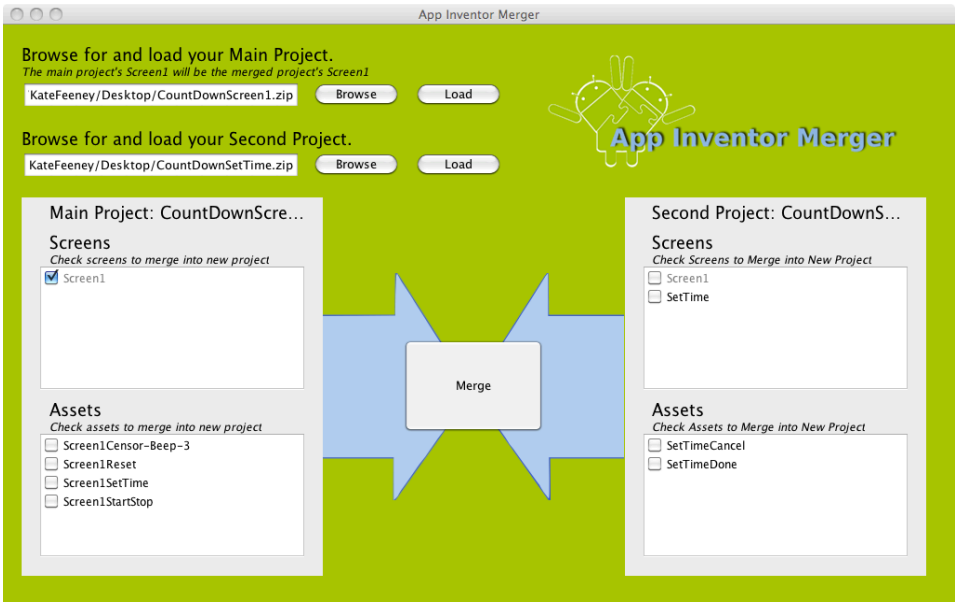
A file browser window will appear. Find and select the CountDownSetTime.zip file, then click Open. The path for the project file will appear in the second project text box. Click the second project's Load button to load the project into the AIMerger.



A list of the second project's assets and screens will appear in the lower right hand corner of the screen and this means that the second project has been loaded into the AIMerger.

Merge the Projects

Once two projects have been loaded a Merge button will appear between them.

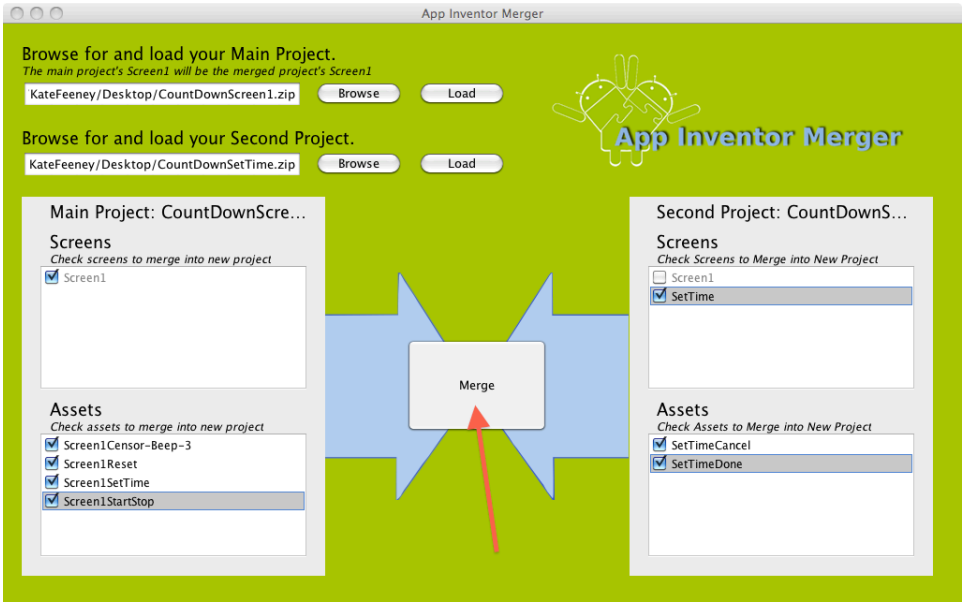


Check the boxes next to all of the screens and assets you wish to merge into the final app.

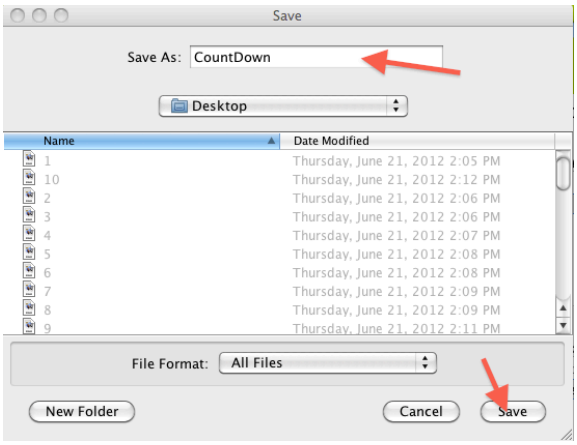
In this case only one screen is being selected from each project but in other cases multiple screens can be merged from the same project file. For this example all assets

and screens are selected except Screen1 from the second project. Screen1 from the second project is grayed out since Screen1 from the main project is required and two screens with the same name can be merged.

Click Merge.



A save dialog window will appear. Browse to where you would like to save the project, enter the project's name and then click Save. The project will be saved as a zip file. In this example the project's name will be CountDown and the file CountDown.zip will be saved on the desktop since that is the directory selected (although this file can be saved in any directory).



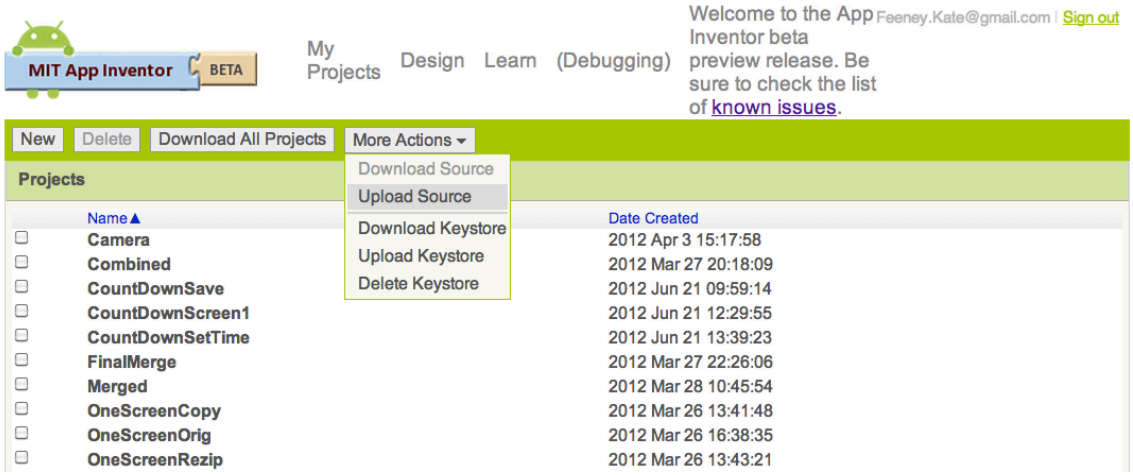
A dialog box will appear letting you know that your projects have been successfully merged. There will now be a zip file on the desktop named CountDown.zip. The dialog box will also ask if you would like to merge another project. Since this example only has two developers working on two different project files, there is no need to merge another project. Click No and confirm that you want to close the merger. For more information

about merging more than two project see the Merging More Than Two Projects section below.

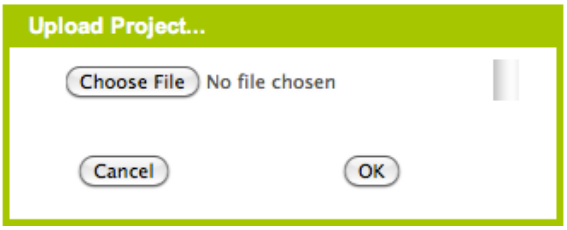
Upload Final Project to App Inventor

The zip file created by the AIMerger is your new project file. This project file can be uploaded to App Inventor so that you have a complete project in App Inventor.

To upload the file, launch App Inventor and go to the My Projects page. Click on the More Actions dropdown and select Upload Source.



The following dialog box will appear. Select Choose File. A file browsing window will appear. Find and select the zip file that was just created by the AIMerger. For this example select the CountDown.zip file located on the desktop and then click Open followed by OK.



Your new project now appears in your list of projects and opens in the App Inventor window.

Extras

Universal Databases and Assets

Assets

If your app uses the same asset on multiple screens, then you can make that asset universal. This is done by giving it a name that does not follow the normal naming convention but instead is repeated by all screens. When the projects are merged the universal asset will only need to be selected from one of the projects.

For example, imagine you have a logo, which is an image, that should appear at the top of every screen. Instead of having every developer name the logo something different (Screen1Logo, Screen2Logo …), as the naming convention would require, every developer can simply name it Logo. When the projects are merged only check the Logo asset listed under the main project so that the asset is only loaded into the final app once but all the screens will be able to access it.

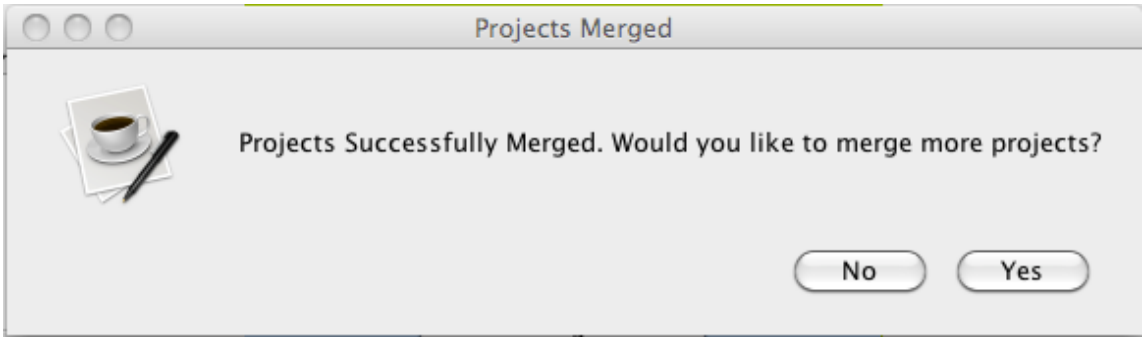
Databases

Apps that use databases can be merged using the AIMerger very easily. Different developers can even work on different screens, that use the same database, separately and then merge them together at the end.

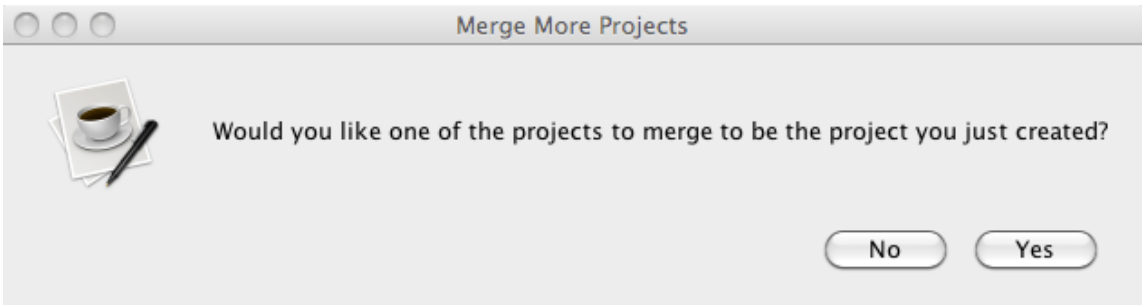
The only requirement for merging screens that share a database is that the name assigned to the database for each screen is the same. Once the projects are merged the same database will work for all the screens.

Merging More Than Two Projects

The AIMerger is still a very useful tool even if there are more than two developers working on more than two project files. The process is exactly the same as the two developer process, described above, until the dialog box letting you know your project has been successfully merged pops up (shown below).

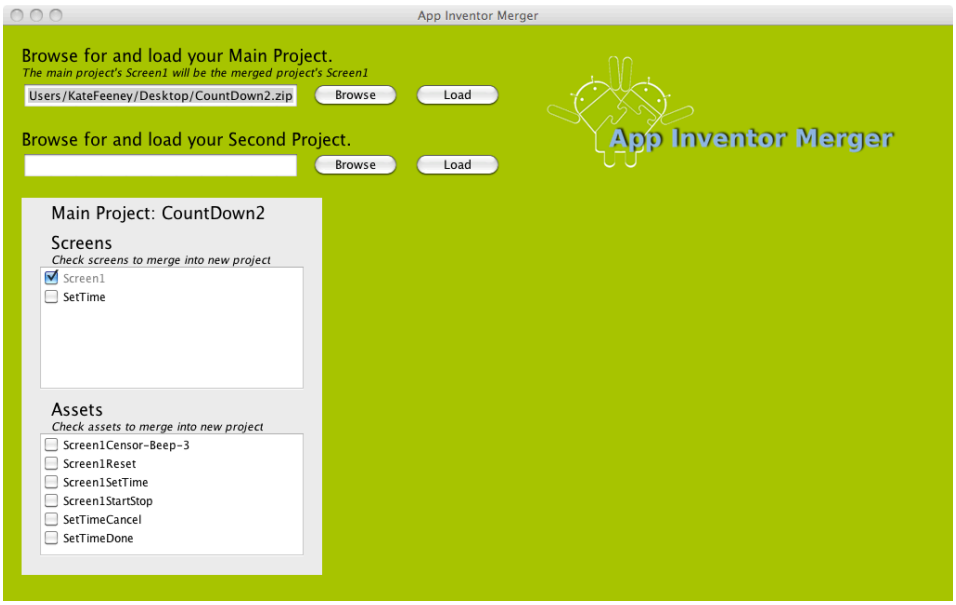


When there were only two developers we selected no but for the case of more than two developers select Yes since there are more projects to be merged. A new dialog box will appear asking if you would like to use the project you just created as the new main project.



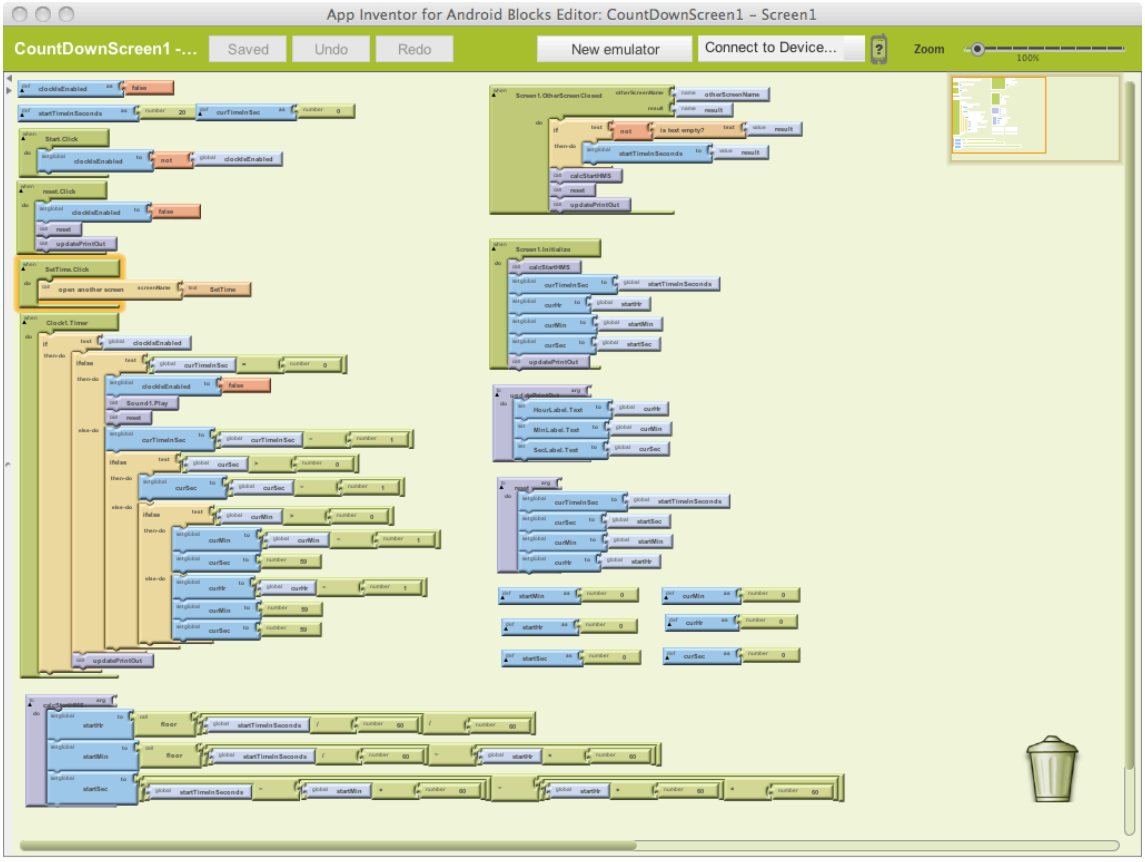
If you select No, the AIMerger will restart with no projects loaded, the same as if you just launched it. This would be used to merge two new projects together.

If you select Yes, the AIMerger will restart with the project you just created loaded as the main project and the second project is blank (as shown in the picture below). This option can be used if you would like to merge a third project with the first two projects you previously merged.

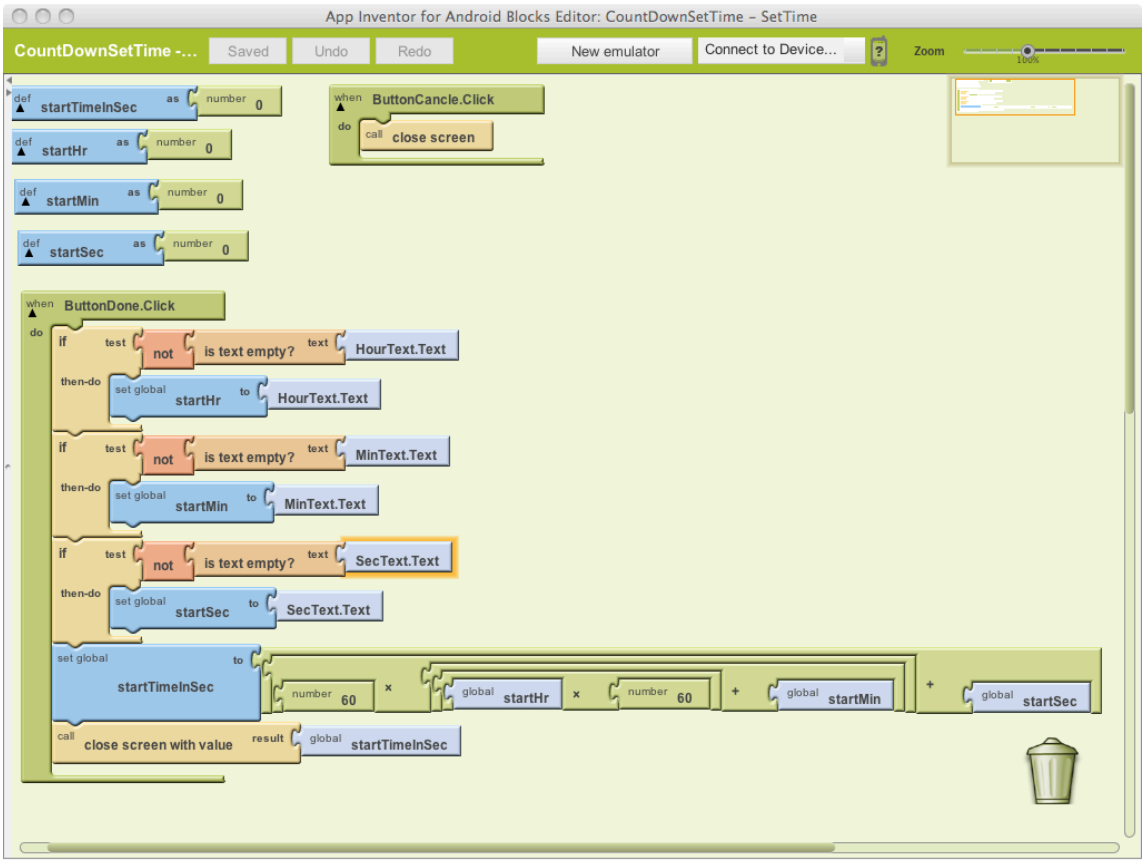


Appendix

Appendix A: Complete Blocks for Screen1 of CountDownScreen1



Appendix B: Complete Blocks for SetTime Screen



REFERENCES

Harmon, Trevor. Displaying a List of Checkboxes. 10 Febuary 1999. March 2012 <www.devx.com/tips/Tip/5342>.

Hasan, Sakibul. Install Apache Ant on Mac OS X; Sakibul Hasan. 30 July 2010. 24 January 2012 <http://sakibulhasan.wordpress.com/2010/07/30/install-apache-ant-on-mac-os-x/>.

Iridescent. Iridescent Learning. July 2012 <http://iridescentlearning.org/programs/technovation-challenge/>.

King, Rachael. Students Build Mobile Apps in Class - Businessweek. 6 February 2012. September 2012 <http://www.businessweek.com/videos/2012-02-06/students-build-mobile-apps-in-class>.

MIT Center for Mobile Learning. UNED, CSEV, Telefónica, Banco Santander and MIT Create the First Ibero-American Community for Digital Entrepreneurship. 8 June 2012. August 2012 <http://appinventor.mit.edu/explore/news/uned-csev-telef%C3%B3nica-banco-santander-and-mit-create-first-ibero-american-community-digital.html>.

Trail: Graphical User Interfaces (The Java™ Tutorials). 1 September 2012. March 2012 <http://docs.oracle.com/javase/tutorial/ui/index.html>.