App This or App That

Presenter Contact Info:
Bridgit Coley
Renaissance Middle Charter School
Mail Code: 6028
Phone: 9305)728-4622
Email: **Bcoley@recscharter.org**

For information concerning IMPACT II opportunites including Adapter and Disseminator grants, please contact:

The Education fund
305-892-5099, Ext. 18
Email: **Lvalle@educationfund.org**
Website: www.educationfund.org

Table of Contents

Standards/Objectives Covered in the Project

SC.8.N.1.6: Understand that scientific investigations involve the collection of relevant empirical evidence, the use of logical reasoning, and the application of imagination in devising hypotheses, predictions, explanations and models to make sense of the collected evidence.

A: Scientific inquiry is a multifaceted activity; The processes of science include the formulation of scientifically investigable questions, construction of investigations into those questions, the collection of appropriate data, the evaluation of the meaning of those data, and the communication of this evaluation.

B: The processes of science frequently do not correspond to the traditional portrayal of "the scientific method."

C: Scientific argumentation is a necessary part of scientific inquiry and plays an important role in the generation and validation of scientific knowledge.

D: Scientific knowledge is based on observation and inference; it is important to recognize that these are very different things. Not only does science require creativity in its methods and processes, but also in its questions and explanations.

SC.8.N.1.5: Analyze the methods used to develop a scientific explanation as seen in different fields of science.

A: Scientific inquiry is a multifaceted activity; The processes of science include the formulation of scientifically investigable questions, construction of investigations into those questions, the collection of appropriate data, the evaluation of the meaning of those data, and the communication of this evaluation.

B: The processes of science frequently do not correspond to the traditional portrayal of "the scientific method."

C: Scientific argumentation is a necessary part of scientific inquiry and plays an important role in the generation and validation of scientific knowledge.

D: Scientific knowledge is based on observation and inference; it is important to recognize that these are very different things. Not only does science require creativity in its methods and processes, but also in its questions and explanations

SC.8.N.2.2: Discuss what characterizes science and its methods.

A: Scientific knowledge is based on empirical evidence, and is appropriate for understanding the natural world, but it provides only a limited understanding of the supernatural, aesthetic, or other ways of knowing, such as art, philosophy, or religion.

B: Scientific knowledge is durable and robust, but open to change.

C: Because science is based on empirical evidence it strives for objectivity, but as it is a human endeavor the processes, methods, and knowledge of science include subjectivity, as well as creativity and discovery.

Tool Used: App Inventor

Description/Overview

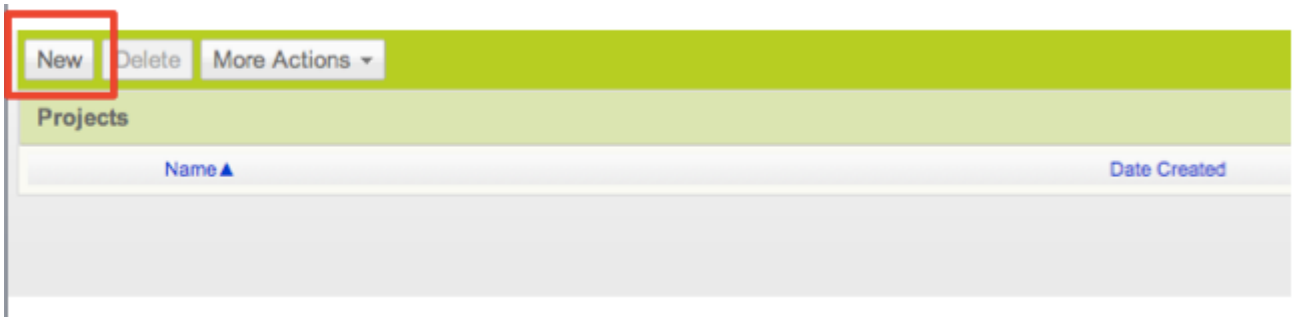
App Inventor lets you develop applications for Android phones using any web browser and either a connected phone or emulator. The App Inventor server stores your work and helps you keep track of your projects. You build apps by working with the following; App *Inventor Designer*, where you select the components for your app and the *App Inventor Blocks Editor*, where you assemble program blocks that specify how the components should behave. You assemble programs visually, fitting pieces together like pieces of a puzzle. Your app appears on the phone step-by-step as you add pieces to it, so you can test your work as you build. When you're done, you can package your app and produce a stand-alone application to install. If you don't have an Android phone, you can build your apps using the *Android emulator*, software that runs on your computer and behaves just like the phone.

Some other features that make the program great is the user- friendly interface and they provide a free book and curriculum, which is already developed. Students and teachers can test using phone or Android emulator only available with a paid subscription.

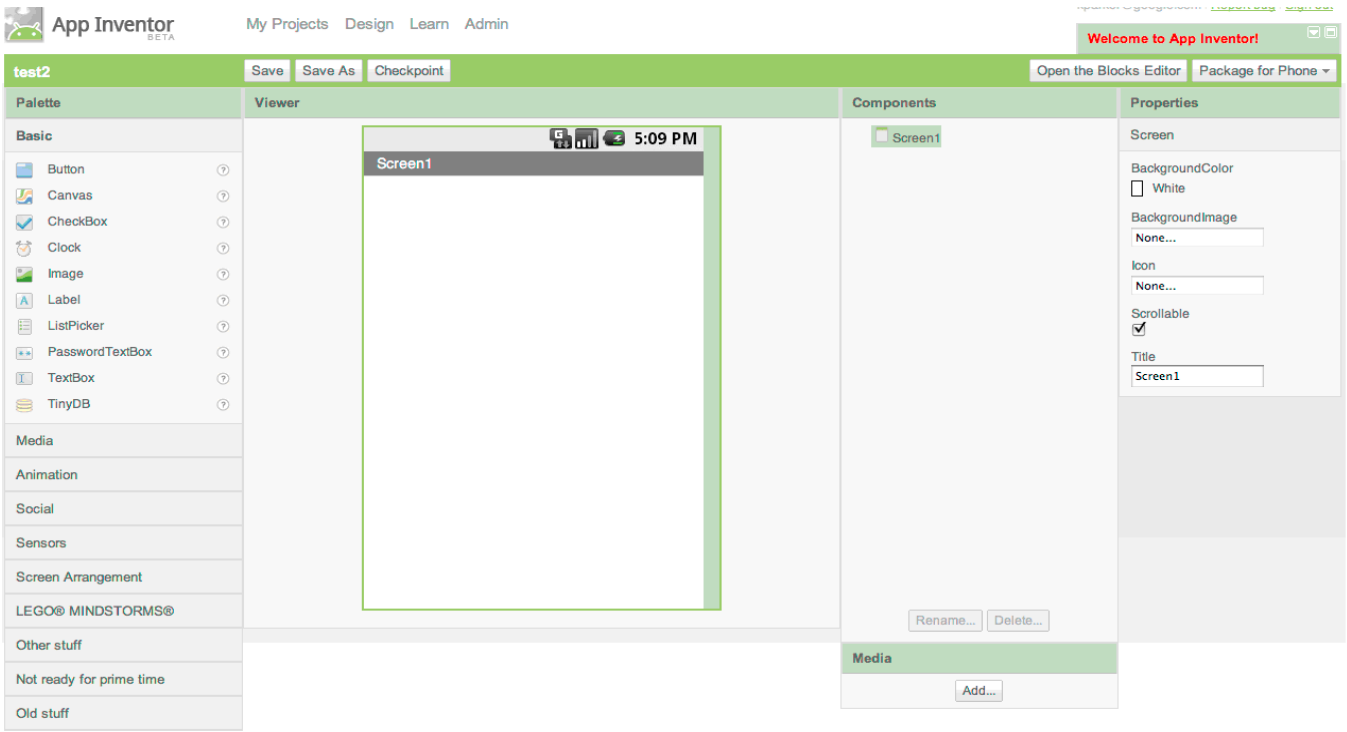
Any classroom would benefit in the BYOD mode because the added elements of development in Critical Thinking skills, computer programming and applying empirical data from topics of study. The tutorial that is explained in my packet on how to set up an app is available on App Inventor by Google. The registration is $25 but it's a very viable program that you can follow to create any app. All teachers and students need is a internet connection and a Google account. With both you can explore the app creation lessons and tutorials before creating anything.

Start the Designer and create a new project (Setup)

In your web browser, first go to the App Inventor website at **http://beta.appinventor.mit.edu/**. If this is the first time you are using App Inventor, you will see a blank projects page. A few things to do on the set up click **New** on the left side, near the top of the page. You can then enter the project name HelloPurr (one word, with no spaces) in the dialog box that appears, then click OK.



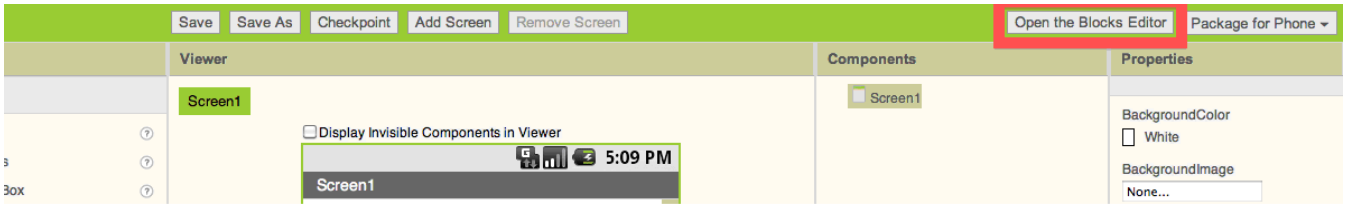
The browser will open the web page called the *Designer*, the place where you select components for your app and design the user interface, and should look like this:

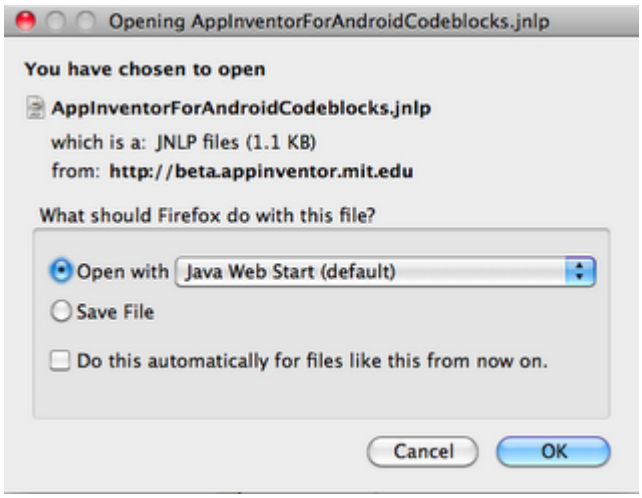


In addition to the Designer, you need to start the *Blocks Editor*, the place were you set the behavior of the app. It is a separate application with its own window, and therefore we need to open two windows to design an app. These windows are linked: changes made in the Designer will be immediately reflected in the Blocks Editor.

Starting the Blocks Editor

When you click **open the blocks editor** from the Designer window, the Blocks Editor program file should download and run.



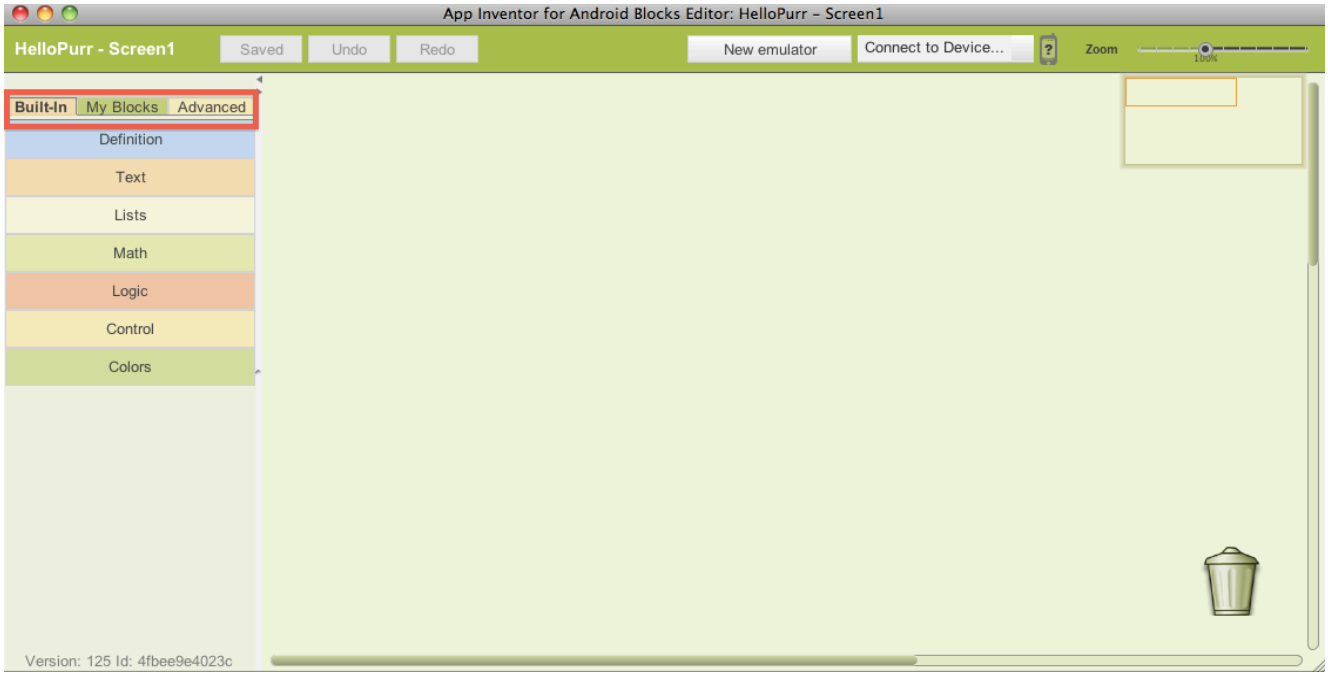


You may be prompted to accept the Blocks Editor file. Click "save", "allow", "keep", or whatever type of acceptance button comes up (it depends on your computer and browser). For Windows users, follow the instructions as the automatic installation wizard suggests. To open the Blocks Editor:

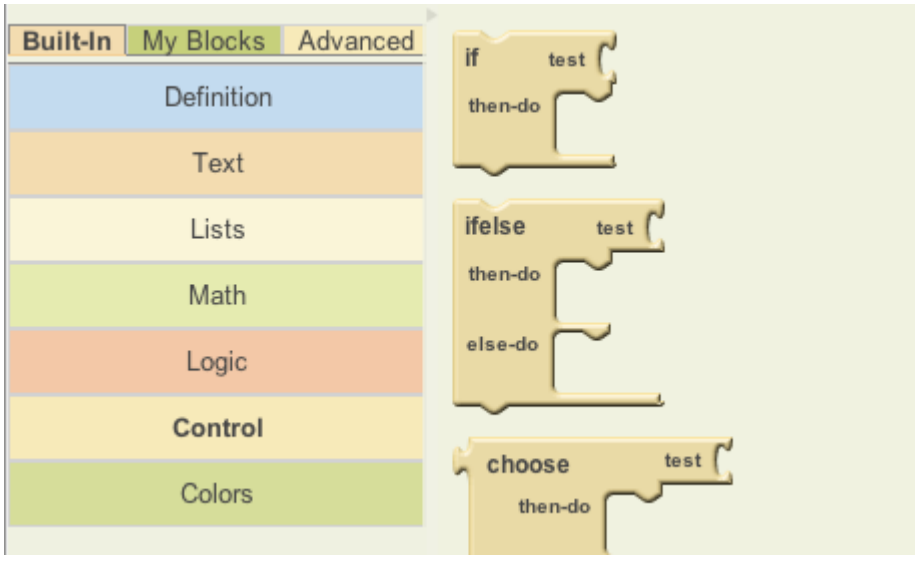
1. Click 'OK' (Open the Java file)
2. Click 'Always trust contents from this publisher' and open the file

3. Click 'Run'

This process may take 30 seconds or longer. If the Blocks Editor fails to open, one possible reason is that your browser is not set up to run downloaded Java applications automatically. To account for this possibility, find the downloaded file named *AppInventorForAndroidCodeblocks.jnlp* and open it. The Blocks Editor window should look like this:



The large empty "canvas" space on the right side is referred as the **work area**, in which you place blocks to assemble the program.



On the left side, it has three **palettes** (Built-In, My Blocks, Advanced); each palette has drawers storing sets of blocks. When you click a drawer, you can see all the blocks stored in that drawer. You can scroll down to see more blocks.

The **Built-In** palette contains the standard set of blocks that are available for every app you build (e.g., Definition, Text, Lists etc.). The blocks under the **My Blocks** palette contains specific blocks that are tied to the set of components that you have chosen for your app. The **Advanced** palette contains blocks for inventing intermediate and advanced apps with more complex logic.

The **Designer** runs from the browser, and the **Blocks Editor** runs from Java; however, they are linked. Therefore, even when you close your Blocks Editor window, all the information in the Blocks Editor is stored in the Designer. When you click the "Open the Blocks Editor" button, a new *.jnlp* file is downloaded to your computer, and you need to open the file again. When a new Blocks Editor opens, it will contain all the blocks you programmed before you closed the Blocks Editor.

As you build your app, you can do "live testing" on an emulator or an Android device. If you have an Android tablet or phone that you want to use while you develop apps, click 4A below. If you do not have an Android device, you can still see your app as you develop it, but with an on-screen "emulator" that comes with App Inventor (you've already installed it when you installed App Inventor). To build with the emulator, click 4B below.

Set up either an Android device or the Android emulator so that App Inventor can connect to it and show your app "live" while you build it.

Hello Purr

Building your first app: HelloPurr

Now that you've set up your computer and device, and you've learned how the Designer and the Blocks Editor work, you are ready to build the HelloPurr app. At this point, you should have the Designer open in your browser, the Blocks Editor open in another window (which will show up as a java coffee cup icon in your task bar or dock), and either an Android device or an Android emulator connected to the Blocks Editor. (See **Setup instructions** if you do not have these things running.)

HelloPurr: tap the kitty, hear him meow

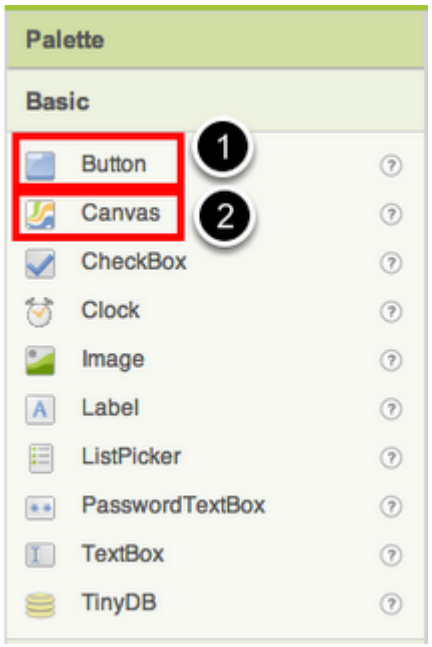
HelloPurr is a simple app that you can build in a very short time. You create a button that has a picture of a cat on it, and then program the button so that when it is clicked a "meow" sound plays.

To build HelloPurr, you'll need a image file of a cat and an audio file with a "meow" sound. Download these files to your computer by clicking the following links. To download: after clicking a link, right click on the image or sound bar and select "Save As." Save both files onto your desktop or downloads folder, or anywhere that you can easily find later.

- Kitty picture: **kitty.png** (Right-click and Save)
- Meow sound: **meow.mp3** (Right-click and Save)

You can also watch a **video** of this app being built.

Select components to design your app

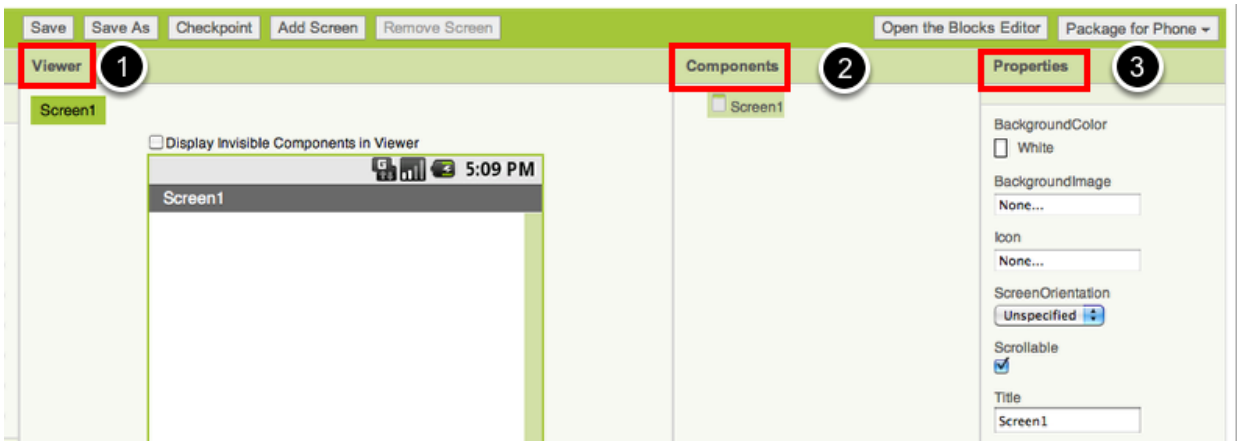


The App Inventor **Components** are located on the left hand side of the *Designer Window* under the title **Palette**. Components are the basic elements you use to make apps on the Android phone. They're like the ingredients in a recipe. Some components are very simple, like a **Label** component, which just shows text on the screen, or a **Button** component (#1 left) that you tap to initiate an action.

Other components are more elaborate: a drawing **Canvas** (#2 left) that can hold still images or animations, an **Accelerometer** sensor that works like a Wii controller and detects when you move or shake the phone, components that send text messages, components that play music and video, components that get information from Web sites, and so on.

To use a component in your app, you need to click and drag it onto the viewer in the middle of the **Designer**. When you add a component to the **Viewer** (#1 below), it will also appear in the components list on the right hand side of the Viewer.

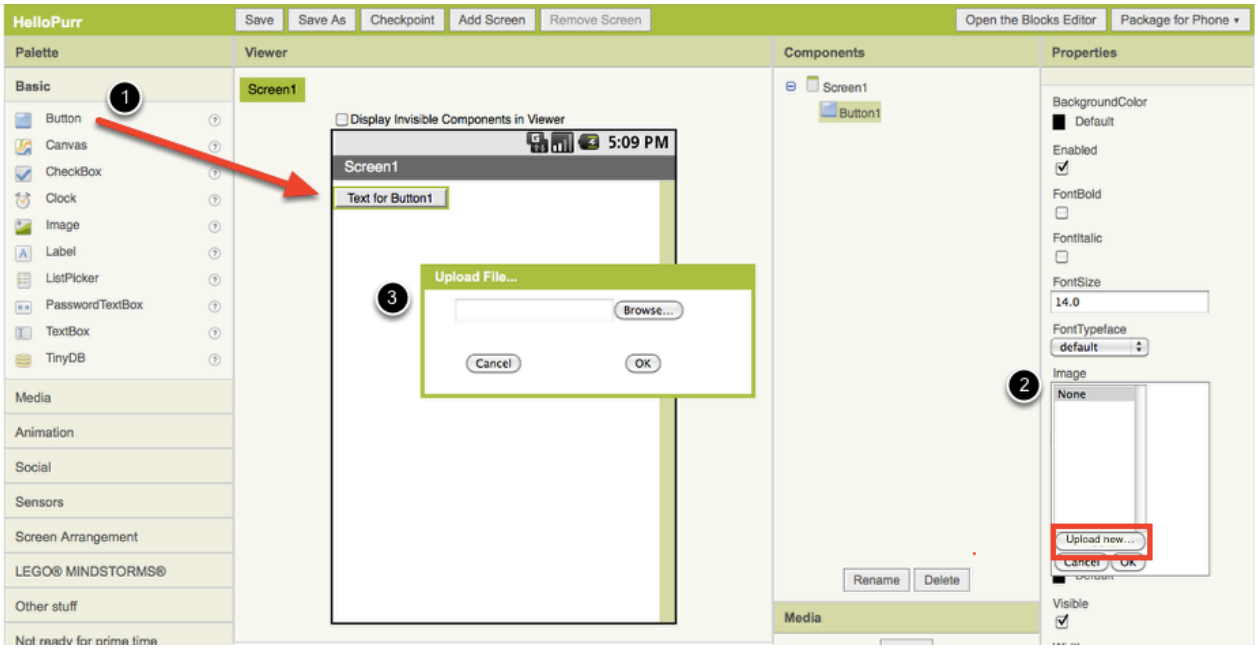
Components (#2 below) have properties that can be adjusted to change the way the component appears or behaves within the app. To view and change the properties of a component (#3 below), you must first select the desired component in your list of components.



Steps for selecting components and setting properties

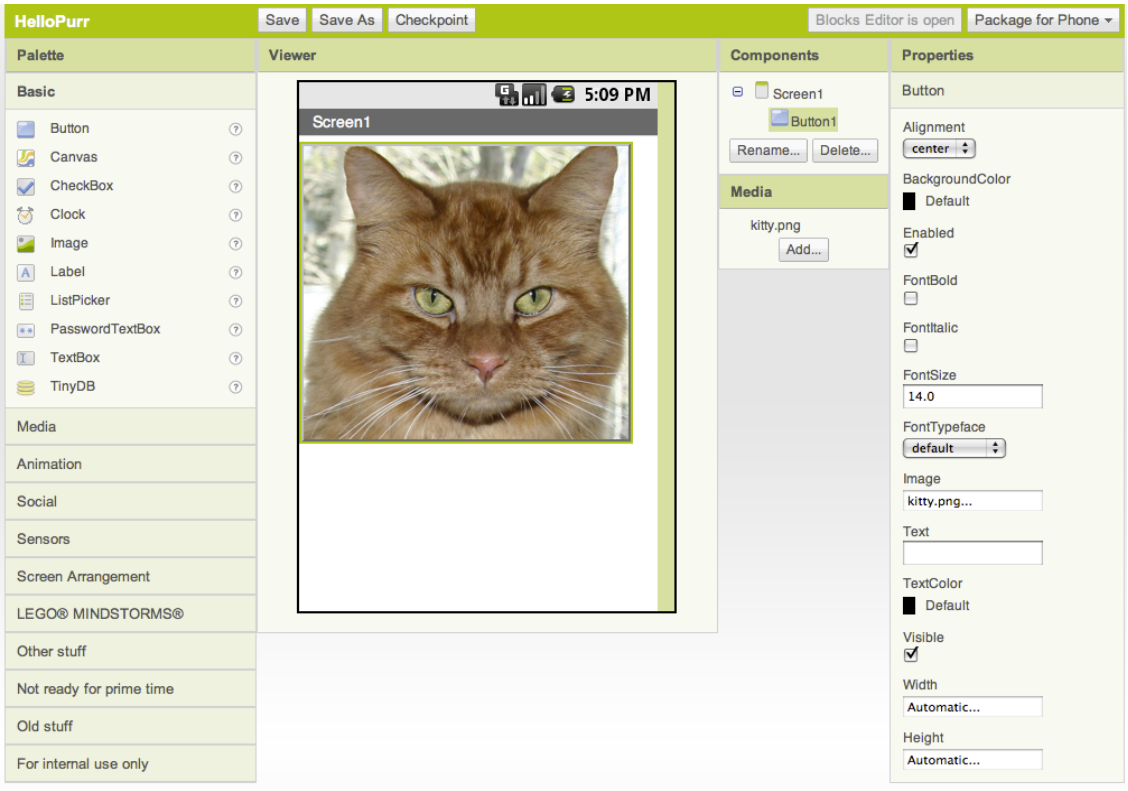
HelloPurr will have a **Button** component that displays the image of the kitty you downloaded earlier. To accomplish this:

Step 1. From the **Basic** palette, drag and drop the **Button** component to Screen1 (#1). To make the button have an image of a cat, in the **Properties** pane, under Image, click on the text "*None*..." and click "*Upload New*..." (#2). A window will pop up to let you choose the image file. Click "Browse" and then navigate to the location of the *kitty.png* file you downloaded earlier (#3). Click the *kitty.png* file, click "Open", and then click "OK".



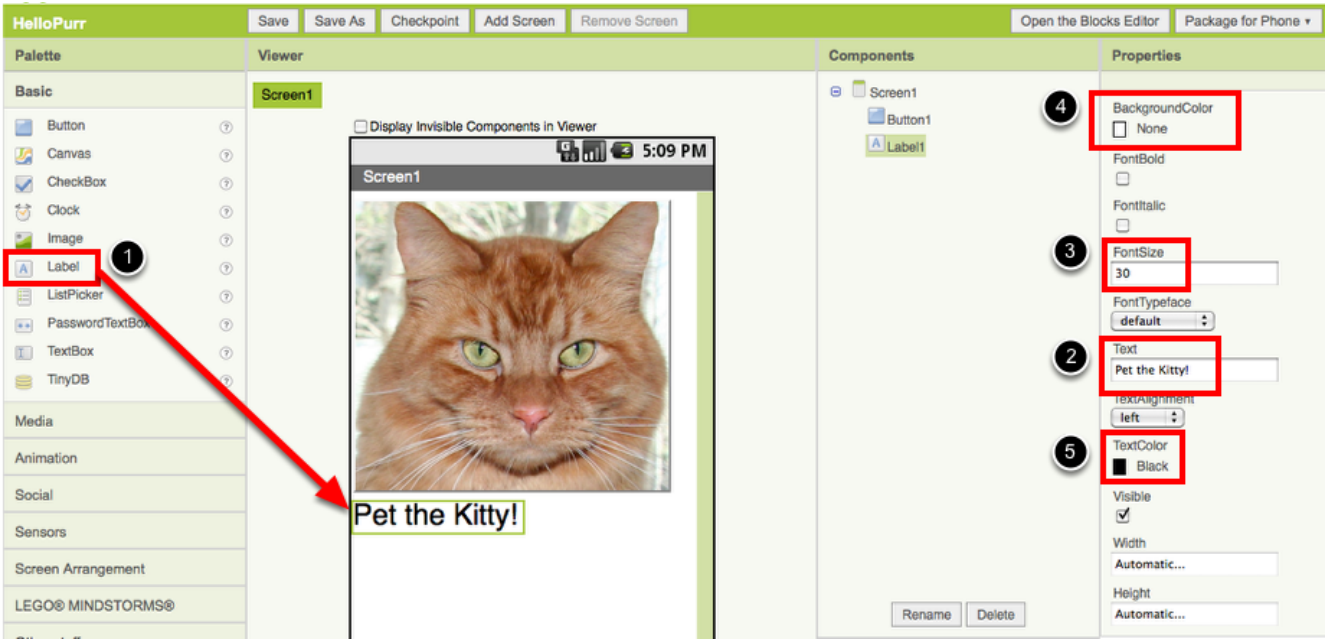
Step 2. Change the Button's **Text** property: Delete "Text for Button1", leaving the Button's text property blank so that there is no writing over the kitty's face. Your Designer should look like this:

If the entire kitty picture is not showing up, you can fix this by setting the Height and Width properties of the button to "Fill Parent". To do this, click on the Button component, go to the right-hand Properties pane, scroll down to the very bottom to where it says Width and click on the word "Automatic..." to activate the drop down list. Choose "Fill Parent". Do the same for the Height property.

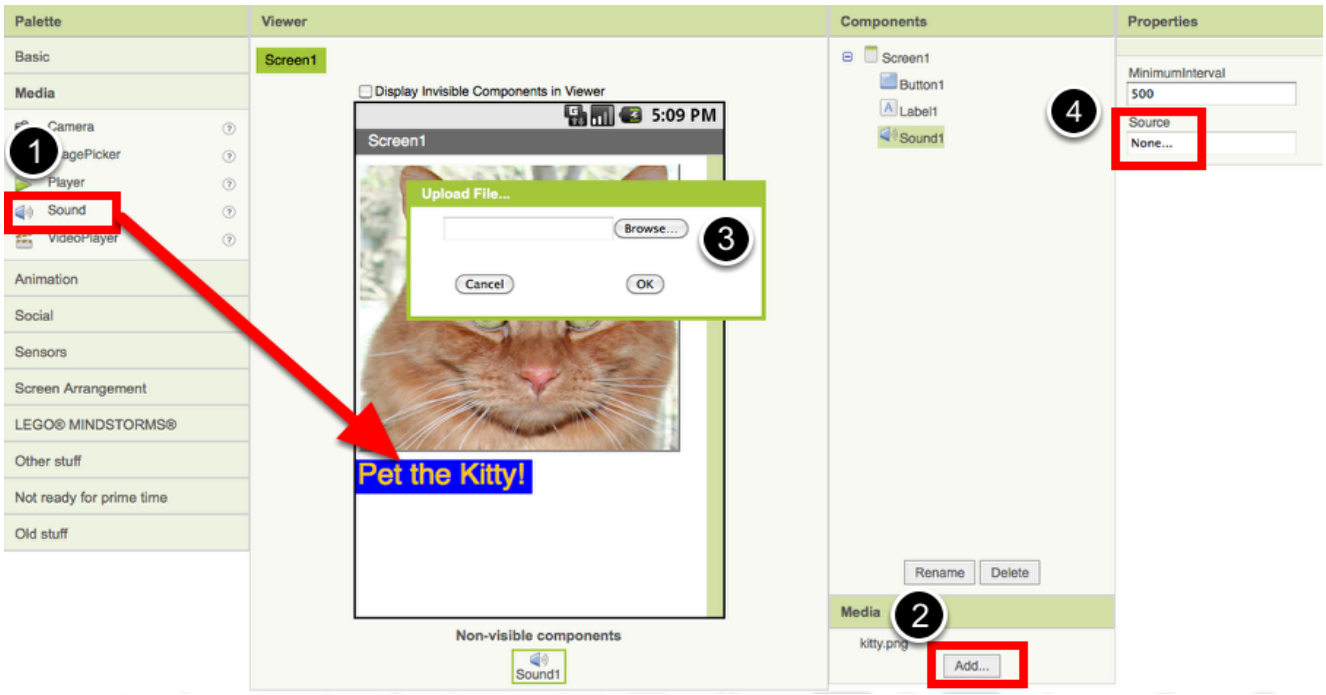


Step 3. From the **Basic** palette, drag and drop the **Label** component to the Viewer (#1), placing it below the picture of the kitty. It will appear under your list of components as Label1.

Under the **Properties** pane, change the **Text** property of Label1 to read "Pet the Kitty" (#2). You'll see the text change in the Designer and on your device. Change the FontSize of Label1 to 30 (#3). Change the BackgroundColor of Label1 by clicking on the box (#4): you can change it to any color you like. Change the TextColor of Label1 (#5) to any color you like. Here, the background color is set to blue and the text color is set yellow.



Step 4. Under Palette, click on the **Media** drawer and drag out a **Sound** component and place it in the Viewer (#1). Wherever you drop it, it will appear in the area at the bottom of the Viewer marked **Non-visible components**. Under the **Media** pane, Click *Add...* (#2) Browse to the location of the **meow.mp3** file that you downloaded earlier and upload it to this project (#3). Under the Properties pane, see that the **Source** property currently says *None...*. Click the word *None...* to change the Sound1 component's **Source** to *meow.mp3* (#4).



Programming with the Blocks Editor

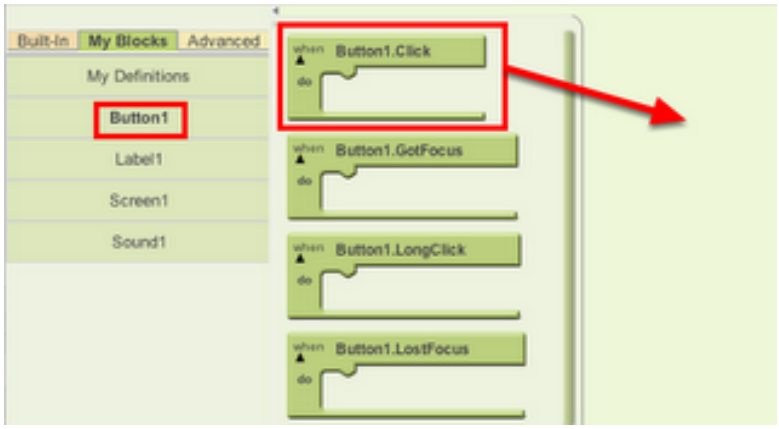
So far you have been arranging your app's screen and components in the *Designer*, which is in a web browser window. To start programming the behavior of the app, you need to go to the *Blocks Editor*. If you do not have a Blocks Editor running, click the button **Open the Blocks Editor** in the upper right of the Designer window. If you need help getting the Blocks Editor started, go back to the **setup instructions** for help.

Note: An easy way to switch between the Blocks Editor and Designer is to use the task bar that shows the applications running on your computer. The Blocks Editor is running locally as a java program and is represented by a Coffee Cup icon. The Designer is running in your web browser so you can get to it by clicking on your browser icon.

Once you have the Blocks Editor in front of you, continue to the next step to start programming your app with blocks.

Making the Sound Play

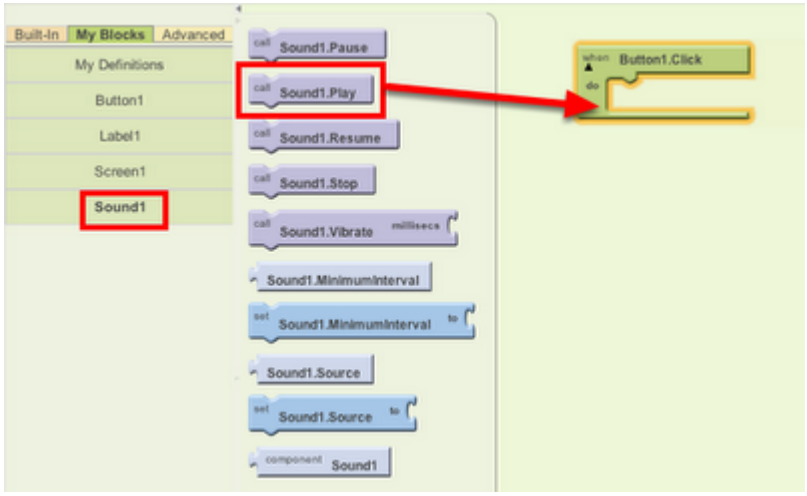
Step 1. Under the My Blocks palette on the left side of the Blocks Editor, click the **Button1** drawer to open it. Drag and drop the Button1.Click block in the work area (the open area on the right).



Those green blocks are called **event handler** blocks. The event handler blocks specify how the phone should respond to certain events: a button has been pressed, the phone is being shaken, the user is dragging her finger over a canvas, etc. The event handler blocks are green in color and use the word *when*. For example, when Button1.Click is an event handler

.

Step 2. Click the **Sound1** drawer and drag the Sound1.Play block and connect it to the "do" section of the when Button1.Click block. The blocks connect together like puzzle pieces and you can hear a clicking sound when they connect.



The purple and blue blocks are called **command** blocks, which are placed in the body of event handlers. When an event handler is executed, it runs the sequence of commands in its body. A command is a block that specifies an action to be performed (e.g., playing sound) when the event (e.g., pressing Button1) is triggered.

Your blocks should look like this at this point:

Now you can see that the **command block** is in the **event handler**. This set of blocks means; "when Button1 is clicked, Sound1 will play." The event handler is like a category of action (e.g., a button is being clicked), and the command specifies the type of action and the details of the action (e.g., playing sound a specified sound).

You can read more about how blocks work here: **Understanding Blocks**.

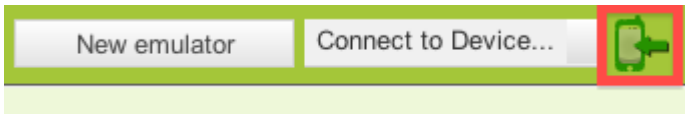
Try It! When you click the button you should hear the kitty meow. Congratulations, your first app is running!

Note: there is a known issue with the Sound component on some devices. If you see an "OS Error" and the sound does not play - or is very delayed in playing, go back into the Designer and try using a Player component (found under Media) instead of the Sound component.

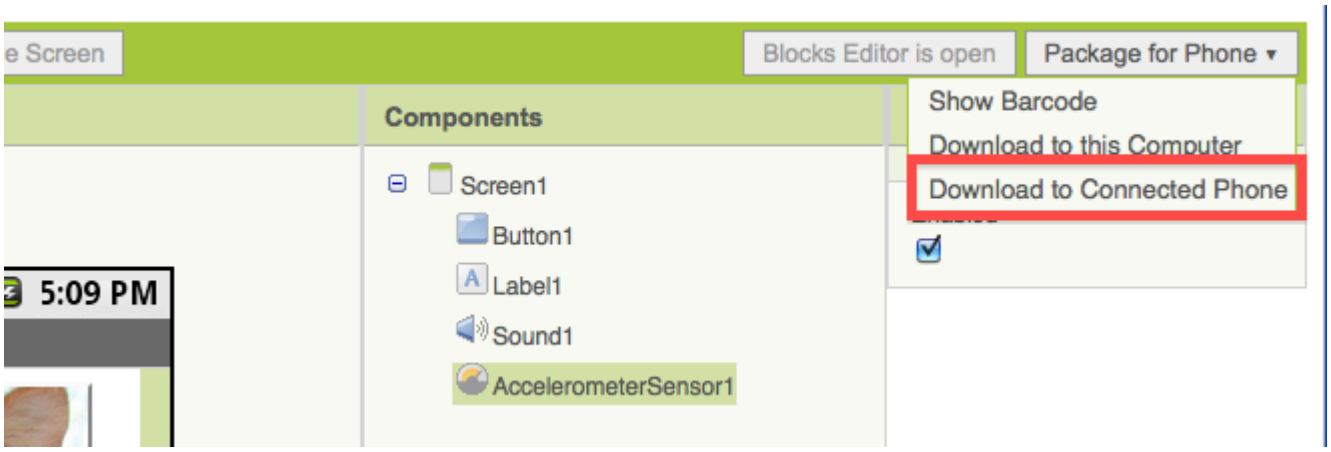
Packaging your app

While your device (emulator or phone/tablet) has been connected to App Inventor, your app has been running in real time on your device. If you disconnect the emulator/phone/tablet from the Blocks Editor, the app will vanish. You can always make it return by reconnecting the device. To have an app running without being connected to App Inventor, you must "**package**" the app to produce an application package (apk file).

To "package" the app to your phone, connect your phone to the Blocks Editor and make sure that the phone icon's color in the upper right of the Blocks Editor is green.



Then go back to the Designer and choose "Package for Phone" at the upper right of the designer page. App Inventor will present three options for packaging:



1. **Show Barcode:** You can generate a Barcode (a QR Code), which you can use to install the app on a phone or tablet that has a camera, with the aid of a barcode scanner, like the ZXing barcode scanner (freely available in Google Play).

Note: this barcode works only for your own device because it is associated with your google account. If you want to share your app with others via barcode, you'll need to download the .apk file to your computer and use third-party software to convert the file into a barcode. More information can be found **here**.

2. **Download to this Computer:** You can download the app to your computer as an apk file, which you can distribute and share as you like by manually installing it on other devices. (sometimes called **"side loading"**.

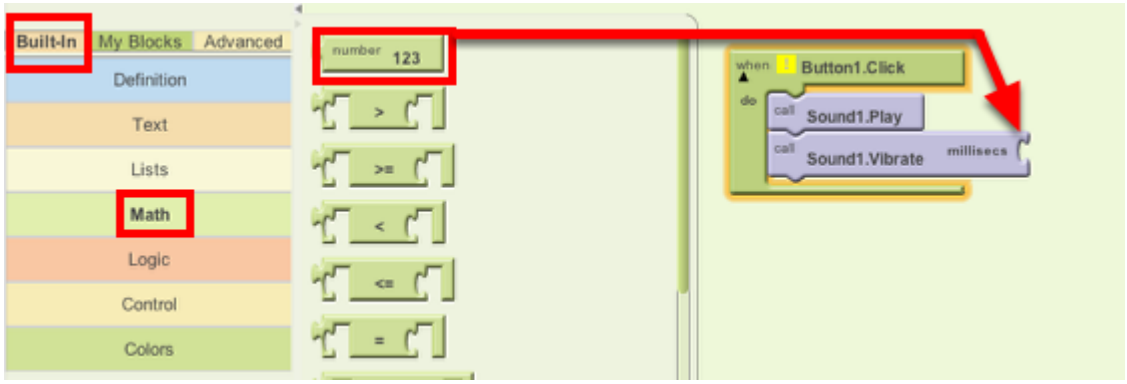
3. **Download to Connected Phone** You can download the apk file directly to the device that is connected to the Blocks Editor. Note that this works even if you are using the emulator as your device!

Challenge! Make the cat purr

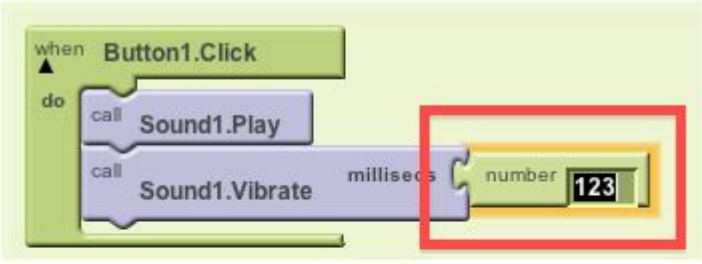
The challenge is to have the cat purr when the phone is shaken. Go to the Blocks Editor and open the **Sound1** drawer and drag the Sound1.Vibrate block and place it under the Sound1.Play block. You will see a yellow warning icon at the left corner of the block, which means the block has a missing component.

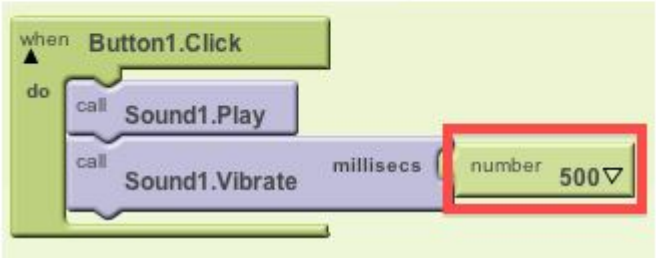
The Sound1.Vibrate block has an open slot, which means you need to plug something into it to specify more about how the behavior should work. Here, we want to specify the length of the vibration. Numbers are calculated in thousandths of a second (milliseconds): to make the phone vibrate for half a second, we need to plug in a value of **500 milliseconds**.

Go to the Built-In palette, go to the **Math** drawer, drag the number block and place it at the socket of the Sound1.Vibrate.

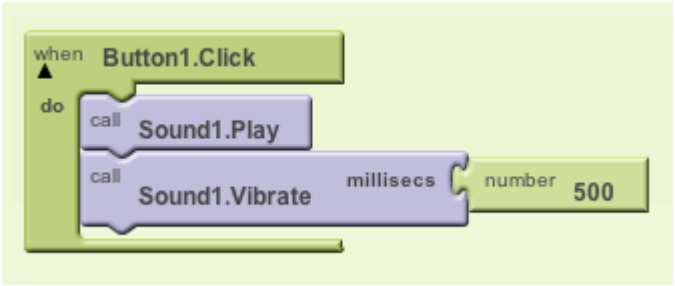


After you place the number block, click the number "123". It highlights the number in black: type "500" with your keyboard.





Done! Notice the yellow warning icon is gone: the block has no longer missing component.



Now connect your phone and tap the image of the cat on the phone. The phone should vibrate *and* meow at the same time.

Review

Here are the key ideas covered so far:

- You build apps by selecting components (ingredients) and then telling them what to do and when to do it.
- You use the **Designer** to select components and set each component's properties. Some components are visible and some aren't.
- You can add media (sounds and images) to apps by uploading them from your computer.
- You use the **Blocks Editor** to assemble blocks that define the components' behavior
- When ... do ... blocks define *event handlers*, that tell components what to do *when* something happens.
- Calls ... blocks tell components to do things.

Download Source Code

If you'd like to work with this sample in App Inventor, download the **source code** to your computer, then open App Inventor, go to the My Projects page, and choose **More Actions | Upload Source**.

Next steps……Now that you know the basics of how App Inventor works we recommend you:

1. Complete additional **Tutorials**.
2. Review the **Reference Documentation**.
3. Join the **User Discussion Forum**.
4. Read the guide to **Understanding Blocks**.
5. Or, if you've been using the emulator and want to start using your phone, you can **set up your Android device** to build apps.
   http://appinventor.mit.edu/explore/tutorials.html

Paint Pot (Part 1)

This tutorial introduces the **Canvas** component for creating simple two-dimensional graphics. You'll build an app that lets you draw on the phone screen in different colors.

What you're building

With the PaintPot app, you can:

- Dip your finger into a virtual paint pot to draw in that color.
- Drag your finger along the screen to draw a line.
- Tap the screen to make dots.
- Use the button at the bottom to wipe the screen clean.
- Include an image as a drawing background.

This tutorial assumes that you have completed the HelloPurr tutorial. It introduces the following App Inventor concepts:

- The **Canvas** component for drawing.
- Controlling screen layout with **Arrangement** components.
- Event handlers that take arguments.
- Variables.

Before starting

Make sure your computer and your phone are set up to use App Inventor. Start a new project in the Designer window, and name it "PaintPot". Open the Blocks Editor, click **Connect to Phone**, and make sure the phone has started the App Inventor app.

Screen title

To get started, go to the Properties panel on the right of the Designer and change the screen **Title** to "PaintPot". You should see this change on phone, with the new title showing in the title bar.

There are three names in App Inventor, and it's easy to confuse them:

1. The name you choose for your project as you work on it (in this case, PaintPot). This will also be the name of the application if you package it for the phone.
2. The name "Screen1", which is the name of the **Screen** component. You'll see it listed in the Components panel in the Designer. You can't change the name of the Screen component in the current version of App Inventor.
3. The **Title** property of the screen, which is what you'll see in the phone's title bar. Title is a property of the **Screen** component. The Title starts out being "Screen1", which is what you used in HelloPurr. However, you can change it, as you're doing for PaintPot. To reiterate, the name and the title of Screen1 are initially the same, but you can change the title if you want.

Set up the Components

You'll use these components to make PaintPot:

- Three **Buttons** for selecting red, blue, or green paint, and another button for wiping the drawing.
- A **Canvas**, the drawing surface. This canvas has a **BackgroundImage**, which is **this kitty** from the HelloPurr tutorial. You can also draw on a blank canvas. That's just a canvas without a background image.
- There's also a component you don't see: you use a **HorizontalArrangement** to make the three color buttons line up.

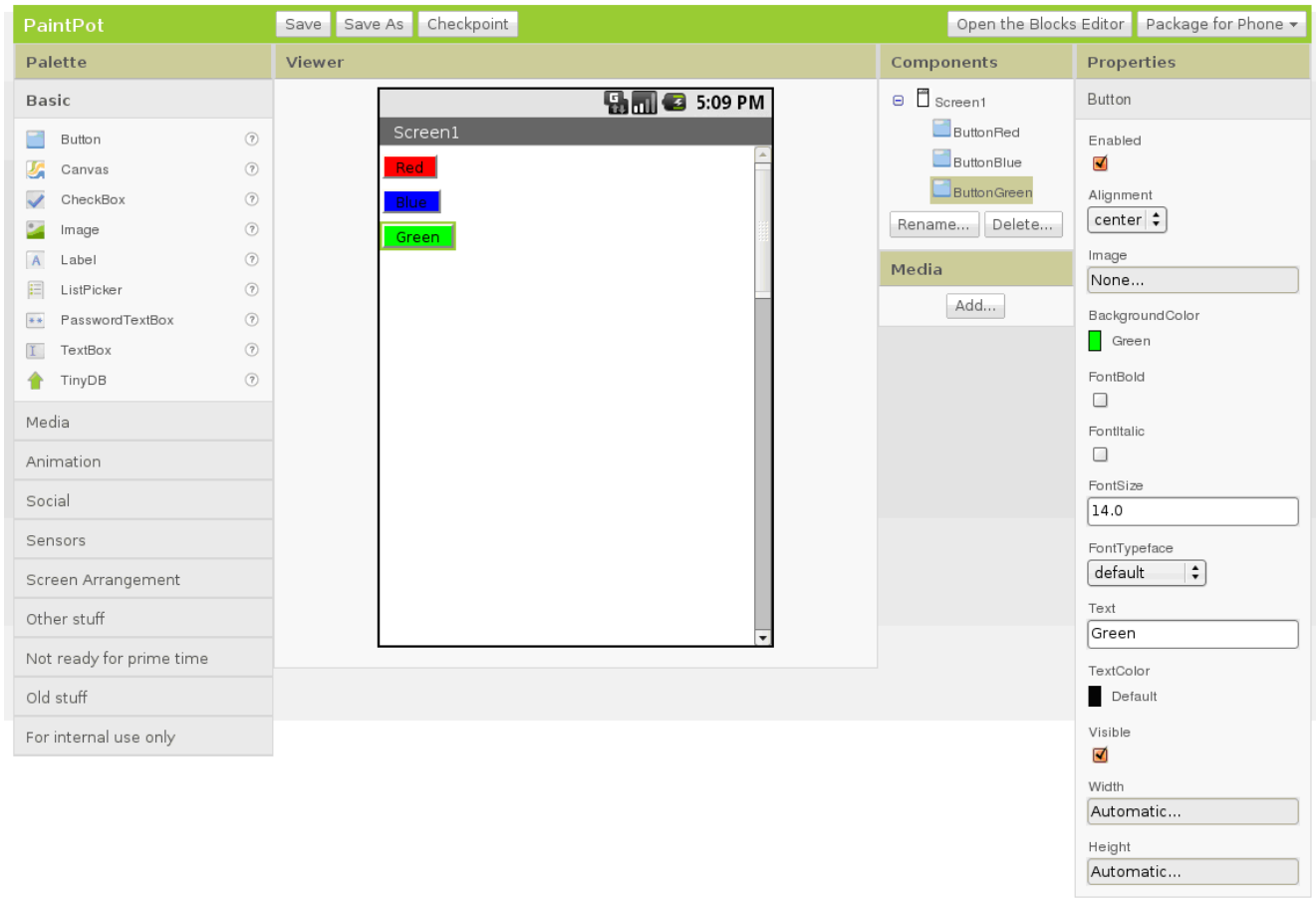
That makes five components in all. Let's get them and build the app.

Color Buttons

- Drag a **Button** component onto the Viewer and change the button's **Text** attribute to "Red" and make its **BackgroundColor** red.
- Click on Button1 in the components list in the Viewer to highlight it (it might already be highlighted) and use the **Rename...** button to change its name from "Button1" to "ButtonRed".

- Similarly, make two more buttons for blue and green, named "ButtonBlue" and "ButtonGreen", placing them vertically under the red button.

Here's how this should look in the designer, with the button names appearing in the list of project components. In this project, you're changing the names of the components rather than leaving them as the default names as you did with HelloPurr. Using meaningful names makes your projects more readable to yourself and others.



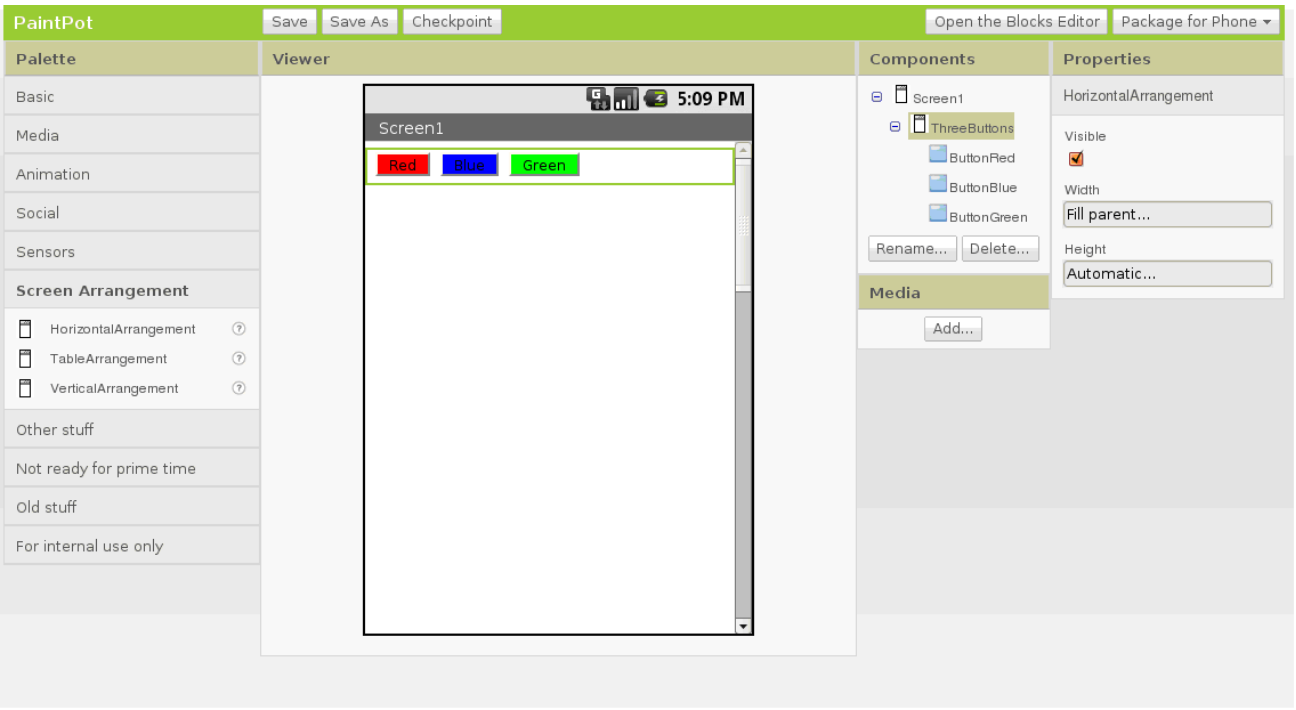
Layout with Screen Arrangement

You should now have three buttons, one above the other. The next step is to make them line up horizontally. You do this using a **HorizontalArrangement** component.

1. From the Palette's Screen Arrangement category, drag out a **HorizontalArrangement** component and place it under the buttons. Change the name of this component from "HorizontalArrangement1" to "ThreeButtons".

2. In the Properties panel, change the **Width** of ThreeButtons to "Fill Parent..." so that it fills the entire width of the screen.
3. Move the three buttons side by side into the **HorizontalArrangement** component. **Hint**: You'll see a blue vertical line that shows where the piece you're dragging will go.

If you look in the list of project components, you'll see the three buttons indented under the ThreeButtons to show that they are now its subcomponents. Notice that all the components are indented under Screen1.



You should also see your three buttons line up in a row on the phone screen, although things might not look exactly as on the Designer. For example, the Arrangement's outline shows in the Designer but not on the phone.

In general, you use Screen Arrangement to create simple vertical or horizontal layouts. You can create more complex layouts by nesting Screen Arrangement components. There is also a **TableArrangement** component (not covered in this tutorial).

Canvas and Wipe Button

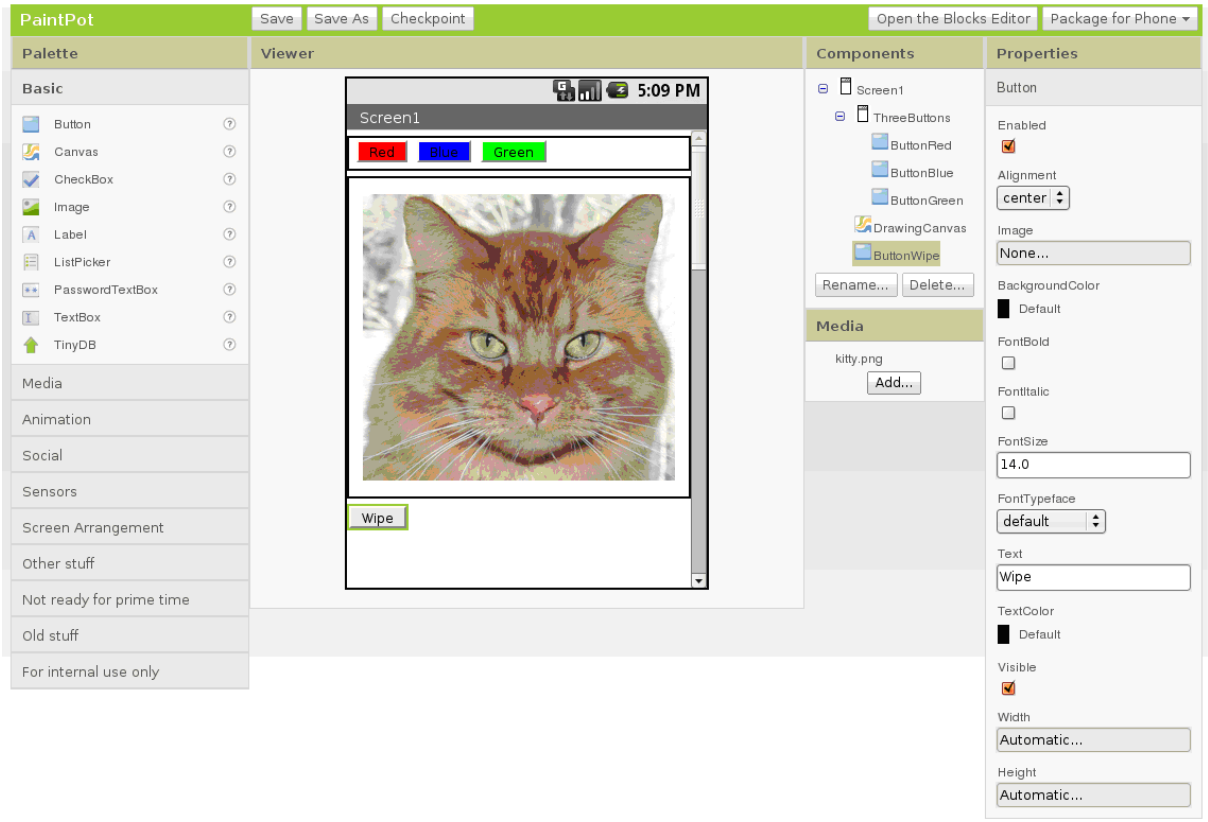
The final two components are the canvas and the wipe button.

1. From the Palette's Basic category drag a **Canvas** component onto the Viewer. Change its name to "DrawingCanvas". Set its **Width** to "Fill Parent" and set its **Height** to 300 pixels.
2. Add a Background Image to the **Canvas**. Click on the field containing "None..." next to **BackgroundImage** in the canvas's Properties panel. You can use the same kitty.png file, if you still have it on your desktop from an earlier tutorial. Or you can use another image.

> You can use any image you like, but you'll get the best results if the size of the image (in pixels) is close to the size at which you'll be showing it on the phone. Also, large images will take a long time to load, and might exceed the memory capacity of the phone allocates for applications.

3. From the Palette, drag the final button onto the screen, placing it under the canvas. Change its **id** to "ButtonWipe" and change its **Text** attribute to "Wipe".

You've now completed the steps to set the appearance of your app. Here's how this should look in the Designer. Next, you'll define how the components behave.



Add behaviors to the components

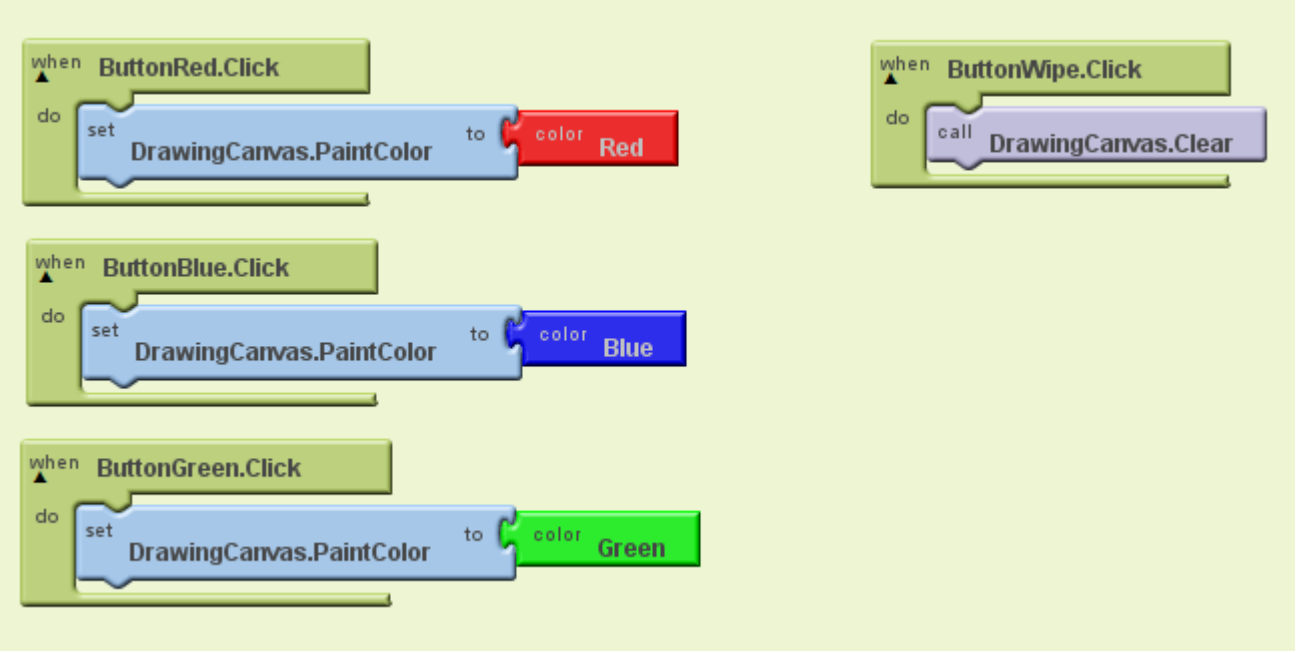
The Blocks Editor should already be open. First set up the buttons that change the paint color. Later you will add blocks to decide what happens when someone touches or drags the screen. Add button event handlers

In the Blocks Editor:

1. Switch to the My Blocks column.
2. Open the drawer for ButtonRed and drag out the when ButtonRed.Click block.
3. Open the DrawingCanvas drawer. Drag out the set DrawingCanvas.PaintColor to block (you may have to scroll the list of blocks in the drawer to find it) and place it in the do section of when ButtonRed.Click .
4. Switch to the Built-In Column. Open the Colors drawer and drag out the block for the color Red and put it into set DrawingCanvas.PaintColor to .
5. Repeat steps 2-4 for the blue and green buttons.
6. The final button to set up is the Wipe button. Switch back to the My Blocks column. Make a click event handler for ButtonWipe by dragging when ButtonWipe.Click from the ButtonWipe drawer. From the DrawingCanvas drawer, drag call

DrawingCanvas.Clear and place it in the do area of the when ButtonWipe.Click block.

The blocks for the buttons should look like this:



Add Touch-event Handlers

Now for the next step: drawing on the **Canvas**. You'll arrange things so that when you touch the canvas, you get a dot at the spot where you touch. If you drag your finger slowly along the canvas, it draws a line.

- In the Blocks Editor, open the drawer for the canvas and drag the when DrawingCanvas.Touched block to the workspace. As soon as you drag the block out, the three plugs on the right automatically fill in with name blocks called **x**, **y**, and **touchedSprite**.

You've already seen button click events. Clicks are simple; because there's nothing to know about the click other than that it happened. Other event handlers such as when ... Touched need information about the event. In App Inventor, this information is expressed as the value of arguments associated with the event handler. For the when ... Touched event, the first two arguments stand for the x and y coordinates of where the touch happened. We'll save **touchedSprite** for a later tutorial.

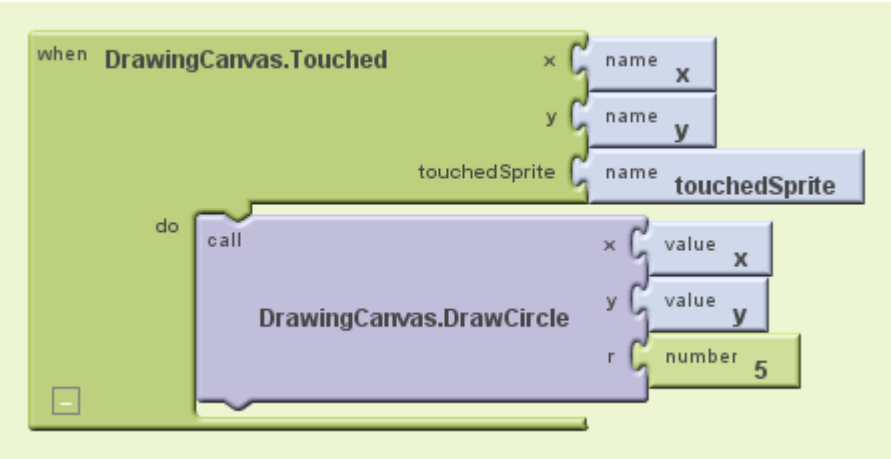
- For this touch event, make the canvas draw a small circle at the point with coordinates (x, y). Drag out a call DrawingCanvas.DrawCircle command from the canvas drawer and place it in the do section of when DrawingCanvas.Touched.

On the right side of the when DrawingCanvas.DrawCircle block are three sockets where you must specify values for the x and y coordinates where the circle should be drawn, and **r**, which is the radius of the circle. For **x** and **y**, you'll use values of the arguments that were supplied to the Touched handler:

1. Open the My Definitions drawer at the top of the column. Find the blocks for value x and value y. The blocks were automatically created when you dragged out the touch event handler block.
2. Drag out the value x and value y blocks and plug them into the corresponding sockets in the when DrawingCanvas.Touched block. Make sure to drag the value blocks, not the corresponding name blocks; they look very similar.
3. You'll also need to specify the radius of the circle to draw. Five (pixels) is a good value for this app. Click in a blank area of the screen to bring up the hover menu and select **math** (green). Select "123" from the dropdown list, to create a number block. Change the "123" to "5" and plug that in for the radius slot.

You can also just type 5 followed by return, to create a number block with a value of 5. This is an example of *typeblocking*: if you start typing, the Blocks Editor shows a list of blocks whose names match what you are typing; if you type a number it creates a number block.

Here's how the touch event handler should look:



Try out what you have so far on the phone. Touch a color button. Now touch the canvas, and your finger should leave a spot at each place you touch. Touching the Wipe button should clear your drawing.

Add Drag Events

Finally, add the drag event handler. Here's the difference between a touch and a drag:

- A touch is when you place your finger on the canvas and lift it without moving it.
- A drag is when you place your finger on the canvas and move your finger while keeping it in contact.

When you drag your finger across the screen, it appears to draw a giant, curved line where you moved your finger. What you're actually doing is drawing hundreds of tiny straight lines: each time you move your finger, even a little bit, you extend the line from your finger's immediate last position to its new position.

A drag event comes with 6 arguments. These are three pairs of x and y coordinates that show:

- The position of your finger back where the drag started.
- The current position of your finger.
- The immediately previous position of your finger.

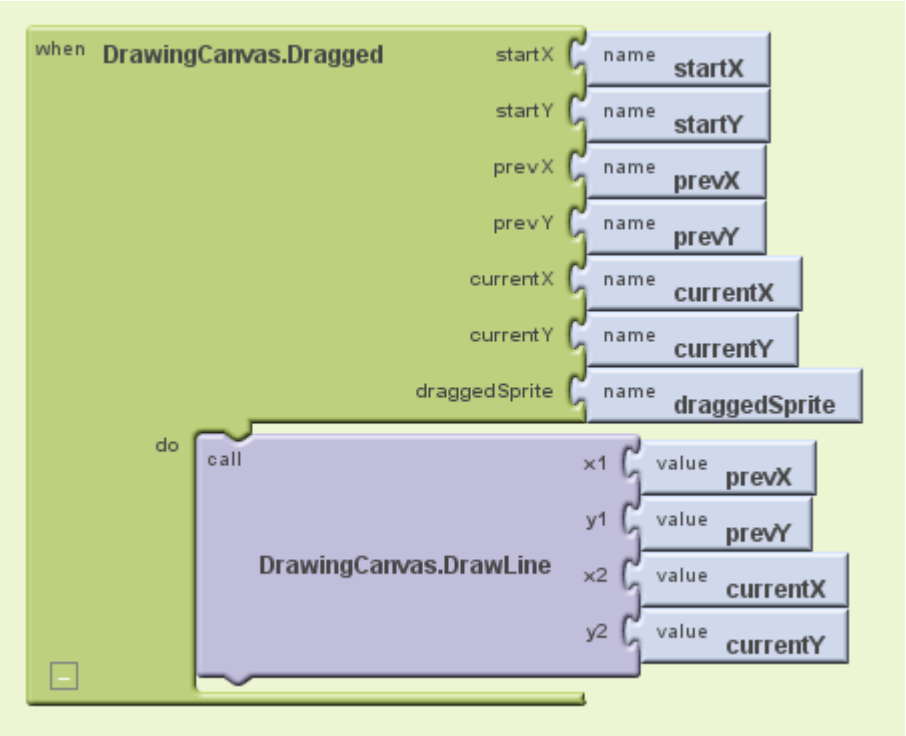
There's also a sprite, which we'll ignore for this tutorial. I can be addressed in another lesson.

Now make dragging draw a line between the previous position and the current position by creating a drag handler:

1. From the DrawingCanvas drawer, drag the when DrawingCanvas.Dragged block to the workspace.
2. Also from the DrawingCanvas drawer, drag the call DrawingCanvas.DrawLine block into the do slot of the when DrawingCanvas.Dragged block.
3. Click on the My Definitions drawer. You should see the blocks for the arguments you need. Drag the corresponding value blocks to the appropriate slots in when DrawingCanvas.Dragged: **x1** and **y1** should be **prevX** and **prevY**; **x2** and **y2** should be **currentX** and **currentY**.

To see if directions were followed you will see what it looks like on the next page. If it isn't correct back track and se what step was performed correctly.

Here's the result:



Test your work by trying it on the phone: drag your finger around on the screen to draw lines and curves. Touch the screen to make spots. Use the Wipe button to clear the screen.

Paint Pot (Part 2)

This project extends **Part 1** of the tutorial to create both large and small dots, as a demonstration of how to use *global variables*.

Starting

Make sure you've completed the **Set Up** process and you have your completed project from **PaintPot Part 1** loaded.

Start where you left off at the end of Part 1, with the project open in App Inventor. Use the **Save As** button to make a copy of PaintPot so you can work on the new version without affecting the original version. Name the copy "PaintPotV2" (with no spaces). After saving a copy, you should see PaintPotV2 in the Designer.

Creating variables

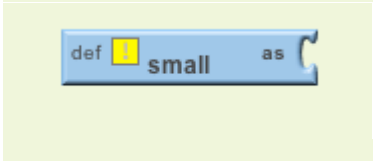
The size of the dots drawn on the canvas is determined in the when DrawingCanvas.Touched event handler where call Drawing. DrawCircle is called with **r**, the radius of the circle, equal to 5. To change the thickness, all we need to do is use different values for **r**. Use **r** = 2 for small dots and **r** = 8 for large dots.

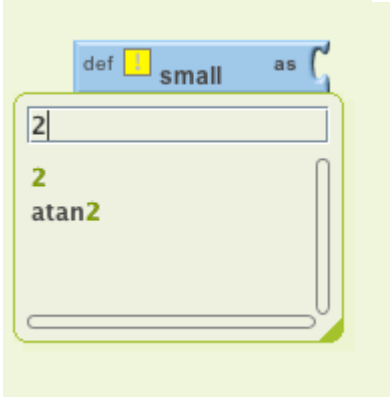
Start by creating names for these values:

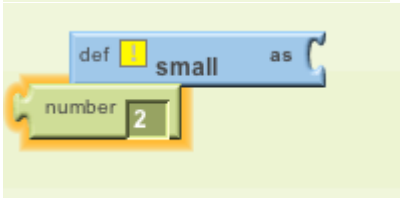
1. Open the Blocks Editor if it isn't already open and connect the phone. Your phone should show the buttons and the canvas you built previously.
2. In the Blocks Editor, in the Built-In column, open the Definitions drawer. Drag out a def variable block. Change the text that reads "variable" to read "small". A yellow warning exclamation mark will appear on the block. If you mouse over this you'll see a warning message explaining that the block has an empty socket.
3. You need to fill in the socket with a number block that specifies the value for "small" -- use 2 as the value. To create the number block, type the number 2. A menu will appear, showing you all the possible blocks that include "2" in their name. Click on the first one, which is the number 2 itself, and a number block with the value 2 should appear. Plug that in to the def variable block. The yellow warning mark will disappear, because the empty socket has been filled. (The second value listed in the menu is the math block atan2 which you won't use here.)

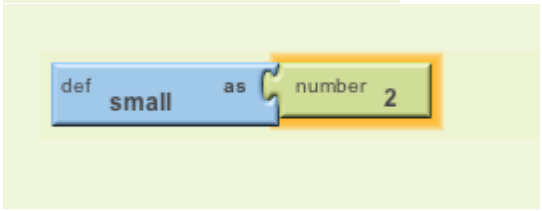
Here are the steps in the sequence:

1. 

2. 

3. 

4. 

5. 

You've now defined a global variable named small whose value is the number 2.

Similar to small, define a global variable big, whose value is 8.

Finally, define a global variable dotsize and give it an initial value of 2.

You might wonder whether it would be better programming style to make the initial value of dotsize be the value of small rather than 2. That would be true, except for a subtle programming point: Doing that would be relying on the assumption that small will already have a value at the point in time when dotsize is assigned its value. In App Inventor, you can't make assumptions about the order in which different def blocks will be processed. In general, of course, you really *would* like to specify the order in which variables are assigned. You can do this by assigning all values when the application is initialized, using the Screen initialize event. The **Quiz Me** tutorial gives an example of initialization.

Using variables

Now go back to the touch event handler you set up in Part 1 and change the call to DrawCircle block so that it uses the value of dotsize rather than always using 5.

In the Blocks Editor, switch to the My Blocks column, and open the My Definitions drawer. You should see six new blocks, two for each of the three variables defined:

- a global ... block that produces the value of the variable
- a set global ... block that sets the variable to a new value.

These blocks were automatically created, similarly to the way that the blocks for x and y were created when you defined the when DrawingCanvas.Touched event handler in the part 1 of this tutorial. "Global" means "global variable", in contrast to the event-handler arguments, whose blocks are labeled "value". The difference is that the argument values are accessible only within the body of the event handler, while global variables are accessible throughout the entire program.

- Go to the when MyCanvas.Touched event handler and replace the number 5 block in call DrawCircle with the global dotsize block from the MyDefinitions drawer.

Changing the values of variables

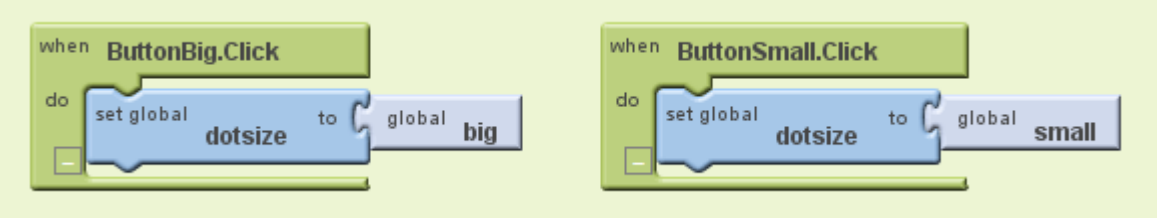
Now set up a way to change dotsize to be small (2) or big (8). Do this with buttons.

1. In the Designer, drag a **HorizontalArrangement** component into the Viewer pane below the DrawingCanvas component. Name the component "BottomButtons".
2. Drag the existing ButtonWipe into BottomButtons.
3. Drag two more button components from the Palette into BottomButtons, placing them next to ButtonWipe.
4. Name the buttons "ButtonBig" and "ButtonSmall", and set their **Text** to read "Big dots" and "Small dots", respectively.
5. In the Blocks Editor under My Blocks, create a when ... Clicked event handler for ButtonSmall that changes dotsize to be the value of small. To change dotsize use the

set global dotsize to block from the MyDefinitions drawer and plug in the global small block.

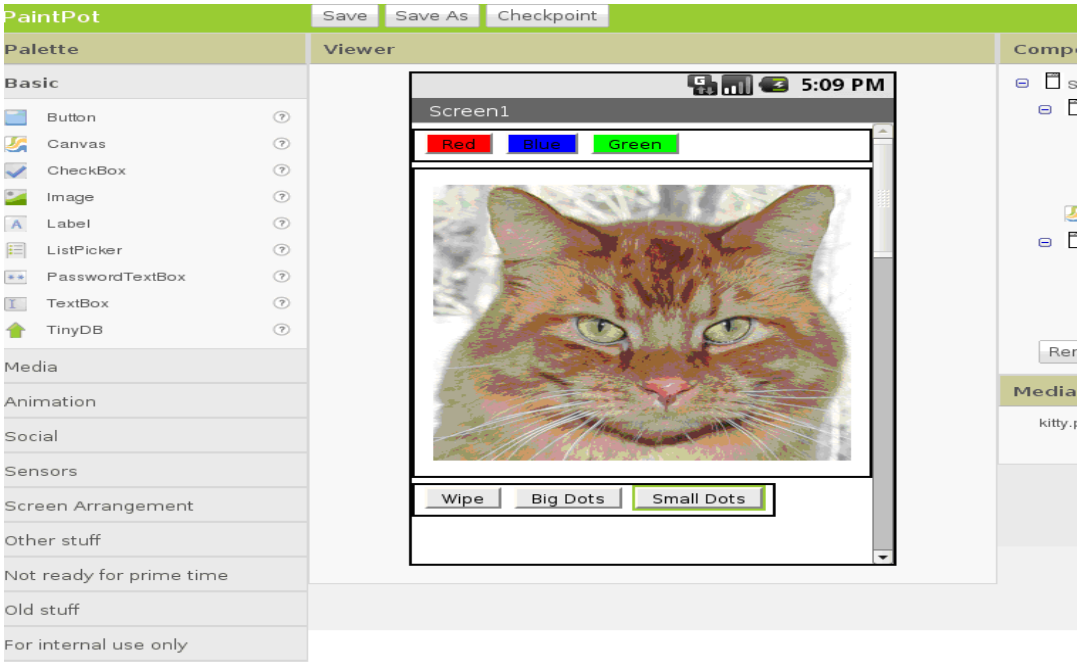
6. Make a similar event handler for ButtonBig.

The two click event handlers should look like this:

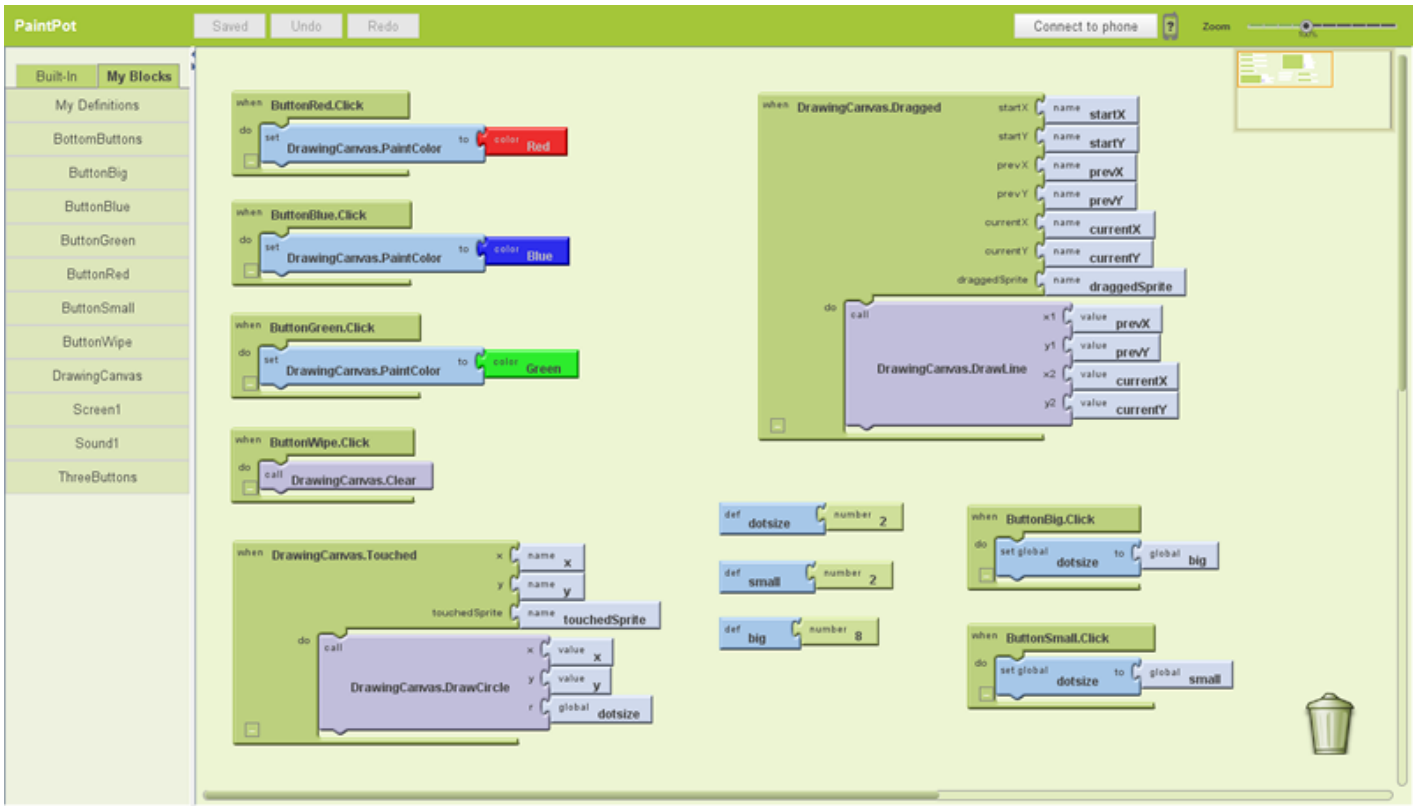


You're done! You can draw in PaintPot and use the new buttons to draw either big dots or small dots. Notice that dragging your finger still produces a thin line. That's because the changes we just made don't affect how DrawLine is called.

Here's the finished program in the Designer:



and in the Blocks Editor you should see:



A bug for you to work on: The program you just built has a slight bug. If you start drawing before pressing any of the paint buttons, the paint color will be black; however, after you choose a color, there's no way to get back to black. Think about how you could fix that.

Review

You create global variables by using def blocks from the Definitions drawer.

For each global variable you define, App Inventor automatically supplies a global block that gives the value of the variable, and a set global ... to block for changing the value of the variable. These blocks can be found in the My Definitions drawer.

Suggestions for further use:

1. Have students select an FCAT Science topic and create a quiz to allow all students to respond.

2. Have students create quantative data from any experiment so that they can guess or estimate then use the created app to compare.

3. Another idea is when teaching Punnett Square the can cross match parents to see probability of offspring outcome.

Other Resources for App Creations

1. **https://itunes.apple.com/us/course/ipad-iphone-app-development/id495052415**

2. **www.fullsail.edu/**

3. **http://appinventor.mit.edu/**

The Education Fund's

Adapter Grant Application

M-DCPS teachers, media specialists, counselors or assistant principals may request funds to implement an IMPACT II idea, teaching strategy or project from the Idea EXPO workshops and/or curriculum ideas profiled annually in the *Ideas with IMPACT* catalogs from 1990 to the current year, 2013-14.

Most catalogs can be viewed at The Education Fund web site at www.educationfund.org **under the heading, Publications. How-to booklets for each idea can be accessed at** www.educationfund.org **under Publications. They are listed under Curriculum Idea Packets.**

- Open to all K-12 M-DCPS teachers, counselors, media specialists
- Quick and easy reporting requirements
- Grants range from $150 - $400.
- Grant recipients recognized at an Awards Reception in late January.

To apply, you must contact the teacher (the Disseminator) who developed the idea. Contact may be made by attending a workshop at the Idea EXPO given by the IMPACT II disseminator teacher.

Project funds are to be spent within the current school year or an extension may be requested. An expense report with receipts is required by June 15th.

APPLICATION DEADLINE: December 10th.

Apply online at www.educationfund.org.

For more information contact:
Lorna Pranger Valle
The Education Fund
305-892-5099, ext. 18;
Lvalle@educationfund.org