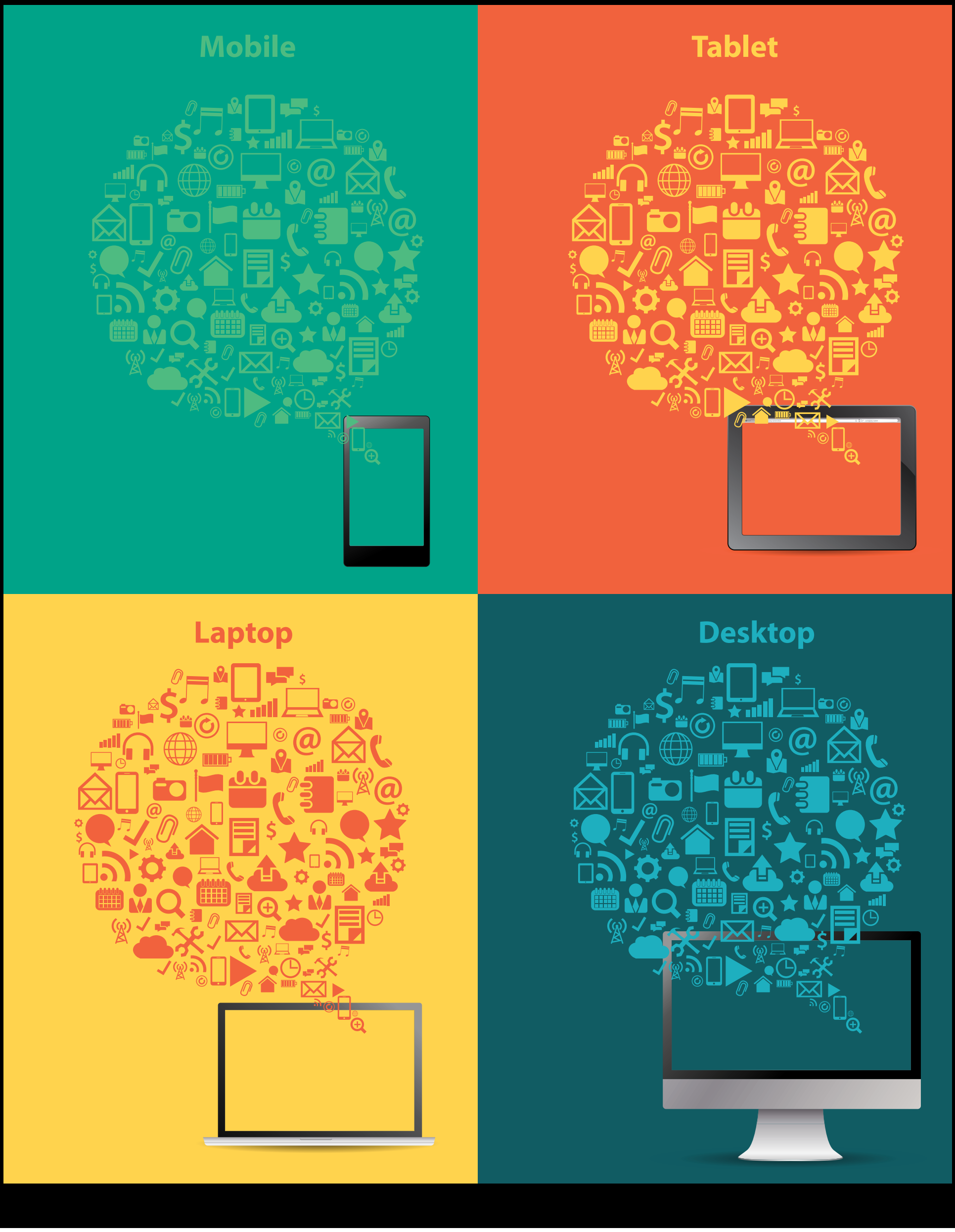
GLOBAL EDITION



Starting Out with App Inventor for Android

Tony Gaddis • Rebecca Halsey

ALWAYS LEARNING

PEARSON

STARTING OUT WITH

App Inventor for Android

First Edition
Global Edition

This page intentionally left blank.

STARTING OUT WITH App Inventor for Android

First Edition Global Edition

Tony Gaddis

and

Rebecca Halsey

PEARSON

Boston Columbus Indianapolis New York San Francisco
Upper Saddle River Amsterdam Cape Town Dubai London Madrid Milan
Munich Paris Montréal Toronto Delhi Mexico City São Paulo Sydney
Hong Kong Seoul Singapore Taipei Tokyo

Vice President and Editorial Director, ECS: *Marcia J. Horton*
Acquisitions Editor: *Matt Goldstein*
Editorial Assistant: *Kelsey Loanes*
Program Manager: *Carole Snyder*
Project Manager: *Rose Kernan, RPK Editorial Services, Inc.*
Project and Program Manager Team Lead: *Scott Disanno*
Media Team: *Steve Wright*
R&P Project Manager: *Rachel Youdelman*
Publishing Administrator and Business Analyst,
Global Edition*: Shokhi Shah Khandelwal*
Assistant Acquisitions Editor,
Global Edition*: Aditee Agarwal*
Assitant Project Editor, Global Edition*: Sinjita Basu*
Senior Manufacturing Controller,
Production, Global Edition*: Trudy Kimber*
Operations Specialist: *Vincent Scelta*
Full-Service Project Management*: iEnergizer Aptara®, Ltd.*
Cover Design: *Lumina Datamatics*
Cover Photo: *Shutterstock/My Life Graphic*
Cover Printer: *Ashford Colour Press*

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World Wide Web at: www.pearsonglobaleditions.com

Microsoft and/or its respective suppliers make no representations about the suitability of the information contained in the documents and related graphics published as part of the services for any purpose. All such documents and related graphics are provided "as is" without warranty of any kind. Microsoft and/or its respective suppliers hereby disclaim all warranties and conditions with regard to this information, including all warranties and conditions of merchantability, whether express, implied or statutory, fitness for a particular purpose, title and non-infringement. In no event shall microsoft and/or its respective suppliers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information available from the services.

The documents and related graphics contained herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Microsoft and/or its respective suppliers may make improvements and/or changes in the product(s) and/or the program(s) described herein at any time. Partial screen shots may be viewed in full within the software version specified.

Microsoft® and Windows® are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library

ISBN-13: 978-1-292-08032-1
ISBN-10: 1-292-08032-9

Typeset in Sabon LT Std by iEnergizer Aptara®, Ltd.

Printed and bound by Ashford Colour Press in the United Kingdom.

Brief Contents

Contents

Chapter 3 Input, Variables, and Calculations 153

Chapter 4 Decision Blocks and Boolean Logic 211

Chapter 5 Repetition Blocks, Times, and Dates 271

Chapter 9 Graphics and Animation 439

Chapter 10 Working with Text 485

Chapter 11 Text to Speech and Text Messaging 533

Preface

Cell phones have become an important part of most students' lives. Even students with limited computer experience have no trouble using their phones to send text messages, check their email, and update their Facebook statuses. Of course, the typical cell phone today is much more than a mere phone. It's a powerful computer with many unique capabilities, including the ability to run thousands of available programs, or apps.

Even though students regularly download, install, and use apps on their phones, they do not typically think of their phones as computers. In fact, students have a unique relationship with their phones that is different, and more personal, than the relationship they have with their laptop computers. When students learn that they can create their own mobile apps—especially apps that take advantage of a phone's unique capabilities (such as text messaging, location sensing, etc.)—they become excited and motivated to learn.

This book capitalizes on that excitement and motivation by using App Inventor 2 to teach introductory programming skills. App Inventor 2 is a free, cloud-based development platform that is provided by The MIT Center for Mobile Learning. It allows users with no prior programming experience to make their own Android apps. It is extremely easy to use, and it combines a visual GUI designer with a drag-and-drop code editor. An on-screen Android emulator or an actual Android device that is connected to the computer (either wirelessly or with a USB cable) runs apps as they are created. Because App Inventor 2 allows students to create apps and see them running on a phone, programming becomes a personally meaningful skill.

Programming With Blocks

For many beginning students, learning the syntax of a programming language can be a daunting task. Precious time that should be devoted to learning the fundamentals of programming is often spent tracking down missing semicolons or unbalanced braces.

Syntax errors in App Inventor are never a problem, because they never happen! You build an app by dragging and dropping "blocks" into an editor. The blocks, which represent actions and data, can be snapped together, like the pieces of a puzzle, to create fully functional programming statements. Because you don't have to spend time locating and fixing syntax errors, you can concentrate on planning the actions that you want your app to perform and arranging them into the proper sequence.

Runtime and logic errors can still occur, of course, because the student can use the wrong instruction or get instructions out of order. But because syntax is not an issue, the student devotes his or her time to developing and debugging algorithms.

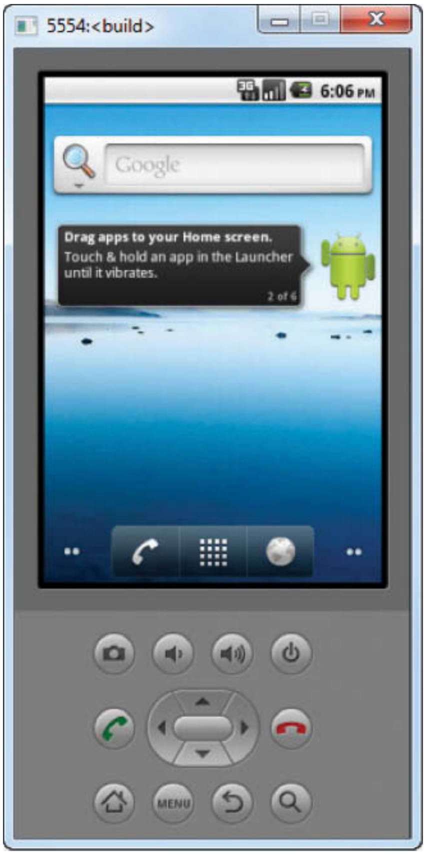
Using the Emulator or Android Devices

You use a Windows, Mac, or Linux computer to develop apps with App Inventor 2, but to test your apps, you use either the Android emulator, which is included with App Inventor, or an actual Android device such as a smartphone or a tablet. An Android device can be connected to the computer either wirelessly (via Wi-Fi) or with a USB cable. This book can be used with either approach.

The emulator, which is shown in Figure P-1, is a simulated Android phone. As you are using App Inventor to develop an app, the app appears and runs on the emulator's screen. You can interact with the emulator in many of the same ways that you interact with an actual smartphone. Although the emulator is limited (for example, it does not have a GPS sensor to report its location, and it cannot make phone calls), it does provide many of the basic features of an actual smartphone.

Most of the topics that are covered in Chapters 1 through 11 can be taught using the emulator. The topics covered in Chapters 12 and 13 require an Android device.

Figure P-1 The Android Emulator



Drag apps to your Home screen.
Touch & hold an app in the Launcher until it vibrates.
2 of 6

App Inventor in the Classroom

App Inventor can be used in a variety of ways in the classroom, and this text is designed to accommodate all of them. Here are some examples:

- You can use this text with App Inventor 2 for the first part of an introductory programming course, and then switch to a traditional programming language. Depending on the amount of time you want to devote to App Inventor, you can use the entire book, or you can omit some of the latter chapters.
- You can use this text with App Inventor 2 for a brief introduction to programming in a computer concepts course or an introduction to technology course. The latter chapters can be omitted to fit the amount of time that you have.
- You can use this text by itself in a semester-long course that uses only App Inventor 2 to teach programming fundamentals.
- You can use this text in short courses or summer programs that use App Inventor 2 to teach programming.

VideoNotes to Accompany This Book

A full set of VideoNotes has been developed to accompany each tutorial in the book. Students can follow along with the authors as they work through tutorials in the videos. Also, one exercise or programming project at the end of each chapter has an accompanying VideoNote that shows the student how to create the solution. To access these supplements, go to www.pearsonglobaleditions.com/Gaddis and click on the image of this book's cover.

Brief Overview of Each Chapter

Chapter 1: Introduction Programming and App Inventor 2

This chapter explains what algorithms and programs are, and why we use programming languages. App Inventor 2 is introduced and the student learns the fundamental steps for creating an app's user interface, using the Blocks Editor to program the app, and using the emulator to test an app.

Chapter 2: Working With Media

In this chapter, the student learns to create apps that use images and sound. Topics include setting the background image for the device's screen and displaying images in image components, as well as on buttons (to create clickable images). The Sound component is introduced for playing sound effects, and techniques for working with colors are presented. The chapter discusses the visual arrangement of components in the app's user interface and the importance of commenting code.

Chapter 3: Input, Variables, and Calculations

In this chapter, the student learns to use TextBox components to read user input. Variables are introduced as a way to store data in memory. App Inventor's math operator blocks are introduced, and the student learns to create math expressions. The Slider component is also discussed.

Chapter 4: Decision Blocks and Boolean Logic

In this chapter, the student learns about App Inventor's decision structures: the `if then` block, the `if then else` block, and the `if then else if` block. The relational operators are introduced, as well as logical operators. The chapter discusses random numbers, their applications, and how to generate them in App Inventor. The Screen component's `Initialize` event is introduced. The chapter concludes with a discussion of the ListPicker and CheckBox components.

Chapter 5: Repetition Blocks, Times, and Dates

This chapter shows the student how to use loops to create repetition structures. App Inventor's `while` and `for each` loops are presented. Counters, accumulators, and running totals are also discussed. The chapter introduces the Clock component as a way to work with dates and times, and also as a way to create a timer. The chapter concludes with a discussion of the DatePicker component.

Chapter 6: Procedures and Functions

In this chapter, the student first learns how to write procedures. The chapter shows the benefits of using procedures to modularize programs and discusses the top-down design approach. Then, the student learns to pass arguments to procedures. Finally, the student learns to write functions, or procedures that return a result.

Chapter 7: Lists

This chapter introduces lists. The student learns to create lists, insert and append items, select items at specific and random positions, remove items, replace items, search for items, and more.

Chapter 8: Storing Data on the Device

This chapter discusses the File component and the TinyDB component. The File component allows you to read and write text files on the device or emulator. TinyDB is a simple database component that allows you to store data as tag-value pairs.

Chapter 9: Graphics and Animation

App Inventor provides components for creating graphics and animations. In this chapter, the student first learns to draw primitive graphics with the Canvas component. Then, the Ball and ImageSprite components are discussed. Simple games are created that use collision detection, the Clock component, and sprites.

Chapter 10: Working with Text

In this chapter, the student learns to process strings at a detailed level. Various text-processing capabilities are discussed, such as concatenation, comparing strings, trimming strings, converting case, finding, replacing, and extracting substrings, and string splitting.

Chapter 11: Text to Speech and Text Messaging

This chapter begins with an introduction to the TextToSpeech component, which converts text to spoken words. (The component reads text aloud.) Next, the student learns to use the Texting component to send and receive text messages.

Chapter 12: Sensors

This chapter focuses on the sensors that are found on an Android device. The sensors that are introduced are: The LocationSensor, for determining the device's physical location, the OrientationSensor, for determining the device's orientation in 3D space, and the AccelerometerSensor, for determining the device's acceleration in 3D space. This chapter concludes with a discussion of using the ActivityStarter component to launch Google Maps.

Chapter 13: Other App Inventor Capabilities

This chapter presents various components that work on Android devices. The components that are covered in this chapter give capabilities such as recording audio, taking photos, selecting images from the device's gallery, playing videos, selecting entries from the contact list, scanning barcodes, using voice recognition, connecting to a Twitter account, and storing data on a Web server with a TinyWebDB component.

Appendix A: Setting Up App Inventor

Appendix B: Connecting an Android Device to App Inventor

Appendix C: Uploading Your Application to App Inventor Gallery and Google Play Store

Appendix D: Component Reference

Appendix E: Answers to Checkpoints

Features of the Text

Concept Statements

The major sections of the text starts with a concept statement. This statement concisely summarizes the main point of the section.

Example Apps

The text has an abundant number of complete and partial example apps, which are each designed to highlight the topic currently being studied.

Tutorials

Each chapter has several hands-on tutorials that lead the student through the process of developing or completing an app. These tutorials give the student experience performing the tasks discussed in the chapters.

VideoNotes

Online videos developed specifically for this book are available for viewing at www.pearsonglobaleditions.com/Gaddis. Icons appear throughout the text, alerting the student to videos about specific topics.

Notes

Notes appear at several places throughout the text. They are short explanations of interesting or frequently misunderstood points relevant to the topic at hand.

Tips

Tips advise the student on the best techniques for approaching different programming problems.

Checkpoints

Checkpoints are questions placed at intervals throughout each chapter. They are designed to query the student's knowledge quickly after learning a new topic.

Review Questions

Each chapter presents a thorough set of multiple-choice and short-answer review questions.

Exercises

Each chapter offers a set of exercises for developing apps. The exercises are designed to solidify the student's knowledge of the topics presented in the chapter.

Online Resources

This book's online resource page contains numerous student supplements. To access these supplements, go to www.pearsonglobaleditions.com/Gaddis and click on the image of this book's cover. You will find the following items:

- A link to the App Inventor site
- The book's example apps
- Graphics and audio files that can be used in student projects
- Access to the book's companion VideoNotes

Instructor Resources

The following supplements are available to qualified instructors only:

- Answers to the Review Questions
- Solutions for the exercises
- PowerPoint presentation slides for each chapter

Visit the Pearson Instructor Resource Center (www.pearsonglobaleditions.com/Gaddis) or send an e-mail to computing@pearson.com for information on how to access them.

Acknowledgements

The authors would like to thank Dr. Hal Abelson of MIT for his inspiring work, and particularly for creating App Inventor. We want to thank the entire App Inventor team at MIT for the amazing job they are doing. We also want to thank everyone at Pearson Education for making this book possible. We are extremely grateful that Matt Goldstein is our editor. He and Kelsey Loanes, editorial assistant, guided us through the process of writing this book. We are also fortunate to have Demetrius Hall and Bram Van Kempen as marketing managers. They do an amazing job of getting computer science books out to the academic community. The production team, lead by Camille Trentacoste, worked tirelessly to make this book a reality. We could not have done it without their patience and hard work. Thanks to you all!

Pearson wishes to thank and acknowledge the following people for their work on the Global Edition:

Contributor:
Passent M. El-Kafrawy, *Menoufia University, Egypt*

Reviewers:
Muthuraj M., *Android Developer, Bangalore*
Arup Bhattacharjee, *RCC Institute of Technology, India*
Soumen Mukherjee, *RCC Institute of Technology, India*
Manasa Rangarer, *NMAM Institute of Technology, Nitte, India*
Professor Harsh Bhasin, *Jamia Hamdard*

This page intentionally left blank.

About the Authors

Tony Gaddis

Tony Gaddis is the author of the *Starting Out with* series of textbooks. Tony has nearly twenty years of experience teaching computer science courses, primarily at Haywood Community College. He is a highly acclaimed instructor who was previously selected as the North Carolina Community College "Teacher of the Year" and has received the Teaching Excellence award from the National Institute for Staff and Organizational Development. The *Starting Out with* series includes introductory books covering C++, Java™, Microsoft® Visual Basic®, Microsoft® C#®, Python, Programming Logic and Design, and Alice, all published by Pearson Education.

Rebecca Halsey

Rebecca Halsey is an Associate Professor at Guilford Technical Community College where she teaches classes in Computer Science and Mobile Application Development. She is also developing and leading the new Mobile Application Development curriculum at GTCC. She also has twenty years of industry experience as a software developer.

This page intentionally left blank.

VideoNotes

CHAPTER 1

CHAPTER 2

CHAPTER 3

CHAPTER 4

CHAPTER 5

CHAPTER 6

CHAPTER 7

CHAPTER 8

CHAPTER 9

CHAPTER 10

CHAPTER 11

CHAPTER 12

CHAPTER 13

CHAPTER 1 Introduction to Programming and App Inventor

TOPICS

1.1 Introduction

This book teaches fundamental programming skills using an exciting application known as App Inventor 2. (We will refer to it simply as App Inventor.) App Inventor allows you to quickly and easily create applications, or "apps," for Android smartphones and tablets. It is not necessary to have prior programming experience or knowledge to use this book. App Inventor was created for beginners who have never programmed before.

You might find it surprising that with no previous programming experience, you can learn to create apps for a smartphone or a tablet. Perhaps you have heard that you need to know a lot about programming in languages such as Java to create mobile apps. While it is true that apps are typically created with high-level programming languages, App Inventor takes a different approach. With App Inventor, you use a screen designer to visually create an app's screen, as shown in Figure 1-1. Then, you use a special editor known as the Blocks Editor to create the actions that the app performs. With the Blocks Editor, you do not have to know a language such as Java to program the app. Instead, you visually assemble *code blocks* to create the app's actions. Figure 1-2 shows an example of the Blocks Editor.

With App Inventor, you use a standard computer, like a Windows PC, a Mac, or a Linux system, to create an app. You can connect a supported Android smartphone or tablet to the computer either wirelessly or with a USB cable. As you develop the app, you will see it running on the connected device. (See Appendix B for more information about connecting your Android device to App Inventor.)

Figure 1-1 The App Inventor Designer (*Source:* MIT App Inventor 2, Pearson Education, Inc.)

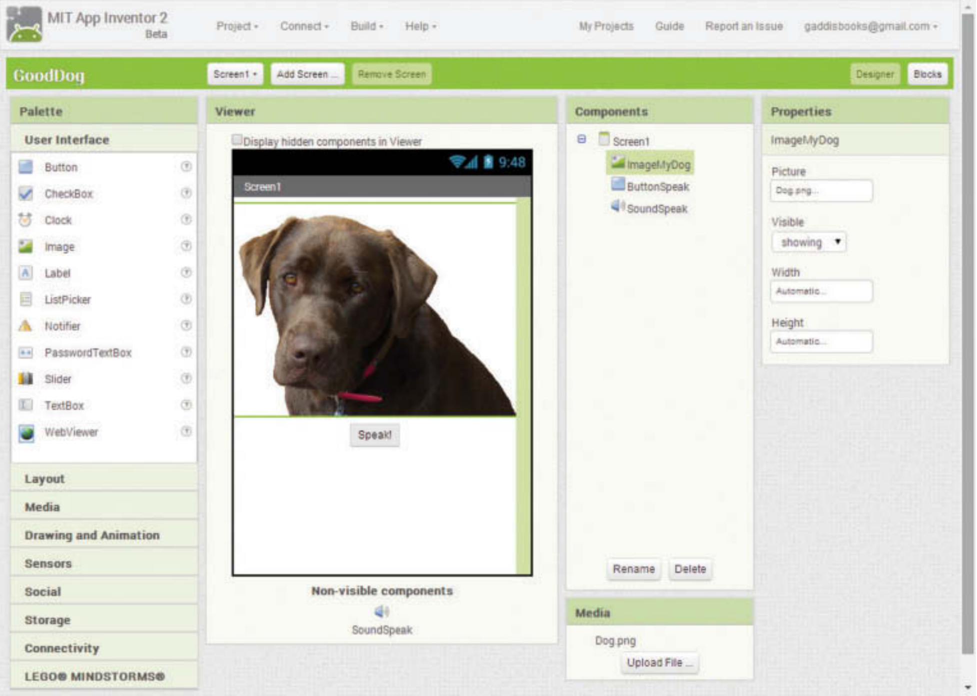
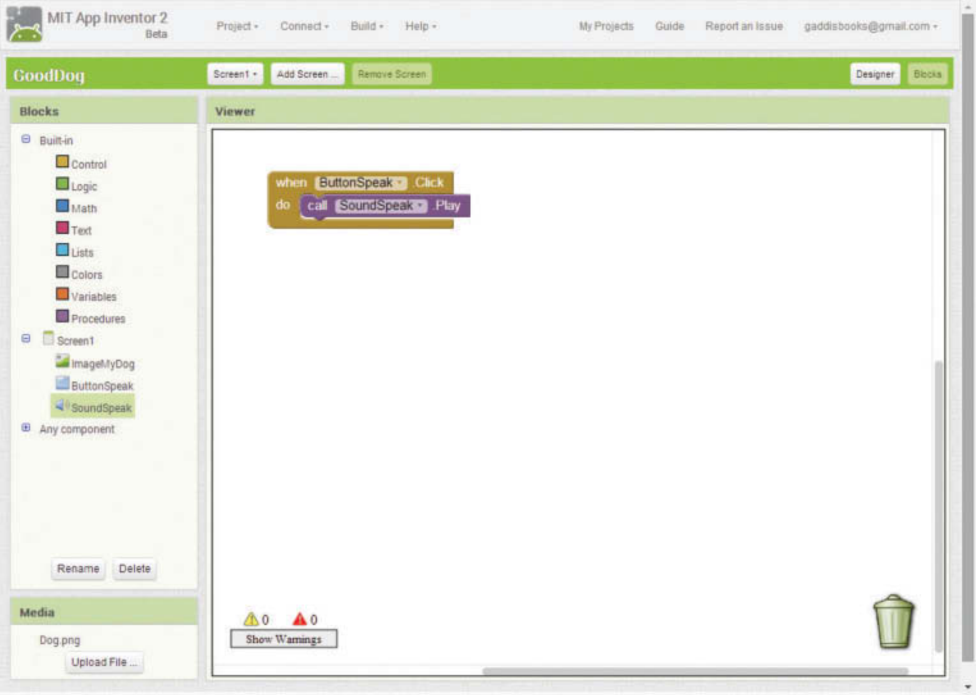


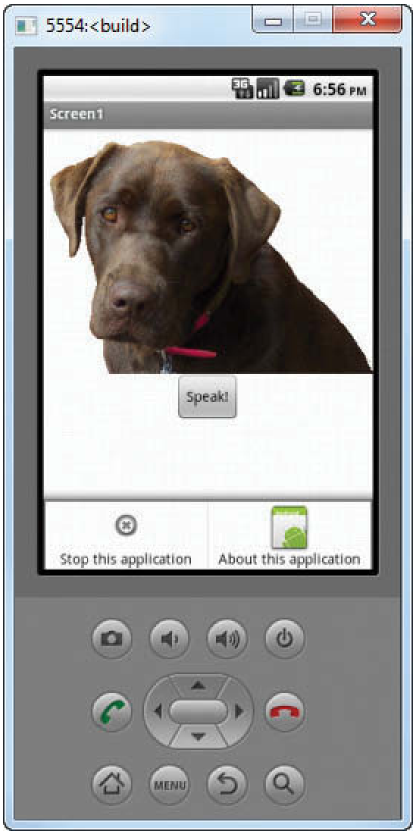
Figure 1-2 The Blocks Editor (*Source:* MIT App Inventor 2)



```
when ButtonSpeak ▾ .Click
do  call SoundSpeak ▾ .Play
```

If you do not have a supported Android device to connect to your computer, App Inventor provides an Android emulator that runs on your computer. The emulator, which is shown in Figure 1-3, is a simulated Android phone. As you are using App Inventor to develop an app, the app appears and runs on the emulator's screen. You can interact with the emulator in many of the same ways that you interact with an actual smartphone. Although the emulator is limited (for example, it does not have a GPS sensor to report its location, and it cannot make phone calls), it does provide many of the basic features of an actual smartphone.

Figure 1-3 The Android Emulator (*Source:* MIT App Inventor 2, Pearson Education, Inc.)



App Inventor Runs in the Cloud

Although you will need to install a program on your computer to run the Android emulator, App Inventor runs in the *cloud*. This simply means that it runs on a remote server that you are accessing over the Internet. App Inventor is part of MIT's Center for Mobile Learning, so it is hosted on servers that are managed by MIT. Additionally, the projects that you create with App Inventor are stored on the remote servers.

There are several advantages to this cloud-based approach. For example, you can access App Inventor and your projects from any computer that is properly set up and connected to the Internet. In addition, the files that you create with App Inventor are maintained and backed up by the host. Also, you can be sure that you are always running the most recent version of App Inventor. Of course, this approach requires that you have an Internet connection to use App Inventor.

Setting Up App Inventor

Before you can work through the tutorials in this book, you must set up App Inventor to work with either the Android emulator or an actual Android device. If you haven't already done so, turn to Appendix A and follow the instructions to set up App Inventor. Appendix A also has an accompanying VideoNote that demonstrates the set up process. You can access the VideoNote from the book's companion website at `www.pearsonglobaleditions.com/Gaddis`. If you have an Android device that you want to connect to App Inventor, read Appendix B after you have set up App Inventor on your computer.

1.2 What Is a Computer Program?

CONCEPT: **A computer program is a set of instructions that a computer follows to perform a task.**

Before jumping straight into App Inventor, you should take a moment to learn some basic concepts about computer programming. The concepts that we discuss in this section apply to all types of computer programming, regardless of whether the computer is a laptop, a supercomputer, or a mobile device.

The title of this section poses the question "What is a computer program?" Before we can answer that, first we should answer the question "What is a computer?" To learn programming, you do not need a deep understanding of how computers work, but you do need to understand in the most basic terms what a computer is. Here's a definition that we can start with:

A computer is a device that follows instructions.

A computer doesn't know how to do anything on its own. It only follows the instructions that are given to it. Having said that, you must realize that a computer cannot follow just any kind of instruction. For example, you can't wake up in the morning and say to your computer, "Make an omelet and serve it to me in bed." That's not an instruction that a computer can understand. That's the kind of instruction that a human (like a butler, if you're lucky enough to have one) can understand. Unfortunately, common computers like the ones you and I have on our desktops don't make breakfast. Their purpose is to work with data. They do things like adding and multiplying numbers, displaying data on the screen, storing data so it can be retrieved later, and so forth. Knowing this, we can expand our definition of what a computer is, as follows:

> *A computer is a device that follows instructions for manipulating and storing data.*

When a computer is designed, it is equipped with a set of operations that it can perform on pieces of data. Most of the operations are very basic in nature. For example, the following are typical operations that a computer can do:

- Add two numbers
- Subtract one number from another number
- Multiply two numbers
- Divide one number by another number
- Move a number from one memory location to another
- Determine whether one number is equal to another number
- And so forth . . .

A computer instruction is merely a command for the computer to perform one of the operations that it knows how to do.

Although an instruction exists for each operation that a computer is able to perform, the individual instructions aren't very useful by themselves. Because the computer's operations are so basic in nature, a meaningful task can only be accomplished if the computer performs many operations. For example, if you want your computer to calculate the amount of interest that you will earn from your savings account this year, it will have to perform a large number of instructions, carried out in the proper sequence. Now we can understand what a computer program is:

> *A computer program is a set of instructions that the computer follows to perform a task.*

So, if we want the computer to perform a meaningful task, such as calculating our savings account interest, we must have a *program*, which is a set of instructions. The instructions in a program must be carefully written so they follow a logical sequence. When a computer is performing the instructions in a program, we say that the computer is *running* or *executing* the program.

Algorithms and Programming Languages

Computer programmers do a very important job. Their job is important because without programs, computers would do nothing! When a programmer begins the process of writing a program, one of the first things he or she does is develop an algorithm. An *algorithm* is a set of well-defined, logical steps that must be taken in order to perform a task. For example, suppose we are writing a program to calculate an employee's gross pay. Here are the steps that should be taken:

1. Get the number of hours that the employee worked, and store it in memory.
2. Get the employee's hourly pay rate, and store it in memory.

3. Multiply the number of hours worked by the hourly pay rate and store the result in memory.
4. Display a message on the screen that shows the amount of money earned. The message must include the result of the calculation performed in Step 3.

Notice that the steps in this algorithm are sequentially ordered. Step 1 should be performed before Step 2, and so forth. It is important that these instructions are performed in their proper sequence.

The steps shown in the pay-calculating algorithm are written in English. Although you and I might easily understand the algorithm, it is not ready to be executed on a computer. The instructions have to be translated into *machine language,* which is the only language that computers understand. In machine language, each instruction is represented by a binary number. A *binary number* is a number that has only 1s and 0s. Here is an example of a binary number:

```
1011010000000101
```

When you or I look at this number, we see only a series of 1s and 0s. To the computer, however, this number is an instruction, which is a command to perform some operation. A computer program that is ready to be executed by the computer is a stream of binary numbers representing instructions.

As you can imagine, the process of translating an algorithm from English statements to machine language instructions is very tedious and difficult. To make the job of programming easier, special programming languages have been invented. *Programming languages* use words instead of numbers to represent instructions. A program can be written in a programming language, which is much easier for people to understand than machine language, and then be translated into machine language. Programmers use special software called *compilers* or *interpreters* to perform this translation.

Over the years, many programming languages have been created. If you are working toward a degree in computer science or a related field, you are likely to study languages such as Java, Python, C++ (pronounced "C plus plus"), and Visual Basic. These are only a few of the languages that are used by professional programmers to create software applications. Each of these languages has its own set of words that the programmer must learn in order to use the language. The words that make up a programming language are known as *keywords*. For example, the word `print` is a keyword in the Python 2 language. It prints a message on the screen. Here is an example of how the `print` keyword might be used to form an instruction in a Python 2 program:

```
print "Hello Earthling!"
```

This causes the message *Hello Earthling!* to be displayed on the computer screen. Compare this instruction to the binary number we saw earlier. You can see from this simple example why programmers prefer to use programming languages instead of machine language. Using words to write a program is much easier than using binary numbers.

In addition to keywords, programming languages have *operators* that perform various operations on data. For example, all programming languages have math operators that perform arithmetic. In Java, as well as most other languages, the + sign is an operator that adds two numbers. The following would add 12 and 75:

```
12 + 75
```

In addition to keywords and operators, each language also has its own *syntax*, which is a set of rules that must be strictly followed when writing a program. The syntax rules dictate how keywords, operators, and various punctuation characters must be used in a program. When you are learning a programming language, you must learn the syntax rules for that particular language.

When you write a program with a traditional programming language, you convert your algorithm into a series of *statements*. A programming statement consists of keywords, operators, punctuation, and other allowable programming elements, arranged in the proper sequence to perform an operation. Programmers call these statements *code*. Typically, you type your programming statements into a text editor, save them to a file, and then use a compiler to translate the statements into an executable program. An *executable program* is a file containing machine language instructions that can be directly executed by the computer.

Programming with App Inventor

One way that App Inventor makes programming easy to learn is by eliminating many of the errors that beginning students commonly make. With a traditional programming language, like Java or C++, beginners frequently make typing mistakes that result in misspelled keywords, missing punctuation characters, and other such errors. These types of mistakes are known as *syntax errors*. If a program contains even one syntax error, it cannot be translated into an executable program. As a result, students and professional programmers alike spend a lot of time tracking down syntax errors and fixing them. In App Inventor, however, syntax errors never happen, because you do not type programming statements.

Instead, you drag and drop *code blocks*, which are graphical building blocks, into an editor. The blocks, which represent actions and data, can be “snapped” together, like the pieces of a puzzle, to create fully functional programming statements. Because you don’t have to spend time locating and fixing syntax errors, you can concentrate on planning the actions that you want your app to perform, and arranging them into the proper sequence.

Perhaps the greatest reason that programming is easy with App Inventor is that it’s fun! Rather than writing boring programs that perform calculations or analyze data, you will be creating mobile apps that you can run on your own smartphone or tablet, assuming it is a supported Android device. So if you have a great idea for an app, you can create it and install it on your device. If you want to share your apps with others, you can upload them to the Google Play Store or the App Inventor Gallery. (For more information about submitting your App Inventor apps to the Google Play store and the App Inventor Gallery, see Appendix C.)

Checkpoint

1.1 What is a computer?

1.2 What is a program?

1.3 What is an algorithm?

1.4 What is the only language that computers understand?

1.5 Why were programming languages invented?

1.3 Introducing App Inventor

App Inventor is a Web application that runs in your browser. The following browsers work with App Inventor:

- Google Chrome 4.0 or higher
- Apple Safari 5.0 or higher
- Mozilla Firefox 3.6 or higher

Each time you work with App Inventor to create or modify an app, you will perform the following general steps:

- You will open your browser and go to the App Inventor website.
- You will either create a new project or open an existing project.
- You will open the Blocks Editor.
- You will connect either the Android emulator or an actual Android device to App Inventor.

In Tutorial 1-1, you will perform these steps, using the Android emulator. Before performing this tutorial, make sure you have set up App Inventor on your computer. (If you have not already set up App Inventor, see Appendix A for instructions.)

VideoNote
Starting App Inventor and Creating a New Project

Tutorial 1-1:
Starting App Inventor and Creating a New Project

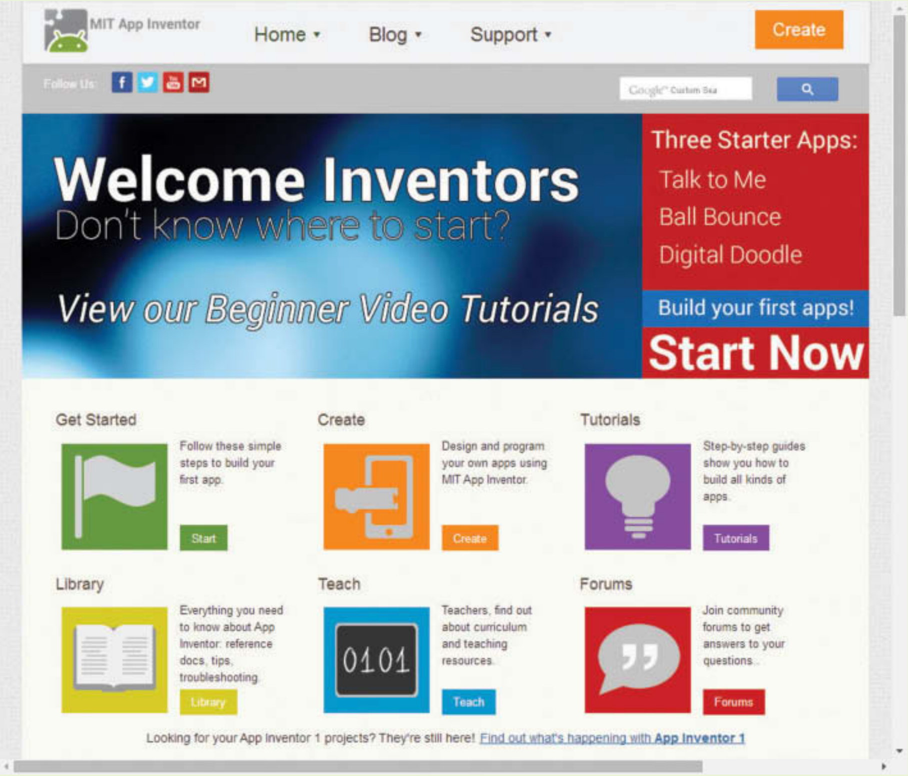
Step 1: Open your Web browser and go to the following address:

```
http://appinventor.mit.edu
```

Step 2: You will see a screen similar to the one shown in Figure 1-4. Click the *Create* button that appears in the upper right area of the screen.

Step 3: If you are not currently signed into your Google account, you will see a screen similar to the one shown in Figure 1-5. (If you are already signed into your Google account, skip to Step 4.) Enter your email address and Google account password, and then click *Sign In*.

Figure 1-4 App Inventor Main Screen (*Source:* MIT App Inventor 2)



MIT App Inventor

Home ▾ Blog ▾ Support ▾ Create

Follow Us:

Welcome Inventors

Don't know where to start?

View our Beginner Video Tutorials

Three Starter Apps:

Talk to Me

Ball Bounce

Digital Doodle

Build your first apps!

Start Now

Get Started

Follow these simple steps to build your first app.

Start

Create

Design and program your own apps using MIT App Inventor.

Create

Tutorials

Step-by-step guides show you how to build all kinds of apps.

Tutorials

Library

Everything you need to know about App Inventor: reference docs, tips, troubleshooting

Library

Teach

Teachers, find out about curriculum and teaching resources.

Teach

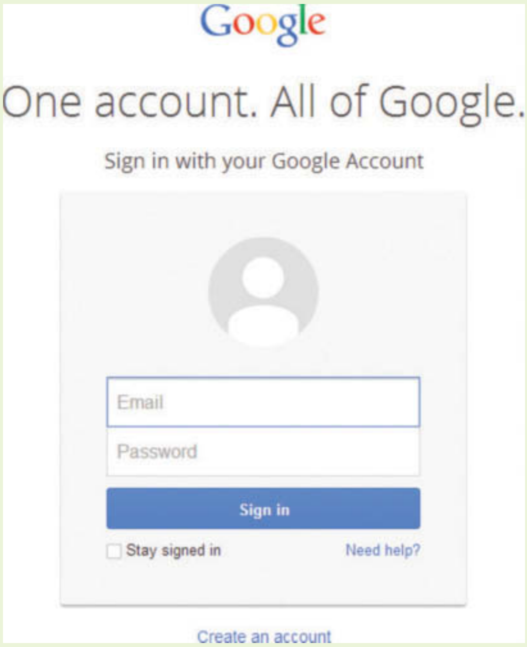
Forums

Join community forums to get answers to your questions.

Forums

Looking for your App Inventor 1 projects? They're still here! Find out what's happening with App Inventor 1

Figure 1-5 Login to Your Google Account (*Source:* Google and the Google logo are registered trademarks of Google Inc., used with permission.)



Google

One account. All of Google.

Sign in with your Google Account

Email

Password

Sign in

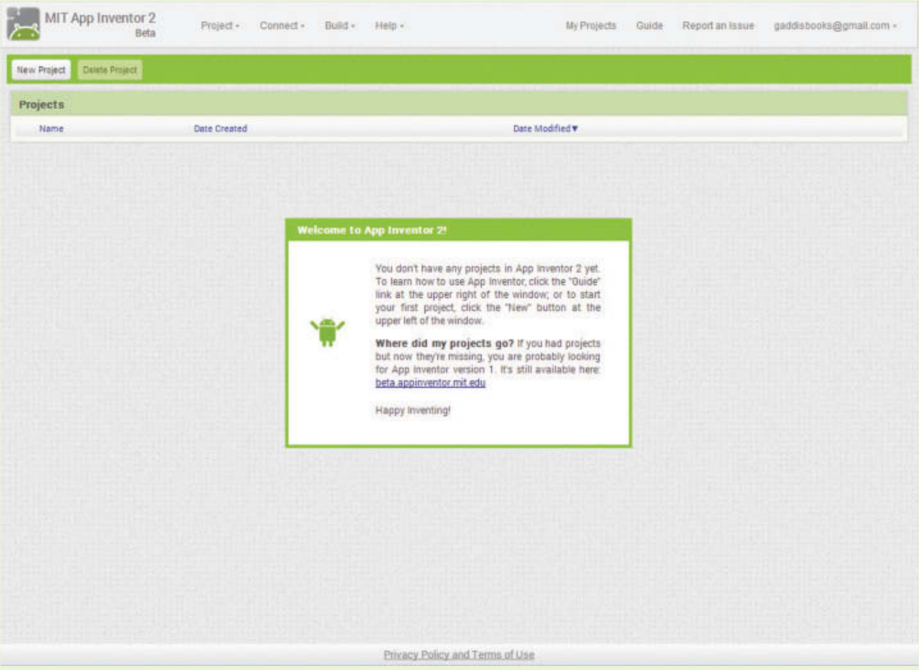
□ Stay signed in Need help?

Create an account

NOTE: If this is the first time you have used App Inventor with the Google account that you are signed in as, you will see a screen indicating that App Inventor is requesting permission to access your Google account. Click the *Allow* button.

Step 4: Next, you will see the *My Projects* screen, as shown in Figure 1-6. This screen normally displays a list of all the App Inventor projects that you have created. From this screen, you can open a project, delete a project, download and upload projects, and perform other actions. There are no projects listed in the screen shown in Figure 1-6 because we haven't created any yet. Any time that you want to display this screen, you simply click the *My Projects* link, as shown in Figure 1-7.

Figure 1-6 The *My Projects* Screen (*Source:* MIT App Inventor 2)

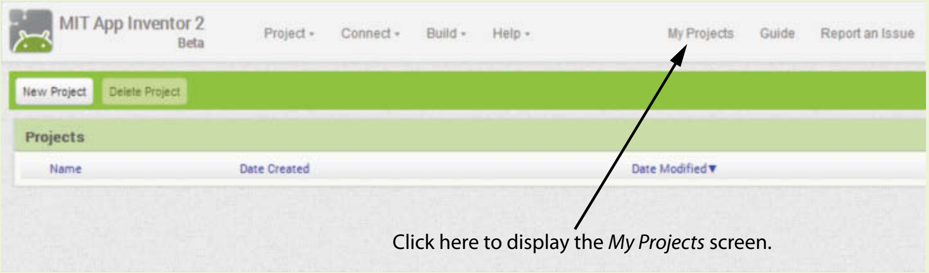


You don't have any projects in App Inventor 2 yet. To learn how to use App Inventor, click the "Guide" link at the upper right of the window; or to start your first project, click the "New" button at the upper left of the window.

Where did my projects go? If you had projects but now they're missing, you are probably looking for App Inventor version 1. It's still available here: beta.appinventor.mit.edu

Happy Inventing!

Figure 1-7 The *My Projects* Link (*Source:* MIT App Inventor 2)



Click here to display the *My Projects* screen.

Step 5: To create a new project, click the *New Project* button, as shown in Figure 1-8. This will display the dialog box shown in Figure 1-9, prompting you to enter the name of the project that you are creating. You must follow these rules when naming a project:

- The project name must begin with an alphabetical letter.
- After the first letter, the remaining characters can be alphabetical letters, numbers, or underscore characters (_).
- You cannot have spaces in a project name.

When you create a project, you should give it a name that describes it. Because this is your first project, enter *MyFirstProject* as the name, and then click the *OK* button.

Figure 1-8 Click the *New* Button to Start a New Project (*Source:* MIT App Inventor 2)

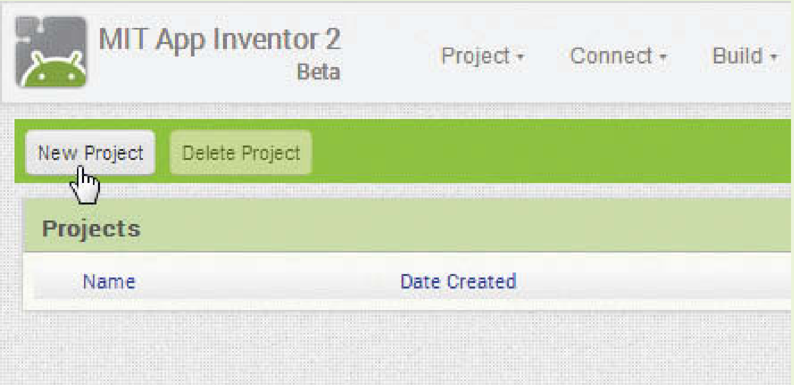
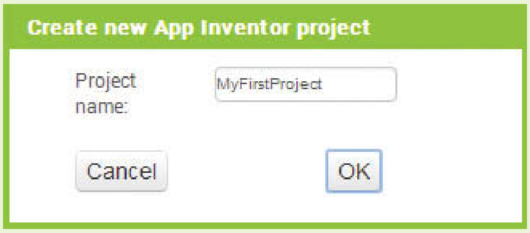


Figure 1-9 Specify a Project Name (*Source:* MIT App Inventor 2)

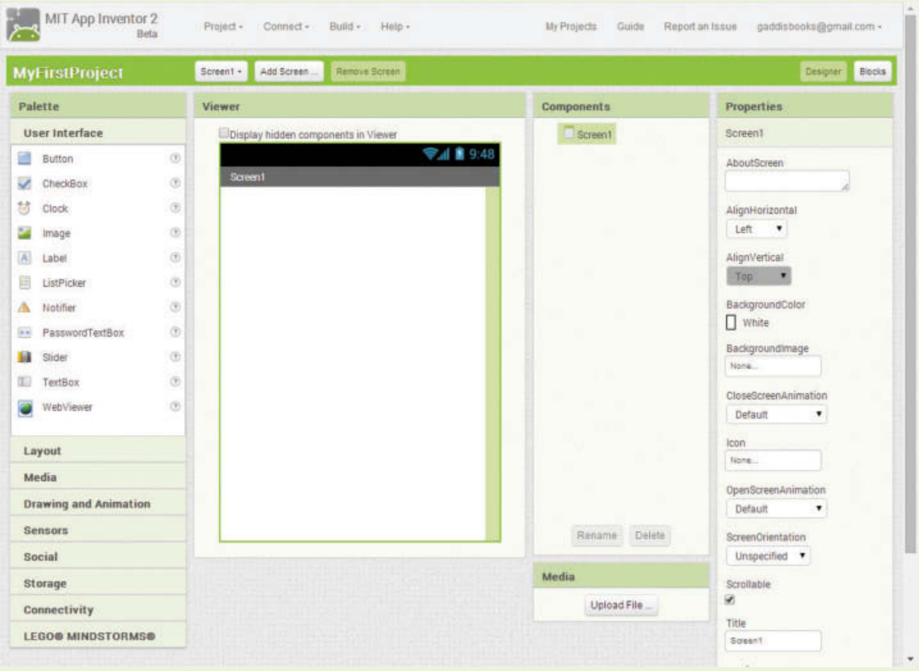

Create new App Inventor project

Project name: MyFirstProject

Step 6: You should now see the screen shown in Figure 1-10. This screen is known as the Designer. When you are developing an app, you will use the Designer to create the app's screen. We will discuss the Designer in greater detail later in the chapter.

Next you will open the Blocks Editor. Click the *Blocks* button in the upper-right area of the screen, as shown in Figure 1-11. The Blocks Editor will appear as shown in Figure 1-12.

Figure 1-10 The *Designer* (*Source:* MIT App Inventor 2)



AboutScreen

AlignHorizontal: Left ▾

AlignVertical: Top ▾

BackgroundColor: □ White

BackgroundImage: None...

CloseScreenAnimation: Default ▾

Icon: None...

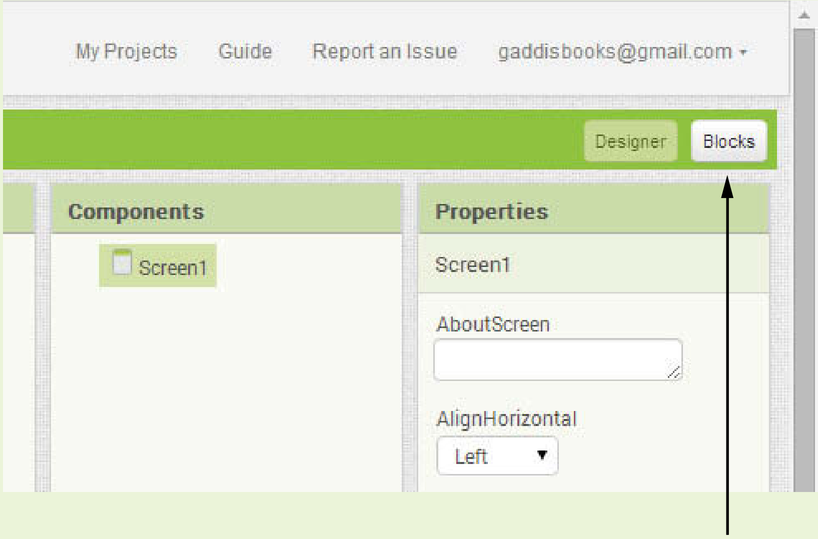
OpenScreenAnimation: Default ▾

ScreenOrientation: Unspecified ▾

Scrollable: ☑

Title: Screen1

Figure 1-11 Click the *Open the Blocks Editor* Button (*Source:* MIT App Inventor 2)



AboutScreen

AlignHorizontal: Left ▾

Click here to open the *Blocks editor.*

Step 7: The next step is to create a new Android emulator. As shown in Figure 1-13, click *Connect* at the top of the screen, and then click

Figure 1-12 The *Blocks Editor* (*Source:* MIT App Inventor 2)

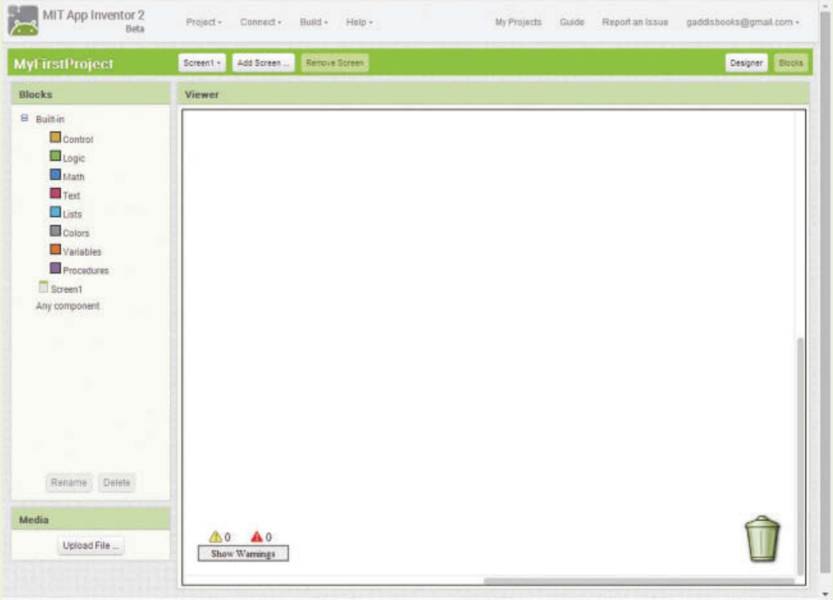
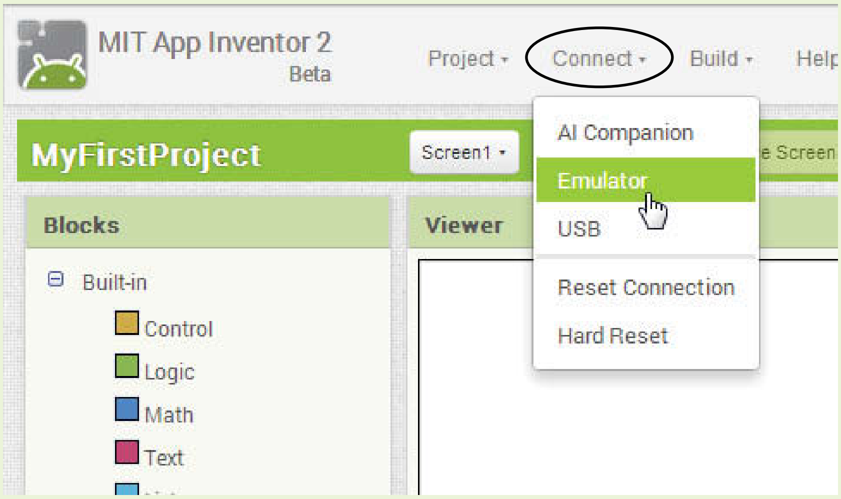


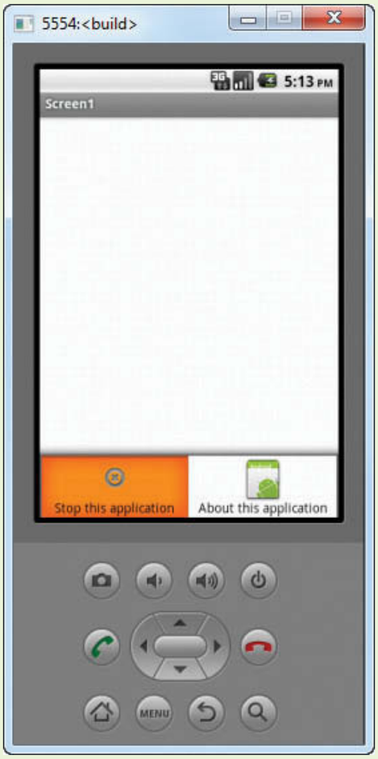
Figure 1-13 Click *Connect* and then Click *Emulator* to Create a New Android Emulator (*Source:* MIT App Inventor 2)



Emulator on the menu that appears. It might take several minutes for the emulator to be created in the computer's memory. Once the emulator has been created and initialized, it will appear as shown in Figure 1-14.

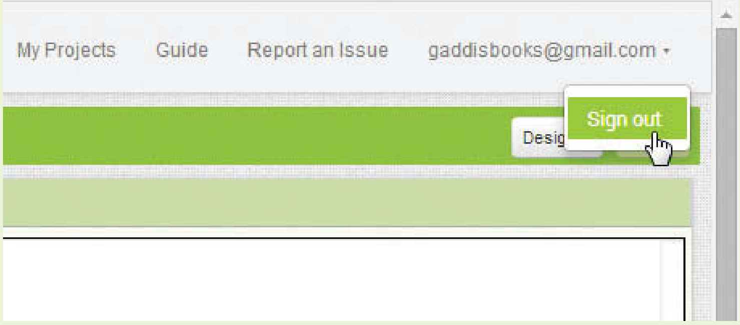
NOTE: In the Windows task bar, the emulator will be represented by an Android Icon ( ).

Figure 1-14 The Android Emulator (*Source:* Microsoft Corporation)



Step 8: If you plan to continue with the next tutorial at this time, leave App Inventor open in your browser and the emulator running. If you plan to continue with the next tutorial at a later time, close the emulator and sign out of App Inventor by clicking your account email, which appears in the upper-right corner of the window, and then clicking *Sign out*. This is shown in Figure 1-15.

Figure 1-15 Signing Out (*Source:* MIT App Inventor 2)



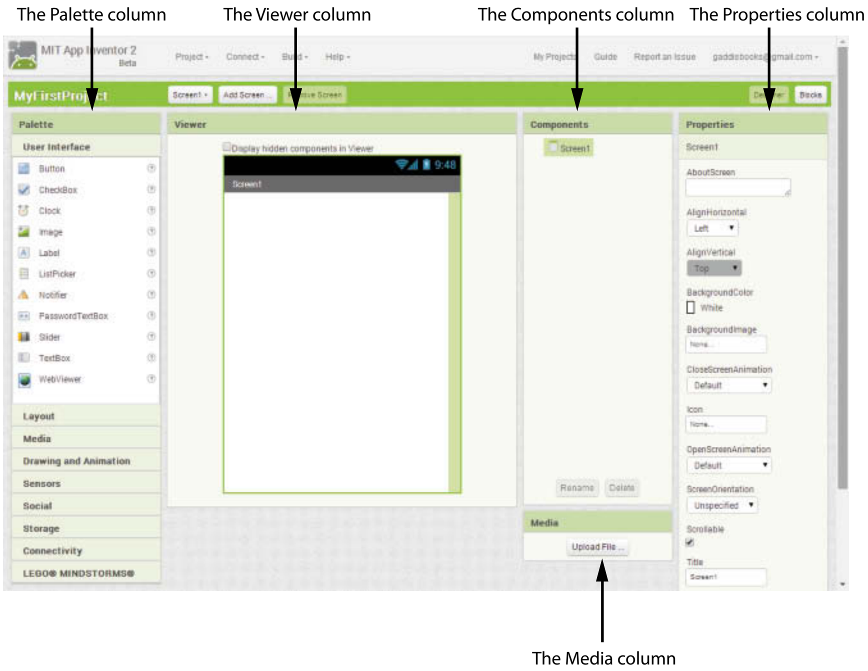
Let’s take a closer look at the various parts of App Inventor.

The Designer

When you create an app with App Inventor, you will use the Designer to create the app's screen. The Designer is organized into the following columns, which are identified in Figure 1-16:

- The Palette column
- The Viewer column
- The Components column
- The Media column
- The Properties column

Figure 1-16 The Designer (*Source:* MIT App Inventor 2)



Let's take a closer look at each of these columns.

The Palette Column

The leftmost column in the Designer is known as the Palette. The Palette provides a list of components that you can use to build your app. A *component* is an item that performs a specific purpose within an app. For example, an Image component displays an image on the screen, a Button component appears as a button that the user can touch, a Texting component sends and receives text messages, a PhoneCall component causes the phone to dial a number, and so forth. When you are creating an app, you select the components that you need from the Palette, and insert them into the app.

The Palette is divided into sections that each contain a group of components. Each section represents a category of components. The different sections, or categories, are:

User Interface—These are the fundamental components for building an app's screen. If you want the app to have a button that the user can click, an image that is displayed on the app's screen, a text box that the user can type input into, or various other basic components, you will find them here.

Layout—This section provides components for organizing other components on the app's screen. They provide ways to arrange components horizontally, vertically, or in rows and columns.

Media—This section provides components for taking photos, recording and playing videos, recording and playing sounds, picking images from the phone's gallery, recognizing speech, and converting text to speech.

Drawing and Animation—This section provides components for creating simple drawings and animations.

Sensors—These components allow your app to access the device's accelerometer (to detect shaking and movement), location sensor (to detect the device's location via GPS and/or network data), and orientation sensor (to detect the device's orientation, or the manner in which it is tilted). There is also a barcode scanner component and a near field communication sensor that allows two phones to exchange data.

Social—These are components that work with the phone's contact list, make phone calls, send and receive text messages, and perform certain operations with Twitter.

Storage—These are components that store data locally on the device or remotely on a Web server.

Connectivity—This section provides components for launching external applications, connecting with Bluetooth devices, and browsing the Web.

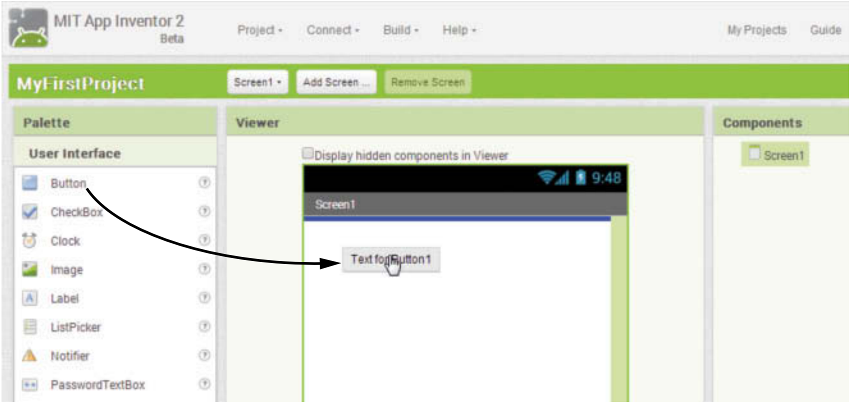
LEGO® MINDSTORM®—These specialized components are used to connect an app with a LEGO® MINDSTORM® NXT robot using Bluetooth.

You open a section in the Palette column simply by clicking its name. In Figure 1-16, you can see the *User Interface* section is open.

The Viewer Column

The Viewer column appears next to the Palette column. The Viewer column shows a rectangular area that represents the app's screen. You design an app's *user interface* (the part of the app that the user sees, and interacts with) by dragging components from the Palette and dropping them onto the simulated screen in the Viewer. Figure 1-17 shows a Button component being created by dragging it from the Palette to the Viewer. You can arrange the components on the simulated screen to make the app's interface look the way you want it.

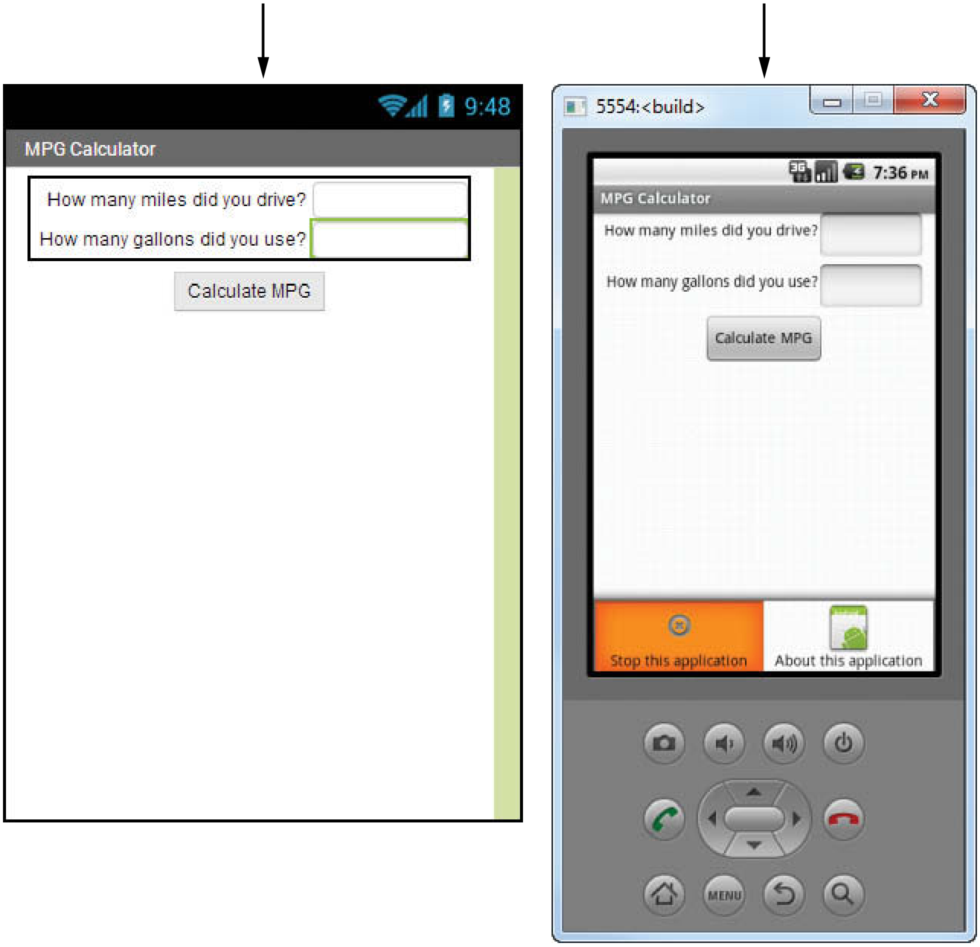
Figure 1-17 Creating a Component by Dragging it from the Palette to the Viewer (*Source:* MIT App Inventor 2)



NOTE: A subtle, but important concept to keep in mind is that the icons that are shown in the Palette are the *types* of components that you can create. When you drag a component from the Palette, you are selecting the type of component that you want. When you drop it into the Viewer, an actual component of the selected type is created.

Keep in mind, however, that the Viewer column does not truly show a *WYSIWYG* (*What You See Is What You Get*) display. The components that you place on the simulated screen in the Viewer might appear slightly different on the emulator screen, or on the device that you have connected to your system. You will be aware of any differences quickly because the components that you drop onto the simulated Viewer screen appear immediately on the emulator or the connected device. For example, Figure 1-18 shows an app screen in the Viewer, and the same screen displayed in the emulator. Notice that the shapes of the text boxes (the rectangles that let the user enter data) and the button are slightly different between the two screens, and the spacing between the components is also different. Think of the Viewer as a tool for arranging components on an app's screen, but always compare your layout with the actual display on the emulator or your connected device.

Figure 1-18 A Screen in the Viewer and the Emulator (*Source:* MIT App Inventor 2)



Screen design in the viewer.

Actual output in the emulator.

The Components Column

The Components column shows a hierarchical tree listing all of the components that you have placed in your app. Each time you drag a component from the Palette and drop it onto the Viewer, an entry representing that component appears in the Component column. You can use the Component column to select any component in your app.

The Media Column

Just below the Components column is the Media column. The Media column allows you to manage the media files (images, videos, and audio files) that you want to use in your app. Because App Inventor stores your apps in the cloud, you have to upload any media files that you want to use in an app. The Media column allows you to upload such files to the App Inventor server, download them from the server to your computer, and delete them from the server when they are no longer needed.

The Properties Column

A component's appearance and other characteristics, are determined by the component's properties. Here are just a few examples:

- If you want to display text on your device's screen, you will use a Label component. The Label component has a property named Text. You set the Label component's Text property to the text that you want to display.
- If you want to display an image on your device's screen, you will use an Image component. The Image component has a property named Picture that determines the image that is displayed. You set the Picture property to the name of the image file that you want displayed.
- If you want an app to play a sound, you will use a Sound component. The Sound component has a property named Source that determines the audio file that is played. You set the Source property to the name of the audio file that you want to play.

Once you have added a component to an app, you use the Properties column to examine and change the component's properties.

The Blocks Editor

The Blocks Editor appears in its own window, separate from the Designer. The Blocks Editor is where you assemble code blocks that perform actions. A code block, or simply a *block*, is a shape that looks something like a puzzle piece. Figure 1-19 shows an example. App Inventor provides numerous blocks that represent actions and data. The blocks are shaped in such a way that you can snap them together to make a program. For example, Figure 1-20 shows several blocks snapped together to make a complete programming statement. (Don't worry about understanding the blocks shown in Figures 1-19 and 1-20. They are just meant to show you examples of how blocks appear.)

Figure 1-19 A Code Block (*Source:* MIT App Inventor 2)

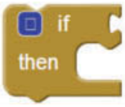
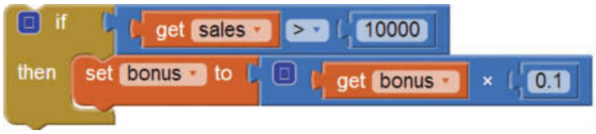


Figure 1-20 A Programming Statement Constructed from Code Blocks
(*Source:* MIT App Inventor 2)



```
if   get sales > 10000
then set bonus to   get bonus × 0.1
```

The Blocks Editor is shown in Figure 1-21. Notice that a workspace is provided for assembling blocks. You drag blocks onto the workspace, and snap them together to create programming statements.

The column on the left side of the Blocks Editor provides access to the blocks that you can use. Notice in Figure 1-22 that the Blocks column is organized in the following manner: Built-In, Screen1, and Any component. Each of these provides a separate set of blocks that you can use in your app. Here is a summary of each:

Built-In—The blocks that you find here are the basic blocks that make up the App Inventor language. You have the built-in blocks available to you in every app.

Screen1—Each time you add a component to Screen1 in the Designer, a set of component blocks are added to this section. Component blocks are blocks that perform an action on a specific component that you have added to the app.

Any component—This section contains advanced blocks that allow us to work with any component in the app.

Figure 1-21 The Blocks Editor (*Source:* MIT App Inventor 2)

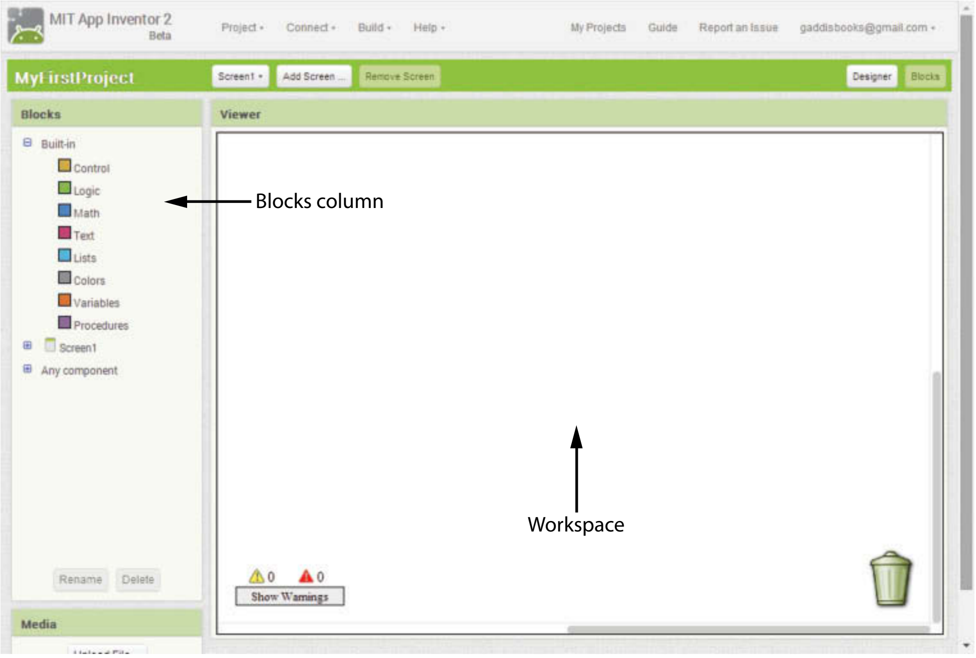
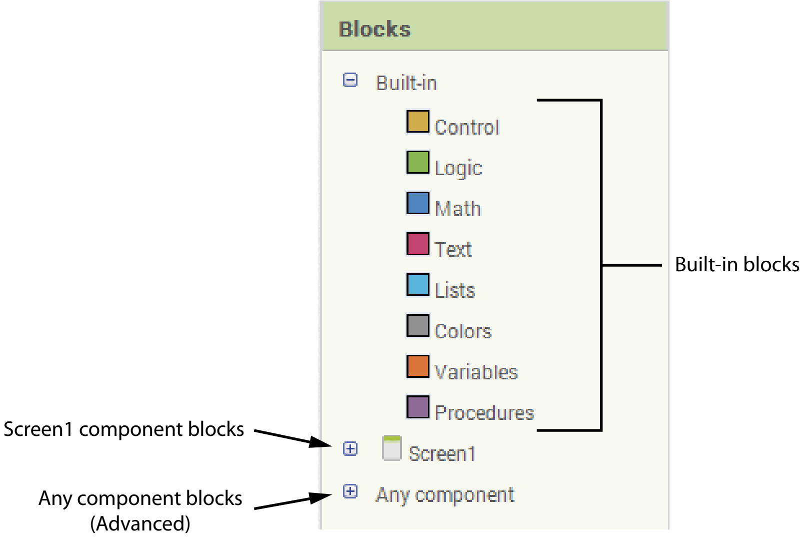


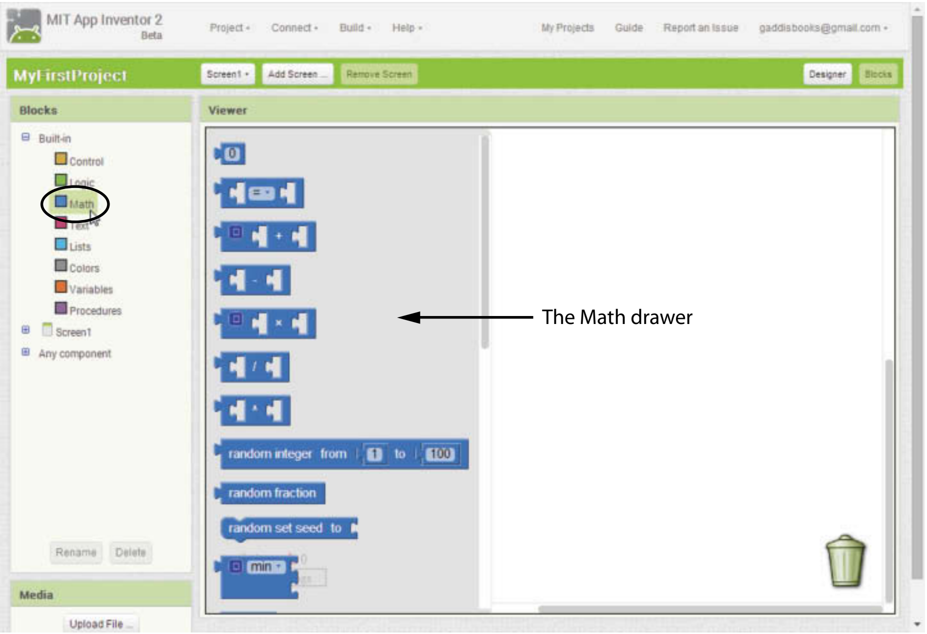
Figure 1-22 The Blocks Column (*Source:* MIT App Inventor 2)



The Built-In Blocks

Notice in Figure 1-22 that the Built-in Blocks section is organized into the following categories: *Control*, *Logic*, *Math*, *Text*, *Lists*, *Colors*, *Variables*, and *Procedures*. When you click one of the categories, a *drawer* containing blocks opens. For example, Figure 1-23 shows what happens when you click *Math*. A drawer containing various math blocks opens. When you open a drawer, you can click and drag a block onto the workspace.

Figure 1-23 The Math Drawer Opened (*Source:* MIT App Inventor 2)



The topmost area of the App Inventor screen is shown in Figure 1-24. The bar at the top shows the following items:

Project—When you click this item, a menu appears. The Project menu allows you to start, save, import, and export projects.

Connect—When you click this item, a menu appears. The Connect menu allows you to connect to an Android device or the Android emulator.

Build—When you click this item, a menu appears. The Build menu allows you to package an app so it can be shared with others.

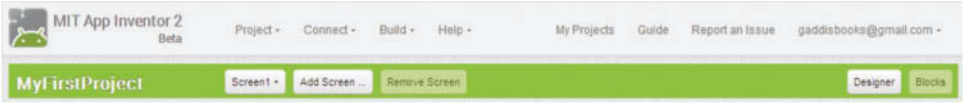
Help—When you click this item, a menu appears. The Help menu provides access to documentation, tutorials, and the App Inventor forum.

My Projects—When you click this item, the My Projects screen is displayed. (The screen was previously shown in Figure 1-6.) The My Projects screen displays a list of all the App Inventor projects that you have created. From this screen, you can open a project, delete a project, download and upload projects, and perform other actions.

Guide—Clicking this item opens a separate Web page containing the App Inventor documentation.

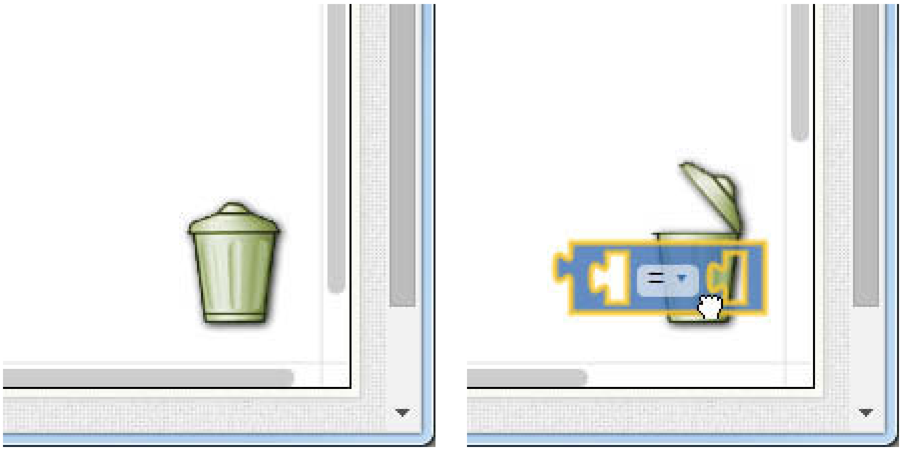
Report an Issue—Clicking this item takes you to the App Inventor support forum.

Figure 1-24 Top Part of the App Inventor Screen (*Source:* MIT App Inventor 2)



A trash can icon appears in the lower-right corner of the Blocks Editor, as shown on the left in Figure 1-25. You can delete blocks that you no longer need by dragging them onto the trash can, as shown on the right in Figure 1-25.

Figure 1-25 The Trash Can Icon (*Source:* MIT App Inventor 2)



TIP: You can also delete a block by selecting it and then pressing the Delete key on the keyboard.

Checkpoint

1.6 True or false: *My First Project* is a legal project name in App Inventor.

1.7 What part of App Inventor do you use to create an app's screen?

1.8 What is a user interface?

1.9 Does the Viewer show a *WYSIWYG* (*What You See Is What You Get*) representation of an app's screen?

1.10 What is the Palette column?

1.11 What is the Viewer column?

1.12 What is the Components column?

1.13 What is the Media column?

1.14 What is the Properties column?

1.15 What is the Blocks Editor?

1.16 What is a code block (or simply, a block)?

1.17 How do you create an emulator and connect to it?

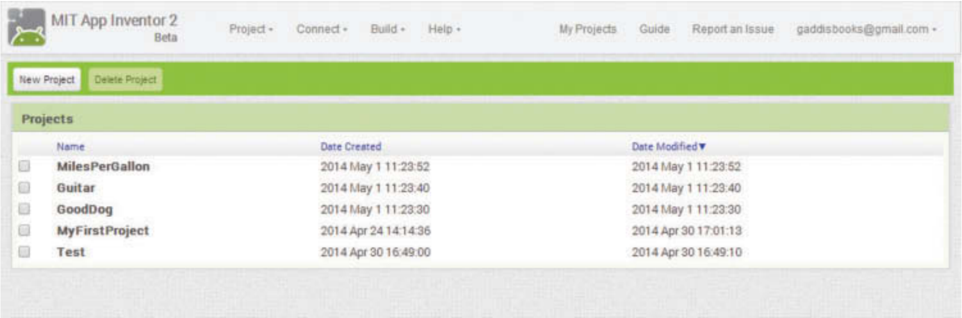
1.4 Getting Hands-On with App Inventor

You are almost ready to create your first app with App Inventor. There are a few more fundamental concepts and procedures that we need to discuss, however. We will cover those, and then in Tutorial 1-2 and Tutorial 1-3, you will create the Hello World app.

Managing Projects

You manage all of your App Inventor projects from the My Projects screen, which is shown in Figure 1-26. When you go to `appinventor.mit.edu` and click the *Create* button, you will be taken to the My Projects screen, unless you were actively working on a project the last time you used App Inventor. If that is the case, you will be

Figure 1-26 The My Projects Screen (*Source:* MIT App Inventor 2)



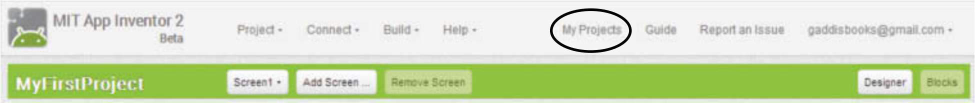
Projects

	Name	Date Created	Date Modified▼
□	MilesPerGallon	2014 May 1 11:23:52	2014 May 1 11:23:52
□	Guitar	2014 May 1 11:23:40	2014 May 1 11:23:40
□	GoodDog	2014 May 1 11:23:30	2014 May 1 11:23:30
□	MyFirstProject	2014 Apr 24 14:14:36	2014 Apr 30 17:01:13
□	Test	2014 Apr 30 16:49:00	2014 Apr 30 16:49:10

taken directly to your most recent project in the Designer. From the Designer or the Blocks Editor, you can click the *My Projects* link at the top of the screen, as shown in Figure 1-27, to go to the My Projects screen.

Figure 1-27 The My Projects Link at the Top of the App Inventor Screen

(*Source:* MIT App Inventor 2)



Notice that near the top of the My Projects screen (shown in Figure 1-26) there are buttons labeled *New Project* and *Delete Project*. The *New Project* button creates a new project. You used this in Tutorial 1-1, when you created the project named MyFirstProject. The *Delete Project* button deletes the project or projects that are currently selected in the project list. (You select a project by checking the checkbox that appears next to its name in the project list.)

Below these buttons, you see a list of your projects. The list shows each project's name and the date and time that it was created. To open a project, you simply click its name, and it is opened in the Designer. If you want to select a project (so you can delete it, or download its source), you click the checkbox that appears to the left of the project's name.

The App's `Screen1` Component

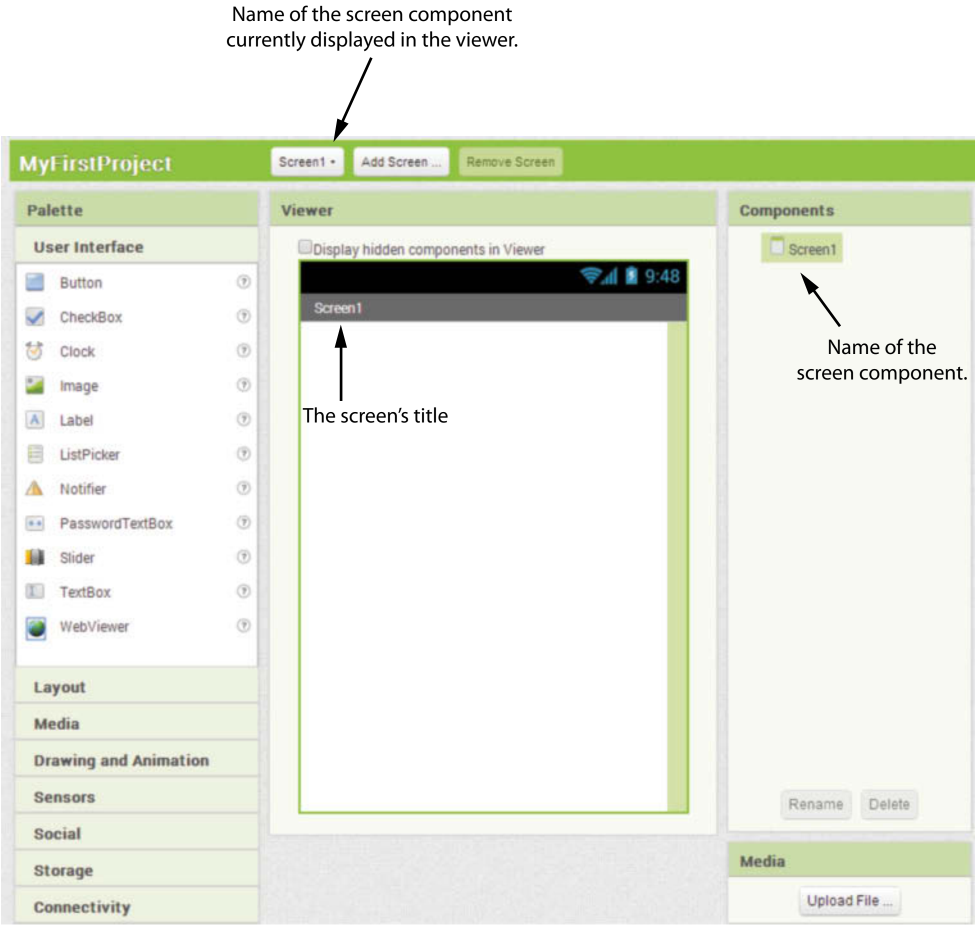
In App Inventor, the most fundamental type of component that an app can have is a Screen. In fact, every app *must* have a Screen component, which acts as a container for all the other components making up the app's user interface. When you start a new project, App Inventor automatically creates an empty Screen component.

Each component in an app must have a unique name that identifies it. When a component is added to an app, App Inventor automatically gives the component a default name. The empty Screen component that is automatically created in an app is named `Screen1`. Figure 1-28 shows the `Screen1` component, as displayed in the viewer.

Each time you add a component to an app, the component's name appears in the Component column. You can see in Figure 1-28 that `Screen1` is listed in the Component column. If you need to work with a particular component, you can select its name in the Component column.

NOTE: The Component column allows you to rename components. Normally, you will want to change the default name that App Inventor gives a component, because the default name does not indicate the component's purpose. The only exception is the `Screen1` component. App Inventor does not allow you to change the name of the `Screen1` component.

Figure 1-28 An App's Screen in the Viewer (*Source:* MIT App Inventor 2)



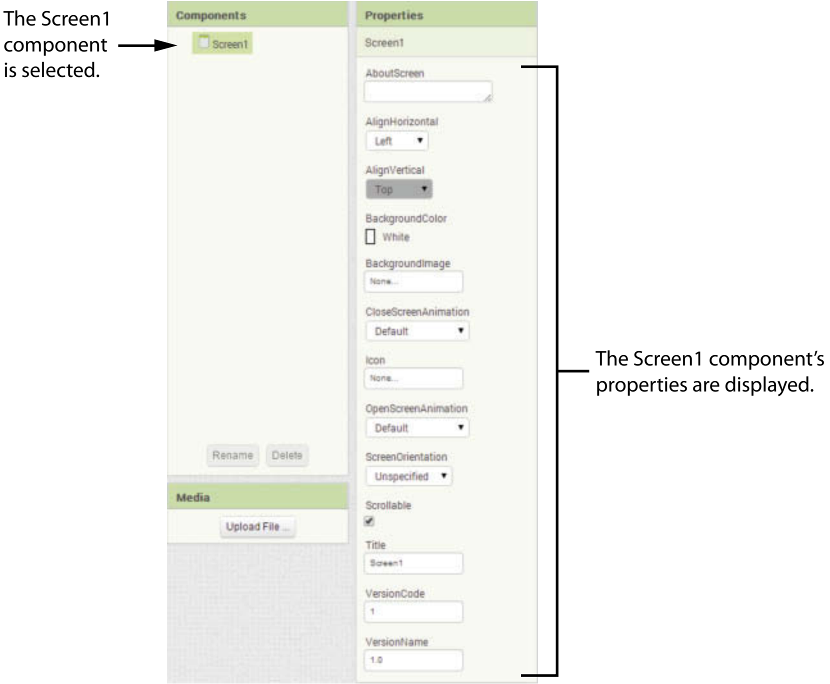
Working with the Properties Column

The appearance and other characteristics of a component are determined by the component's properties. When you select a component (either by clicking the component in the Viewer, or clicking its name in the Components column), that component's properties are displayed in the Properties column. For example, when the `Screen1` component is selected, its properties are displayed in the Properties column as shown in Figure 1-29.

For example, look at the Properties column in Figure 1-29 and notice that one of `Screen1`'s properties is named Title. The Title property determines the text that is displayed in the screen's title bar (the bar that appears at the top of the screen). As you can see from the figure, the default value of this property is *Screen1*. The text that is entered for the `Screen1` component's Title property is displayed in the screen's title bar, both in the Viewer, and in the emulator or other connected device. This is shown in Figure 1-30.

In most cases, you will want to change the value of the `Screen1` component's Title property to something that makes more sense to the user. For example, Figure 1-31 shows the Viewer, the Properties Column, and the emulator after we have changed the Title property to *My First App*. (The `Screen1` component has several other properties, and a summary of all of them appears at the end of this chapter.)

Figure 1-29 The Properties Column, Showing the Selected Component's Properties (*Source:* MIT App Inventor 2)



AboutScreen

AlignHorizontal: Left

AlignVertical: Top

BackgroundColor: □ White

BackgroundImage: None...

CloseScreenAnimation: Default

Icon: None...

OpenScreenAnimation: Default

ScreenOrientation: Unspecified

Scrollable: ☑

Title: Screen1

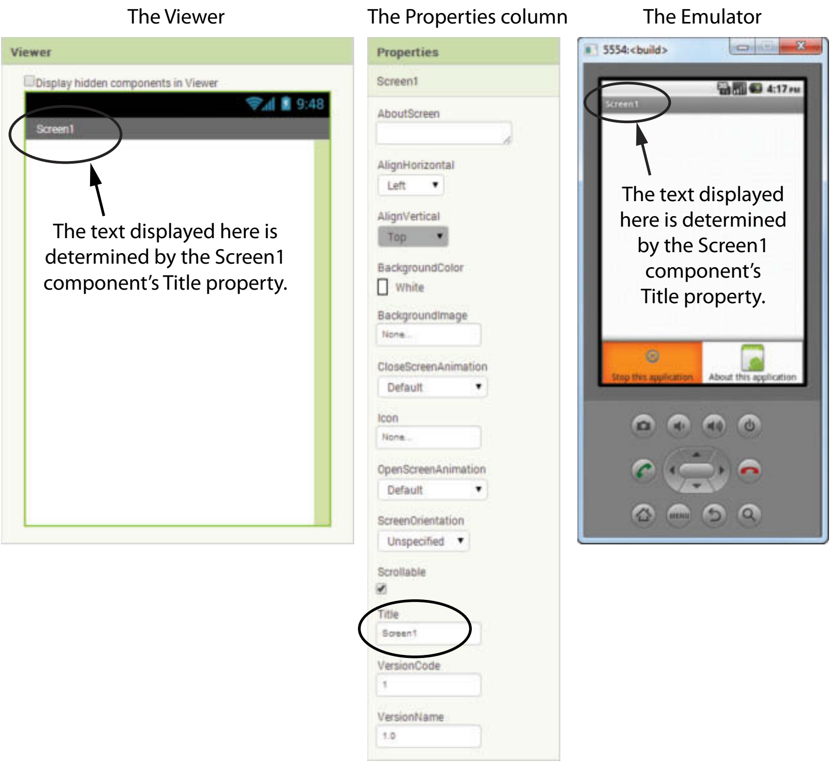
VersionCode: 1

VersionName: 1.0

The Screen1 component is selected.

The Screen1 component's properties are displayed.

Figure 1-30 The `Screen1` Component's Title Property Set to the Text *Screen1* (*Source:* MIT App Inventor 2)



AboutScreen

AlignHorizontal: Left

AlignVertical: Top

BackgroundColor: □ White

BackgroundImage: None...

CloseScreenAnimation: Default

Icon: None...

OpenScreenAnimation: Default

ScreenOrientation: Unspecified

Scrollable: ☑

Title: Screen1

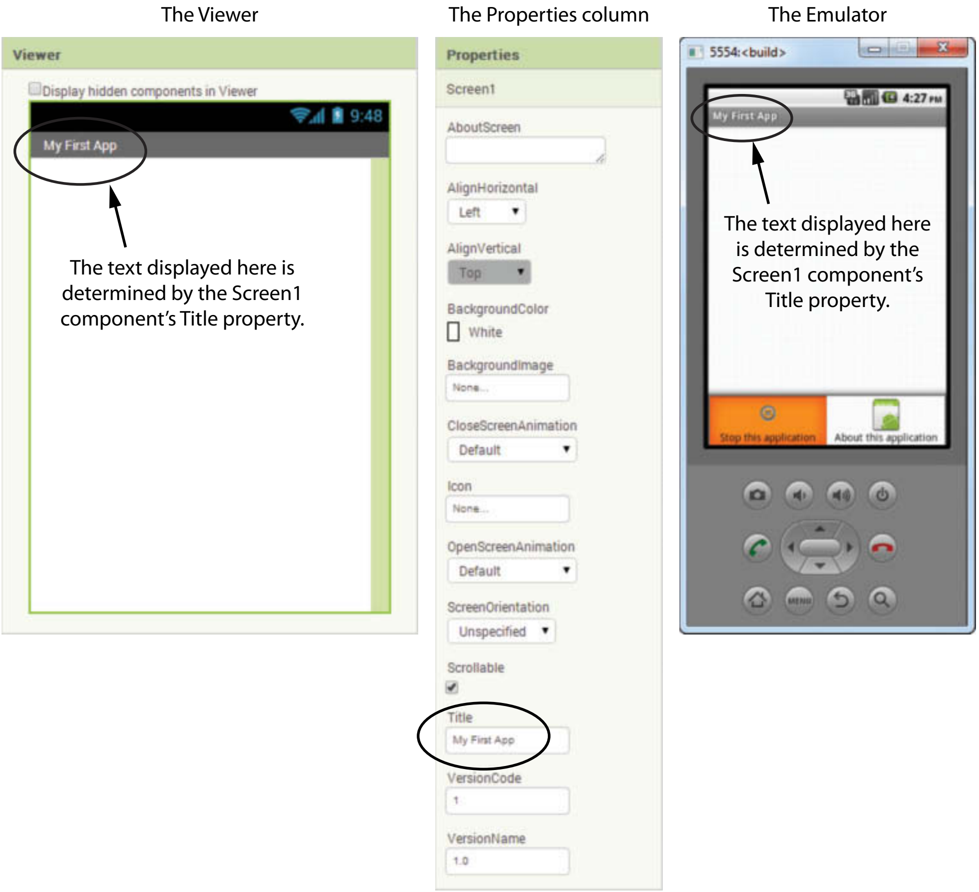
VersionCode: 1

VersionName: 1.0

The text displayed here is determined by the Screen1 component's Title property.

The text displayed here is determined by the Screen1 component's Title property.

Figure 1-31 The `Screen1` Component's Title Property Set to the Text *My First App* (*Source:* MIT App Inventor 2)



The text displayed here is determined by the Screen1 component's Title property.

The text displayed here is determined by the Screen1 component's Title property.

Properties

Screen1

AboutScreen

AlignHorizontal
Left

AlignVertical
Top

BackgroundColor
□ White

BackgroundImage
None...

CloseScreenAnimation
Default

Icon
None...

OpenScreenAnimation
Default

ScreenOrientation
Unspecified

Scrollable
☑

Title
My First App

VersionCode
1

VersionName
1.0

Label Components

Another basic component is the Label. A Label component displays text on the app's screen. You create a Label component by dragging it from the User Interface section of the Palette and onto the app's screen in the Viewer, as shown in Figure 1-32. When you create Label components in an app, they are given default names such as `Label1`, `Label2`, and so forth. For example, Figure 1-33 shows the Components column after a Label component has been created in an app. As you can see in the figure, the name of the component is `Label1`. (Also, notice that the name `Label1` is highlighted in the Components column, and in the Viewer, the component is outlined with a green border. This indicates that the component is currently selected.)

Once you have created a Label component, you set its Text property to the text that you want the component to display. For example, in Figure 1-34, the `Label1` component's Text property (look in the Properties column) is set to the value *Text for Label1*. As a result, *Text for Label1* is displayed on the app's screen in the viewer and on the emulator or other connected device. To change the text that is displayed

Figure 1-32 Creating a Label Component (*Source:* MIT App Inventor 2)

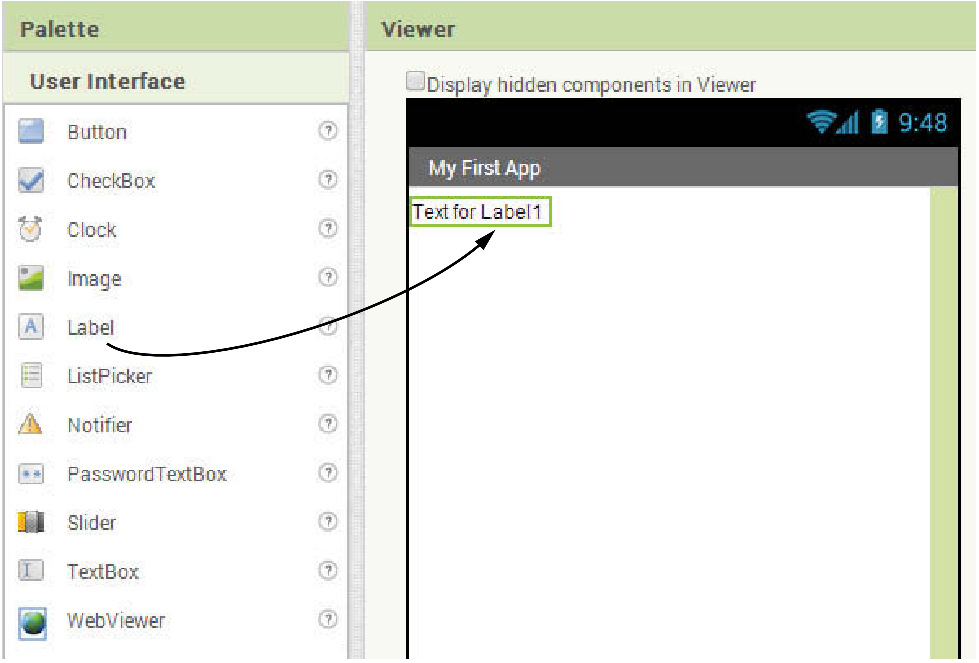
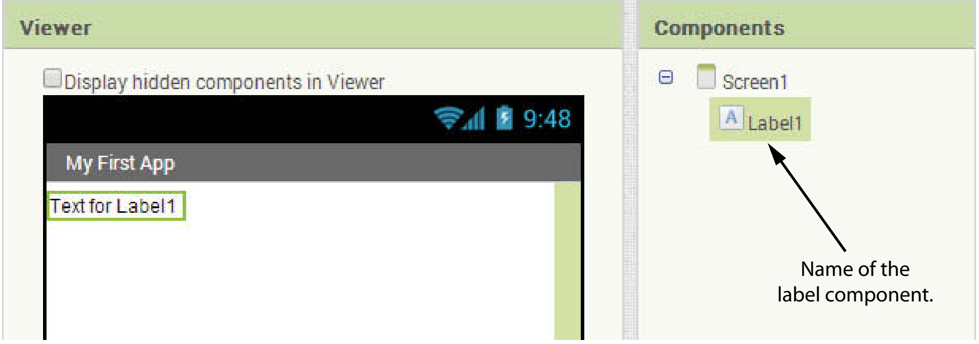


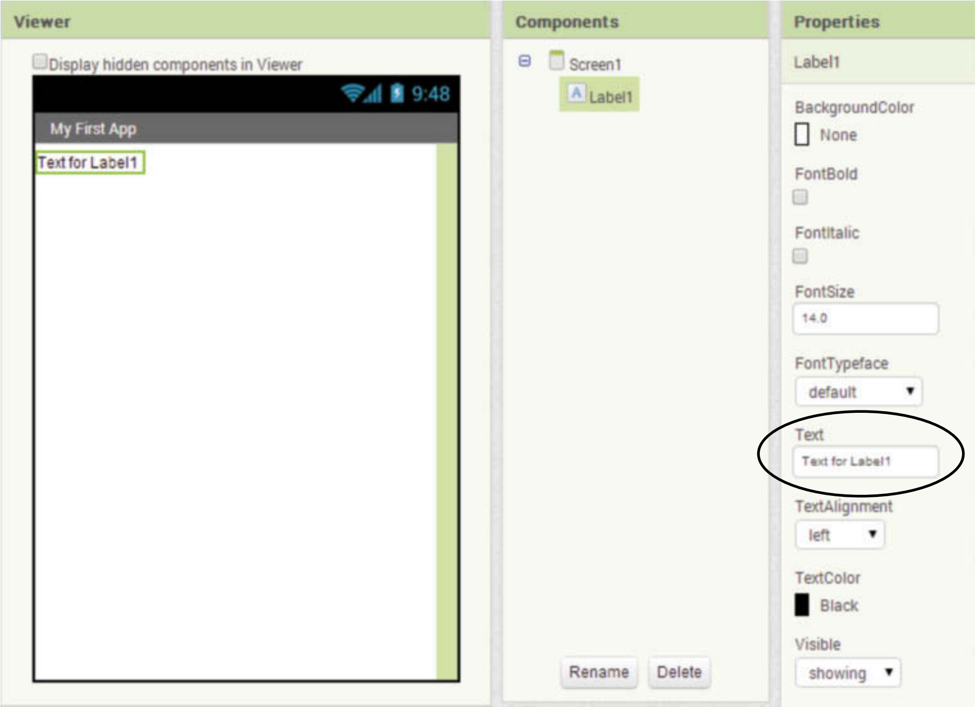
Figure 1-33 The Name of the Component Shown in the Components Column (*Source:* MIT App Inventor 2)


Viewer
☐ Display hidden components in Viewer
9:48
My First App
Text for Label1
Components
Screen1
Label1
Name of the label component.

by a Label component, just make sure the component is selected in the Components tree, and then change the value of the component's Text property in the Properties column. (If the component is not currently selected, simply click its name in the Components column to select it.)

For example, Figure 1-35 shows an app with a Label component, with its Text property set to *Apps are fun to create!* The text is displayed by the component in the Viewer and on the emulator.

Figure 1-34 A Label Component's Text Property Determines the Text that the Component Displays (*Source:* MIT App Inventor 2)



Label1

BackgroundColor
□ None

FontBold
□

FontItalic
□

FontSize
14.0

FontTypeface
default ▾

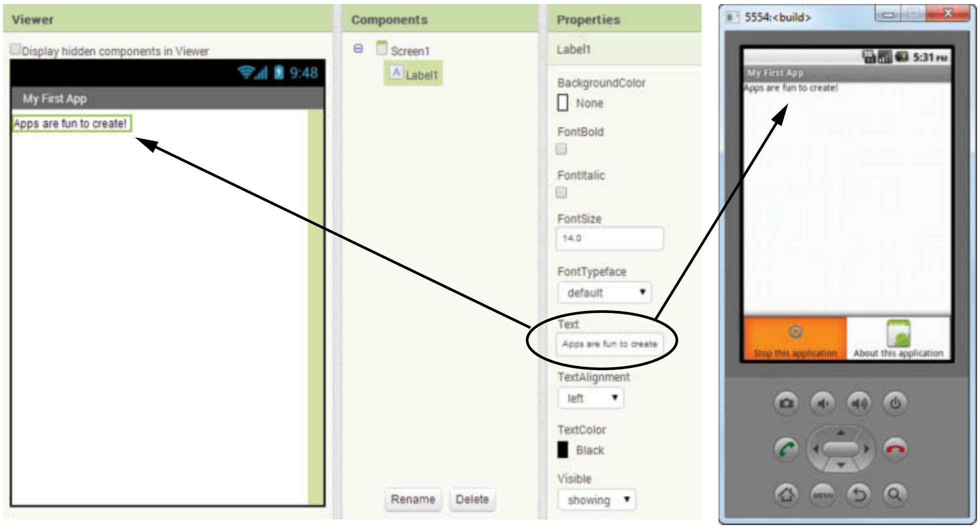
Text
Text for Label1

TextAlignment
left ▾

TextColor
■ Black

Visible
showing ▾

Figure 1-35 A Label Component Displaying the Text *Apps are fun to create!* (*Source:* MIT App Inventor 2)



Label1

BackgroundColor
□ None

FontBold
□

FontItalic
□

FontSize
14.0

FontTypeface
default ▾

Text
Apps are fun to create

TextAlignment
left ▾

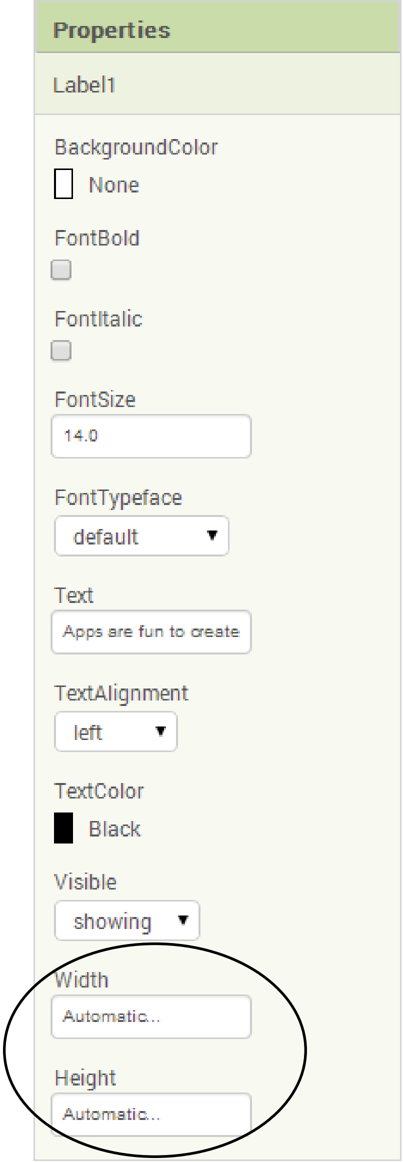
TextColor
■ Black

Visible
showing ▾

Label Width and Height

Label components have two properties, Width and Height, that determine the label's size on the app's screen. Figure 1-36 shows where a Label component's Width and Height properties are located in the Properties column. Notice that both properties are set to the value *Automatic* by default.

Figure 1-36 The Label Component's Width and Height Properties
(*Source:* MIT App Inventor 2)



Properties

Label1

BackgroundColor
□ None

FontBold
□

FontItalic
□

FontSize
14.0

FontTypeface
default ▼

Text
Apps are fun to create

TextAlignment
left ▼

TextColor
■ Black

Visible
showing ▼

Width
Automatic...

Height
Automatic...

When you click the Width or the Height properties, a small dialog box appears, as shown in Figure 1-37. (The dialog box is the same for both the Width and the Height properties.) The possible values that you can set the Width and Height properties to are:

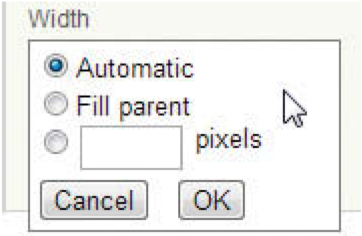
Automatic—When a component's Width property is set to *Automatic*, the component's width will automatically adjust to accommodate the size of the label's text.

When a component's Height property is set to *Fill Parent*, the label's height will automatically adjust to accommodate the size of the label's text.

Fill parent—When a component's Width property is set to *Fill Parent*, the component will be as wide as the container (such as the `Screen1` component) that it is enclosed in. When a component's Height property is set to *Fill Parent*, the component will be as high as the container (such as the `Screen1` component) that it is enclosed in.

A Specified Number of Pixels—You can specify a specific number of pixels for a component's width and/or height. You should avoid this in most cases, because different devices have different screen sizes. Specifying a specific number of pixels for a component's width or height will cause the component to appear differently on different devices.

Figure 1-37 Dialog Box to Set the Width Property (*Source:* MIT App Inventor 2)



Changing a Component's Name

A component's name identifies the component in blocks that make up the app's code, and in the App Inventor environment. When you create a component, App Inventor automatically gives it a name (we refer to this as the default name). For example, suppose you created three Label components in an app. App Inventor would name these components `Label1`, `Label2`, and `Label3`. Default names are not very descriptive, so you should always change a component's name to something that is more meaningful. A component's name should reflect the purpose of the component.

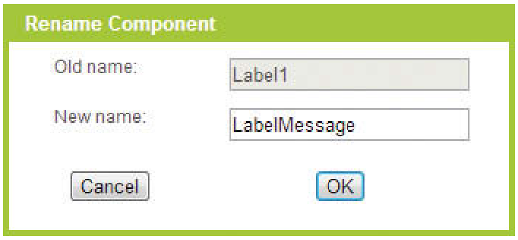
For example, suppose you are creating an app that has several Label components, and one of them is used to display a phone number. A default name such as `Label1` does not convey the component's purpose. A name such as `LabelPhoneNumber` would be much better. When you are working with the app's code blocks, and you see the name `LabelPhoneNumber`, you will know precisely which Label component the code block is referring to.

In the Designer, you can use the Components column to change the name of any component (except the `Screen1` component). Here are the steps:

1. Click the name of the component in the Components column to select it.
2. Click the *Rename* button at the bottom of the Components column.
3. The *Rename Component* dialog box shown in Figure 1-38 will appear. Enter the component's new name and click OK.

In Figure 1-38, we are changing the name of the `Label1` component to `LabelMessage`. Figure 1-39 shows the Components column after the component's name has been changed.

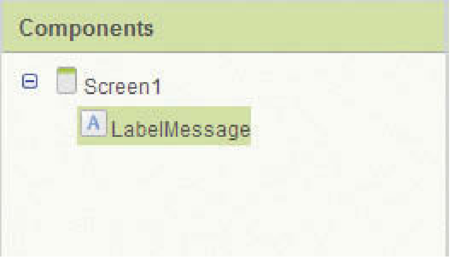
Figure 1-38 Rename Component Dialog Box (*Source:* MIT App Inventor 2)



Old name: Label1

New name: LabelMessage

Figure 1-39 The Component's Name is Changed to `LabelMessage` (*Source:* MIT App Inventor 2)



Rules and Conventions for Naming Components

When naming a component, you must follow these simple rules:

- Component names can contain only letters, numbers, and underscores (_).
- The first character of a component name must be a letter.
- Component names cannot contain spaces.

Table 1-1 lists some example component names and indicates whether each one is legal or illegal.

Table 1-1 Legal and illegal component names (*Source:* Pearson Education, Inc.)

Name	Legal or Illegal?
`3rdTestScoreLabel`	Illegal because component names must start with a letter
`Label*Mobile*Number`	Illegal because the * character is not allowed. Component names can contain only letters, numbers, and underscores.
`Label Contact Name`	Illegal because component names cannot contain spaces
`Label_Contact_Name`	Legal

Optional Conventions Used in this Book

Because a component's name should reflect the component's purpose, programmers often find themselves creating names that are made of multiple words. In this book, we always begin a component's name with a word that indicates the type of component. For example, a Label component's name will always begin with the word *Label*. This is not a requirement, but rather a convention that we follow in this book.

In addition, we use the Pascal naming convention, which makes names easier to read when they contain multiple words. For example, look at the following names, which are written in all lowercase letters:

```
labelcontactname
labeltotalpoints
labelmobilenumber
```

Unfortunately, these names are not easily read by the human eye because the words are not separated. Because we cannot have spaces in component names, we need to find another way to separate the words in a multiword name to make it more readable to the human eye. In this book, we address this problem using the Pascal case naming convention. In a Pascal case name, the first letter of each word is capitalized. Here are some examples:

```
LabelContactName
LabelTotalPoints
LabelMobileNumber
```

Deleting Components

If you add a component to an app and later decide that you don't want the component, it's easy to delete it. Just click on the component's name in the Components column to select it, and then click the *Delete* button that appears at the bottom of the Components column.

Button Components

Buttons are common components in mobile apps, as well as desktop applications. The user can click a button to make some action take place. In App Inventor, you create a Button component by dragging it from the User Interface section of the Palette to the app's screen in the Viewer. This is shown in Figure 1-40.

Button components have a Text property, which holds the text that is displayed on the face of the button. When you create a Button component, it is given a default name such as `Button1`, `Button2`, and so forth, and its Text property will be set to *Text for Button1*, *Text for Button2*, and so forth. Once you create a Button component, you should change its name to something that is more descriptive. You should also change the component's Text property to indicate what the button will do when it is clicked. For example, a button that calculates an average might have the text *Calculate Average* displayed on it. To change a Button component's Text property, just select the component in the Components tree, and then change the value of the component's Text property in the Properties column.

Figure 1-41 shows an example of an app with a Button component. Notice in the figure that we have renamed the component to `ButtonExample`, and we have changed its Text property to *Click Me!* The value of the Text property is displayed on the face of the button in both the viewer and the emulator.

Figure 1-40 Creating a Button Component (*Source:* MIT App Inventor 2)

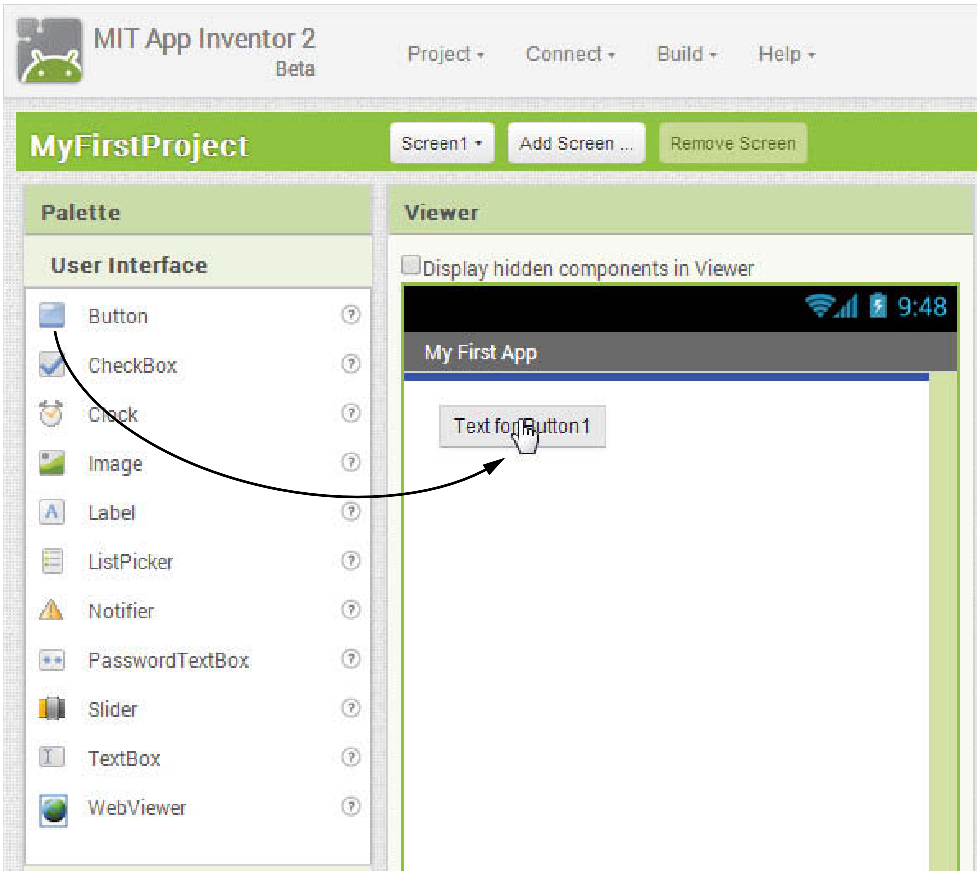
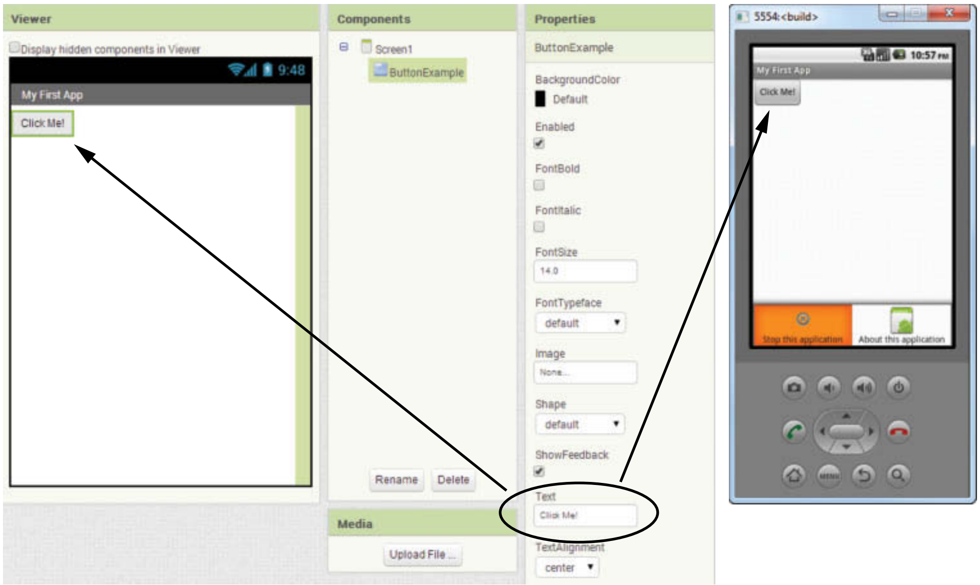


Figure 1-41 A Button Component Displaying the Text *Click Me!* (*Source:* MIT App Inventor 2)



ButtonExample

BackgroundColor: Default

Enabled: ☑

FontBold: □

FontItalic: □

FontSize: 14.0

FontTypeface: default

Image: None...

Shape: default

ShowFeedback: ☑

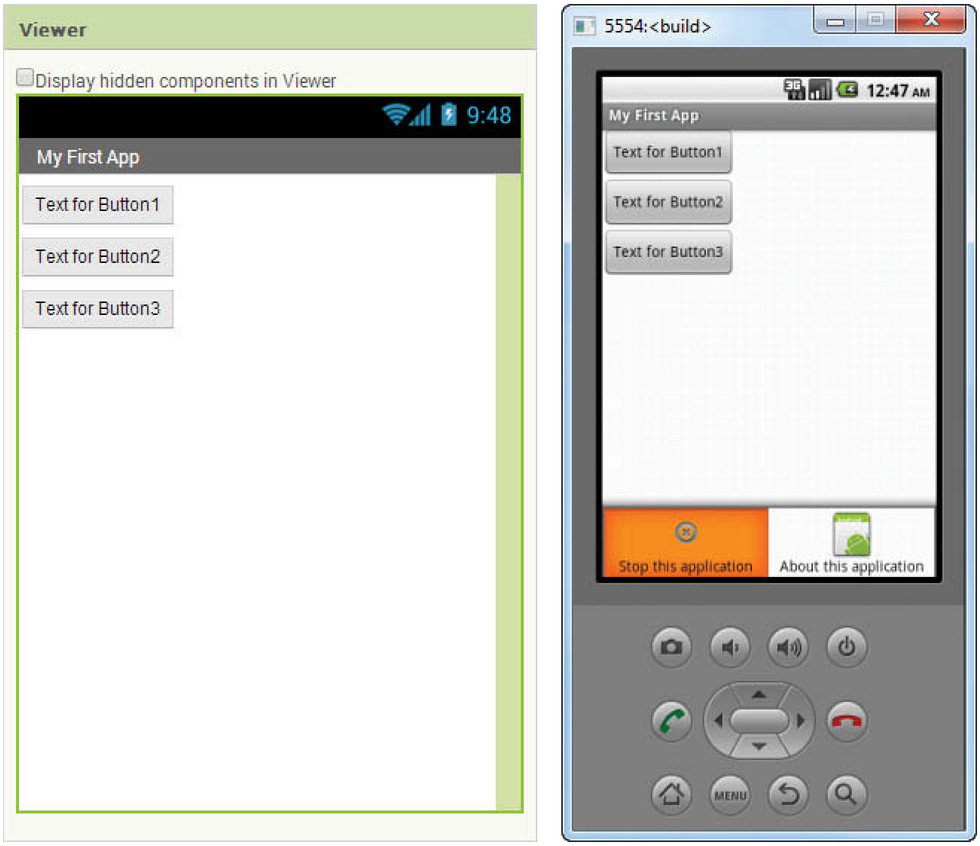
Text: Click Me!

TextAlignment: center

Screen Alignment

When you place components on an app's screen, the components are arranged vertically, from the top of the screen to the bottom of the screen. By default, they are also aligned along the left edge of the screen. For example, Figure 1-42 shows an app with three Button components. The image on the left shows the app's screen in the Viewer, and the image on the right shows the app in the emulator.

Figure 1-42 An App with Three Button Components (*Source:* MIT App Inventor 2)

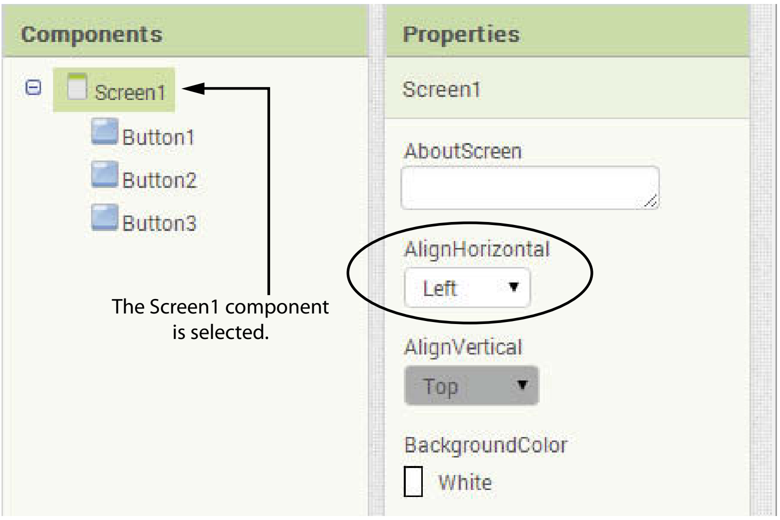


Screen components have an AlignHorizontal property (shown in Figure 1-43) that determines how the components that are contained in the screen are horizontally aligned. You can set the AlignHorizontal property to one of the following values:

- **Left**—Components are aligned along the left edge of the screen
- **Center**—Components are aligned in the center of the screen
- **Right**—Components are aligned along the right edge of the screen

Figure 1-44 shows examples of how each of these settings affect the contents of the screen. The default setting for the AlignHorizontal property is Left.

Figure 1-43 The AlignHorizontal Property (*Source:* MIT App Inventor 2)



AboutScreen

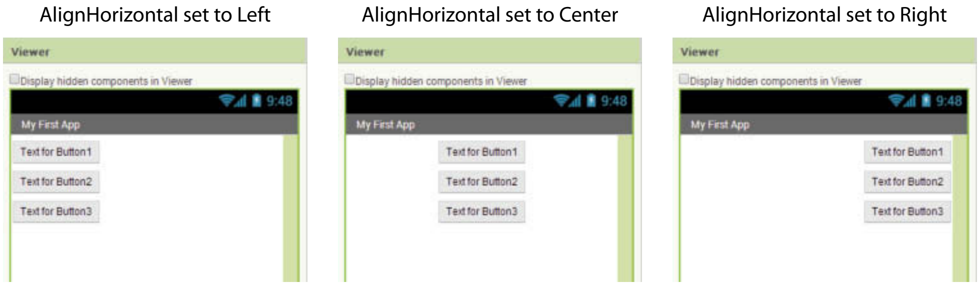
AlignHorizontal: Left

AlignVertical: Top

BackgroundColor: White

The Screen1 component is selected.

Figure 1-44 Examples of the AlignHorizontal Property Settings (*Source:* MIT App Inventor 2)


AlignHorizontal set to Left

AlignHorizontal set to Center

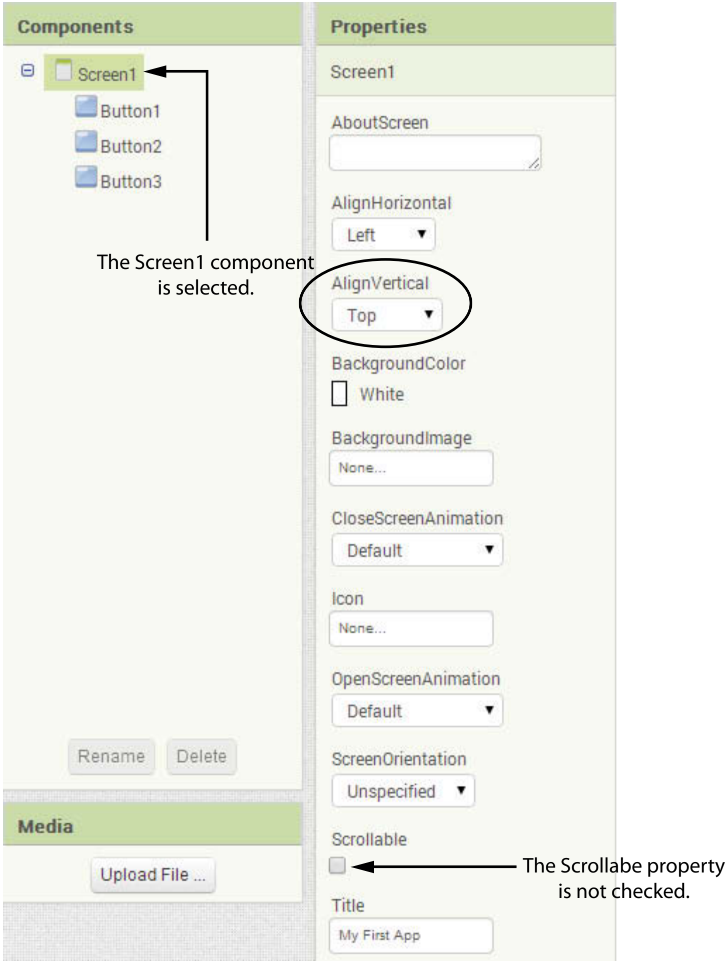
AlignHorizontal set to Right

Screen components also have an AlignVertical property (shown in Figure 1-45) that determines how the components that are contained in the screen are vertically aligned. You can change the AlignVertical property only if the screen is not scrollable (the Scrollable property is not checked). If this is the case, you can set the AlignVertical property to one of the following values:

- **Top**—Components are aligned along the top of the screen
- **Center**—Components are aligned in the center of the screen
- **Bottom**—Components are aligned along the bottom of the screen

Figure 1-46 shows examples of how each of these settings affect the contents of the screen. (In each example, the AlignHorizontal property is set to *Center*.) The default setting for the AlignVertical property is *Top*. If the Scrollable property is checked, the components are automatically aligned to the top of the screen.

Figure 1-45 The AlignVertical Property (*Source:* MIT App Inventor 2)



Components

⊟ Screen1

Button1

Button2

Button3

The Screen1 component is selected.

Properties

Screen1

AboutScreen

AlignHorizontal: Left

AlignVertical: Top

BackgroundColor: □ White

BackgroundImage: None...

CloseScreenAnimation: Default

Icon: None...

OpenScreenAnimation: Default

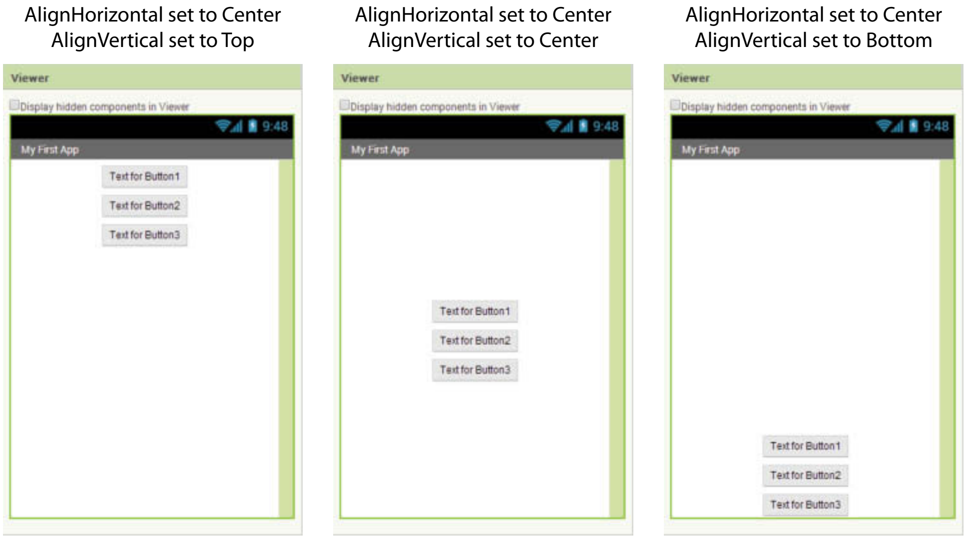
ScreenOrientation: Unspecified

Scrollable: □

The Scrollabe property is not checked.

Title: My First App

Figure 1-46 Examples of the AlignVertical Property Settings (*Source:* MIT App Inventor 2)



AlignHorizontal set to Center
AlignVertical set to Top

AlignHorizontal set to Center
AlignVertical set to Center

AlignHorizontal set to Center
AlignVertical set to Bottom

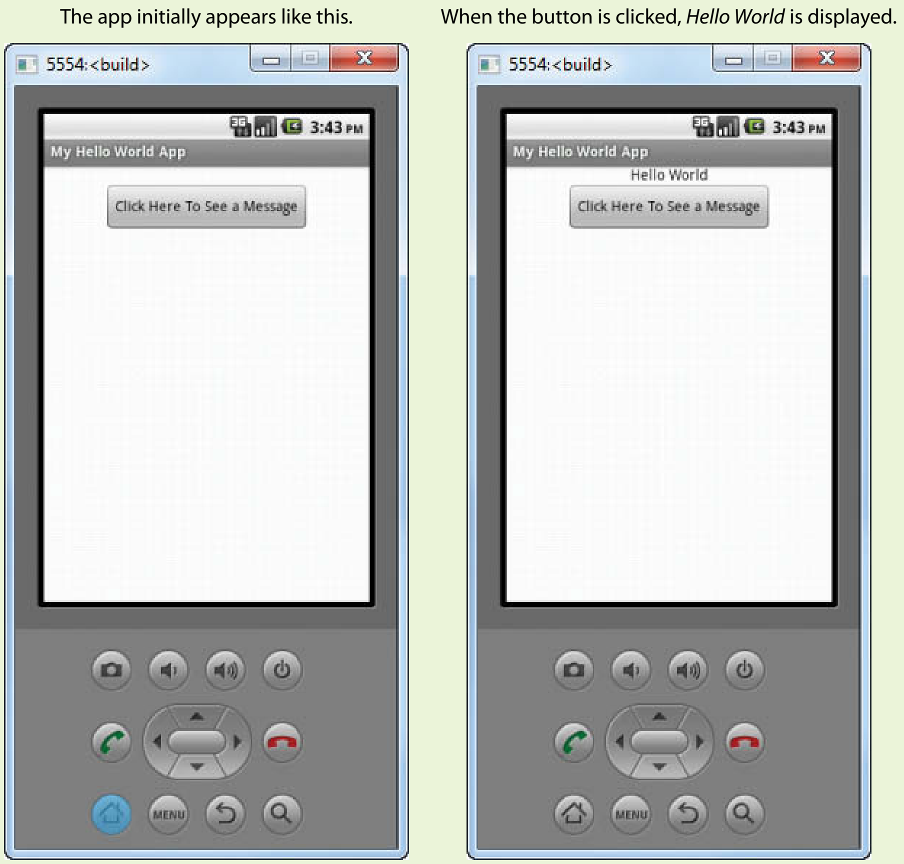
At this point, you know enough to design the screen for your first app. Tutorial 1-2 leads you through the steps to create the screen for the Hello World app.

Tutorial 1-2:
Creating the Screen for the Hello World App

VideoNote
Creating the Screen for the Hello World App

When a student is learning computer programming, it is traditional to start by learning to write a *Hello World* program. A *Hello World* program is a simple program that merely displays the words *"Hello World"* on the screen. In this tutorial and the next, you will use App Inventor to create a *Hello World* app. The app will initially appear as the image on the left in Figure 1-47. Notice that

Figure 1-47 The Completed App (*Source:* MIT App Inventor 2)



The app initially appears like this.

When the button is clicked, *Hello World* is displayed.

the screen contains a button that reads *Click Here To See a Message*. When you click the button, the message *Hello World* will appear, as shown in the image on the right in the figure.

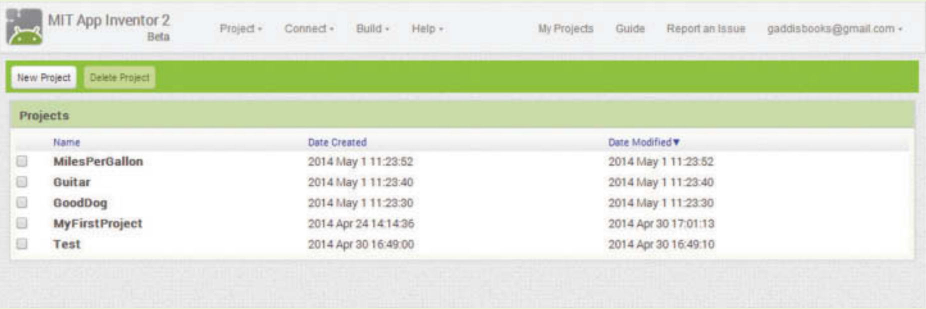
The process of creating this app is divided into two parts. In this tutorial you will create the app's screen. In the next tutorial you will use the Blocks Editor to write code that displays the *Hello World* message to appear when the user clicks the button.

Step 1: If App Inventor is already running on your computer, go to the *My Projects* page, which will appear similar to Figure 1-48. (Your list of projects will be different.)

If App Inventor is not running on your computer:

- Go to `appinventor.mit.edu` with your browser.
- Click the *Create* link that appears on that page.
- If prompted, log into your Google account.
- Go to the My Projects page, which will appear similar to Figure 1-48. (Your list of projects will be different.)

Figure 1-48 The My Projects Page (*Source:* MIT App Inventor 2)

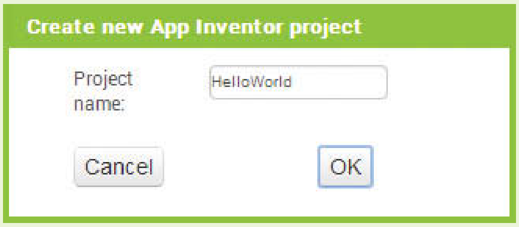


Projects

	Name	Date Created	Date Modified▼
□	MilesPerGallon	2014 May 1 11:23:52	2014 May 1 11:23:52
□	Guitar	2014 May 1 11:23:40	2014 May 1 11:23:40
□	GoodDog	2014 May 1 11:23:30	2014 May 1 11:23:30
□	MyFirstProject	2014 Apr 24 14:14:36	2014 Apr 30 17:01:13
□	Test	2014 Apr 30 16:49:00	2014 Apr 30 16:49:10

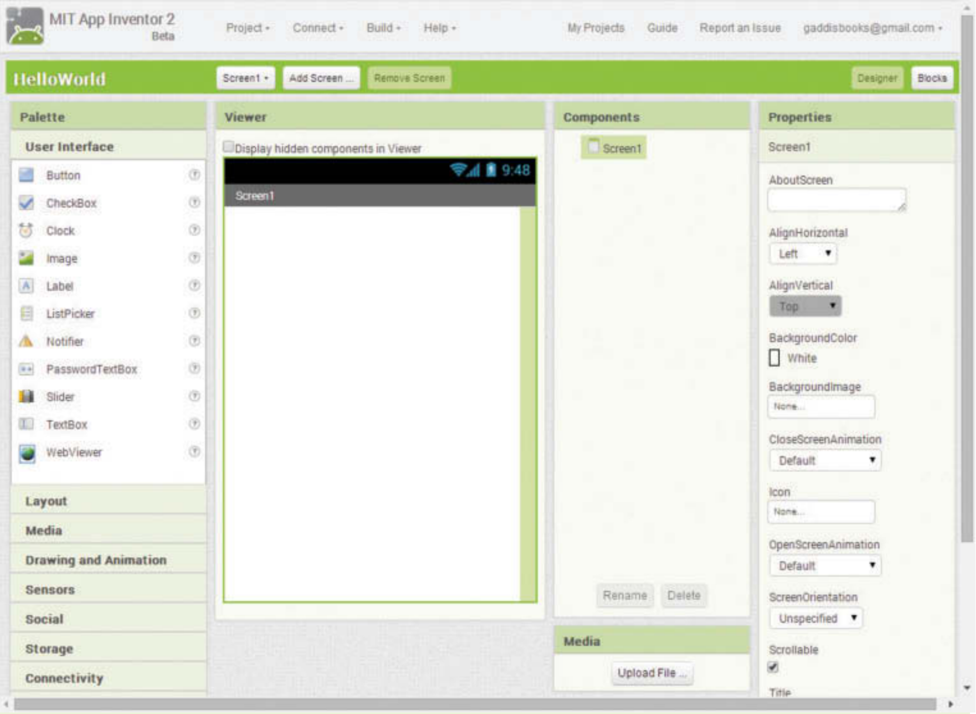
Step 2: Click the *New Project* button that appears above the list of projects. In the dialog box that appears, enter *HelloWorld* as the project name, as shown in Figure 1-49, and click the *OK* button. The project will be created, and the Designer will appear, as shown in Figure 1-50.

Figure 1-49 Enter the Project Name (*Source:* MIT App Inventor 2)



Project name: HelloWorld

Figure 1-50 The HelloWorld Project in the Designer (*Source:* MIT App Inventor 2)



AboutScreen

AlignHorizontal
Left ▾

AlignVertical
Top ▾

BackgroundColor
□ White

BackgroundImage
None...

CloseScreenAnimation
Default ▾

Icon
None...

OpenScreenAnimation
Default ▾

ScreenOrientation
Unspecified ▾

Scrollable
☑

Title

Step 3: The `Screen1` component should already be selected in the Components column. In the Properties column, change the AlignHorizontal property to *Center*, and change the Title property to read *My Hello World App*. This is shown in Figure 1-51.

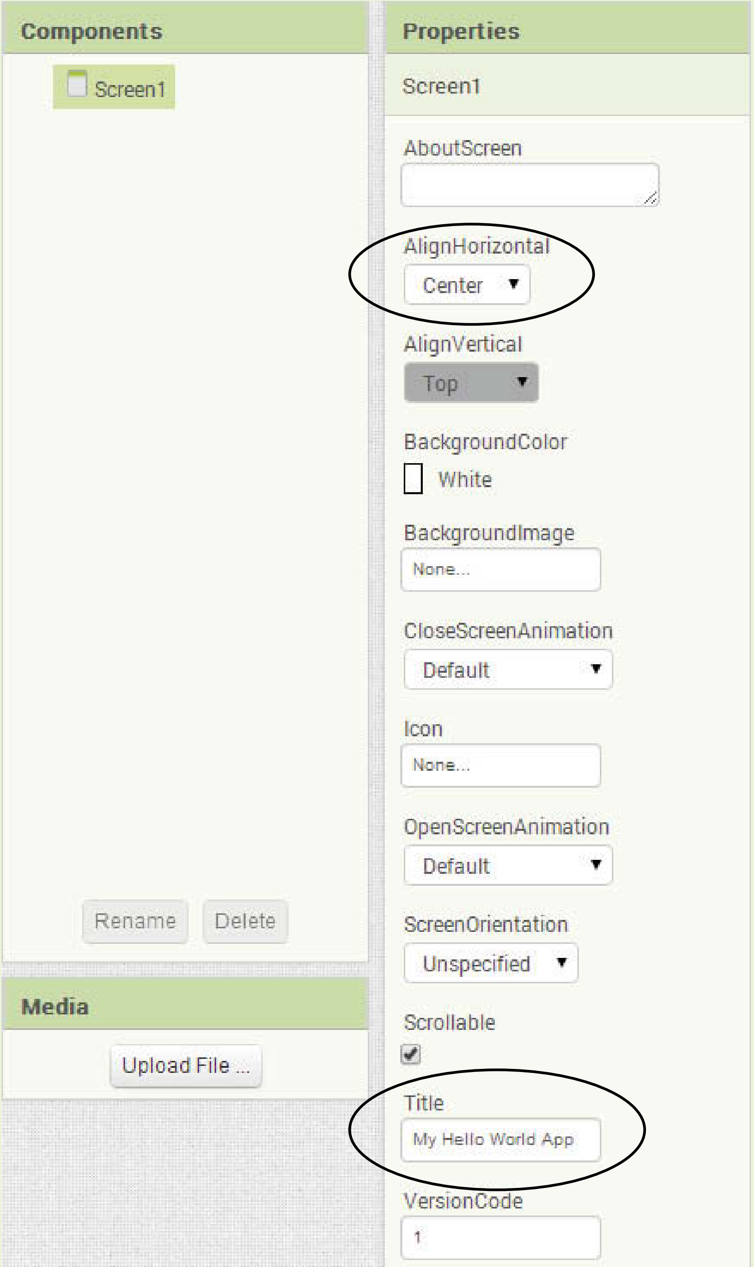
Step 4: Drag a Label component from the Palette to the Viewer, as shown in Figure 1-52. This creates a Label component named `Label1`, with its Text property set to *Text for Label1*.

Step 5: Because the name `Label1` is not very descriptive, you should change the component's name. Make sure the `Label1` component is selected in the Components column, and click the *Rename* button (which appears at the bottom of the Components column). The dialog box shown in Figure 1-53 will appear. Enter `LabelMessage` as the component's new name, and click *OK*. The component's new name should now appear in the Components column, as shown in Figure 1-54.

Step 6: Make sure the `LabelMessage` component is selected in the Components column, and in the Properties column, delete the contents of the Title property. (The Title property should appear empty.) This is shown

Figure 1-51 Setting the `Screen1` Component's Properties

(*Source:* MIT App Inventor 2)



Properties

Screen1

AboutScreen

AlignHorizontal: Center

AlignVertical: Top

BackgroundColor: □ White

BackgroundImage: None...

CloseScreenAnimation: Default

Icon: None...

OpenScreenAnimation: Default

ScreenOrientation: Unspecified

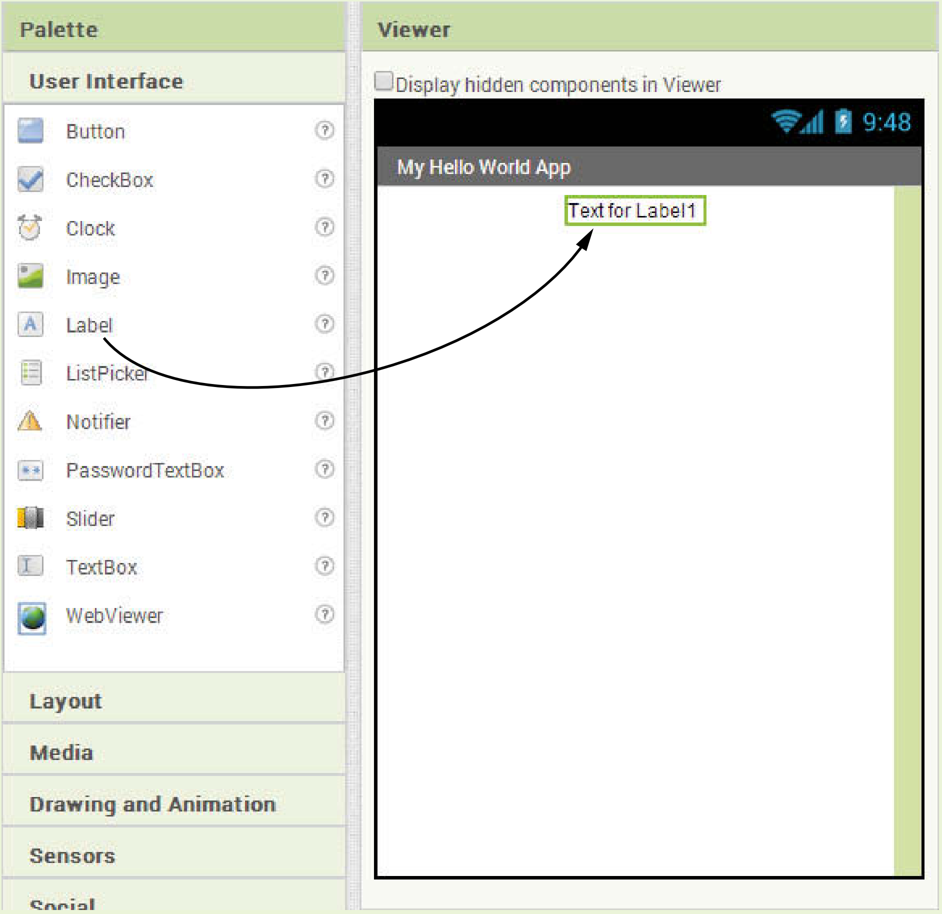
Scrollable: ☑

Title: My Hello World App

VersionCode: 1

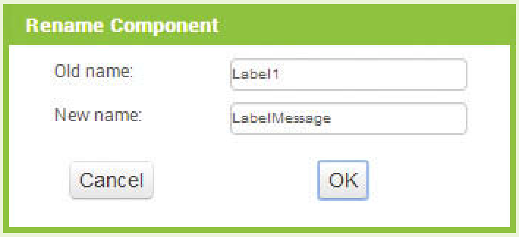
in Figure 1-55. Notice that the label now appears as a small dot in the viewer. This is because the label's Width and Height properties are both set to *Automatic*. Recall that this means the label's size will automatically

Figure 1-52 Creating the Label Component (*Source:* MIT App Inventor 2)



adjust to match the size of the text that it displays. Because the Text property is now empty, the label displays nothing, and its size automatically shrinks down to nothing. In fact, the only way that you can see the label in the Viewer is to select it in the Components column. The green border that indicates the component is selected will appear as a dot.

Figure 1-53 Renaming the `Label1` Component (*Source:* MIT App Inventor 2)


Rename Component

Old name: Label1

New name: LabelMessage

Figure 1-54 The Component Renamed (*Source:* MIT App Inventor 2)

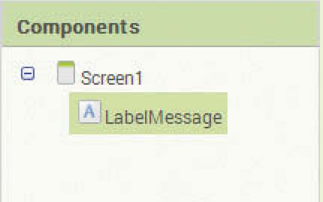
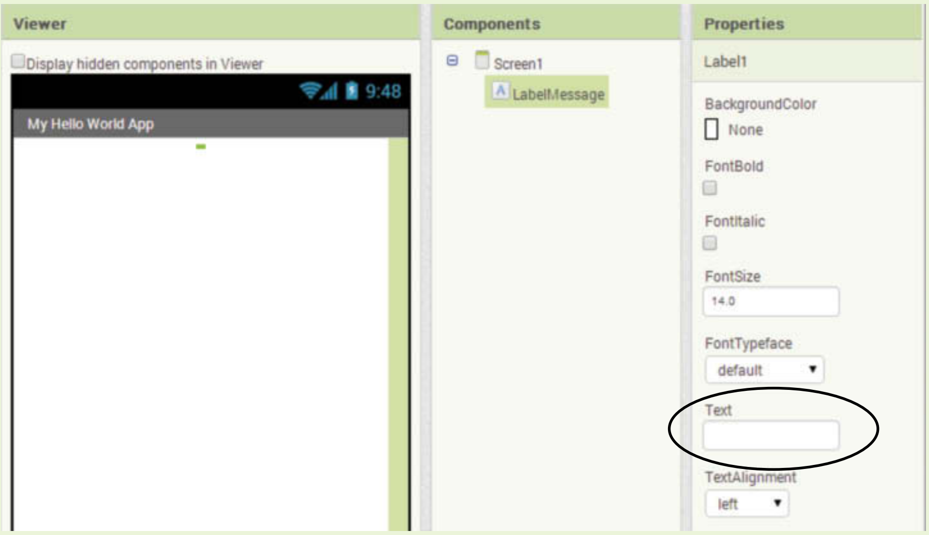


Figure 1-55 The Label's Text Property is Empty (*Source:* MIT App Inventor 2)

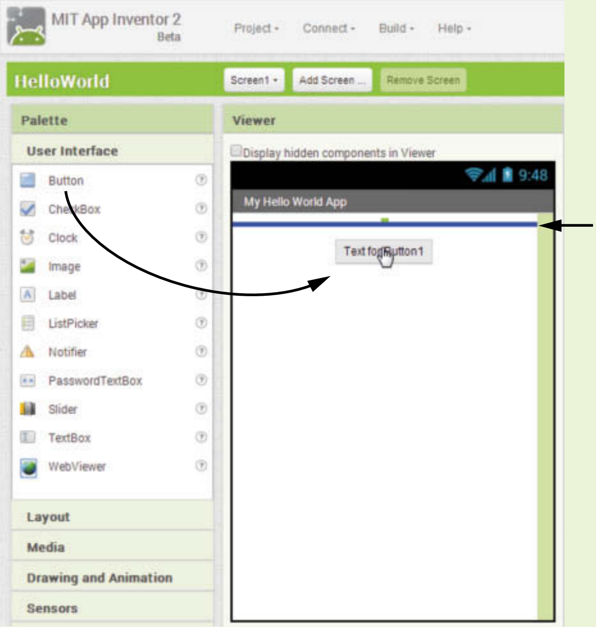


Step 7: Now you will create a Button component. Drag the Button component from the User Interface section of the Palette to the Viewer. Notice that as you drag the component, a thin blue line appears in the viewer, showing where the component will be inserted. You want the blue line to appear below the Label component, as shown in Figure 1-56, when you release the mouse button. This creates a Button component named `Button1`, with its Text property set to *Text for Button1*.

Step 8: Make sure the `Button1` component is selected in the Components column, and change the component's name to `ButtonDisplayMessage`. Then, in the Properties column, change the Text property to *Click Here To See a Message*. This is shown in Figure 1-57.

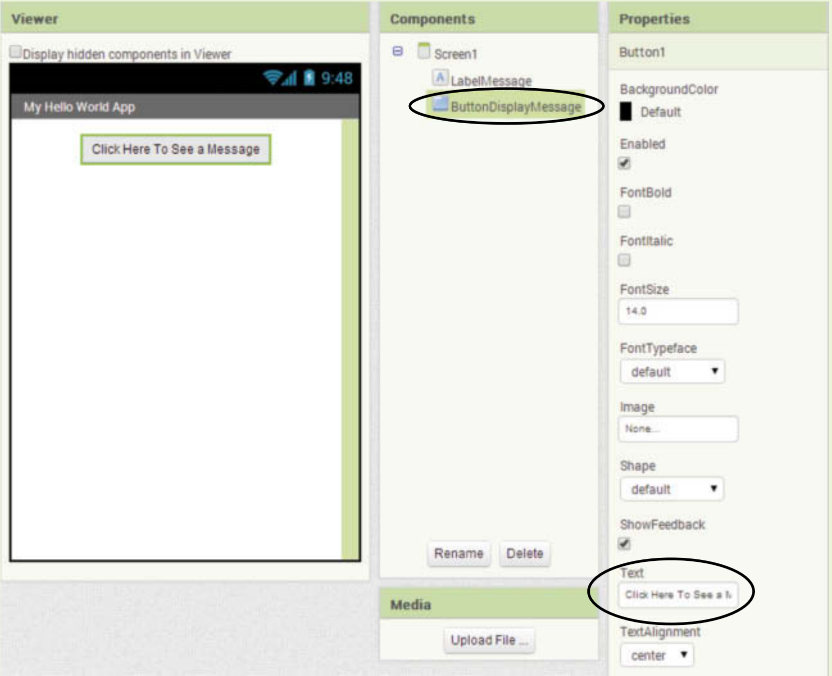
Step 9: You've added all of the components that you will need for this app. Although you haven't written any code, this would be a good time to preview the app's screen in the emulator. Click the *Connect* button in the upper area of the App Inventor screen, and then click *Emulator* on the menu that appears. It might take several minutes for the emulator to be created in the computer's memory. Once the emulator has been created and initialized, it will appear as shown in Figure 1-58.

Figure 1-56 Creating a Button Component (*Source:* MIT App Inventor 2)



The blue line shows where the component will be inserted.

Figure 1-57 The Button Renamed and its Text Property Changed (*Source:* MIT App Inventor 2)



BackgroundColor
Default

Enabled
☑

FontBold
□

FontItalic
□

FontSize
14.0

FontTypeface
default ▾

Image
None...

Shape
default ▾

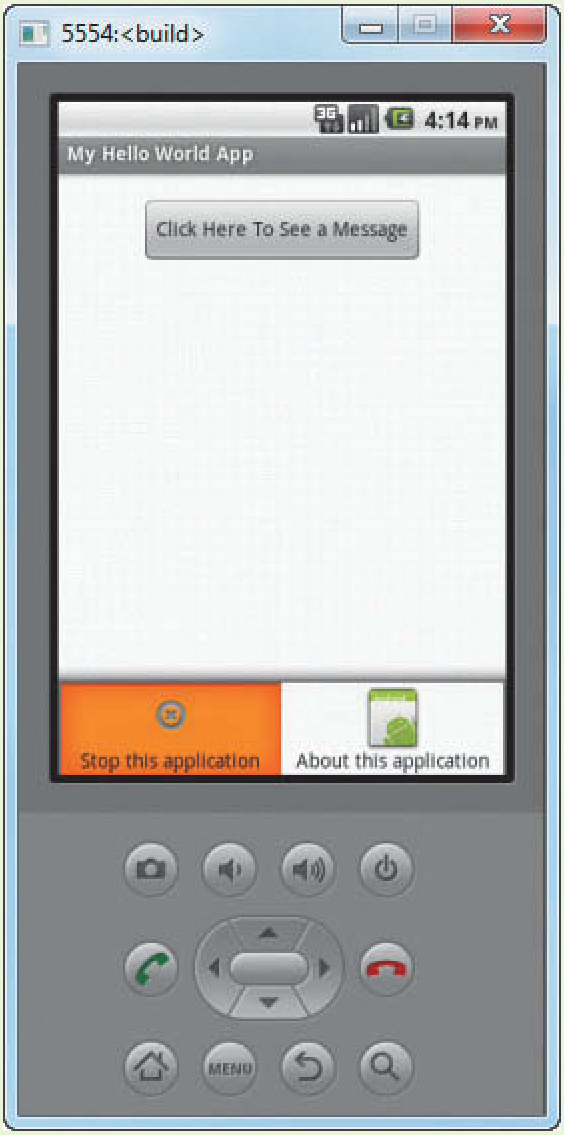
ShowFeedback
☑

Text
Click Here To See a M

TextAlignment
center ▾

If possible, leave the project open in App Inventor. You will finish the app in the next tutorial.

Figure 1-58 The App in the Emulator (*Source:* MIT App Inventor 2)



NOTE: Although the app is running in the emulator, it is not capable of doing anything other than displaying the screen. If you click the Button component, nothing will happen. That is because you have not yet written the code that executes when the button is clicked. You will do that in the next tutorial.

Programming with Blocks

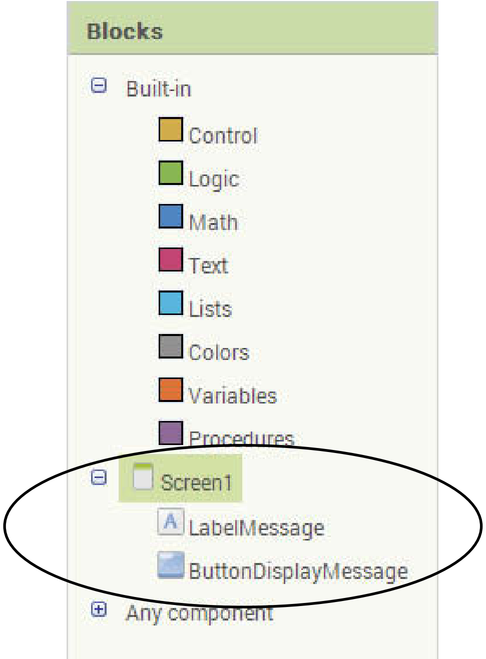
Before you continue with the next tutorial, we will discuss the steps that you must take to complete the Hello World app. Carefully read this section, and then perform the steps in Tutorial 1-3.

First, you need to understand that apps are *event-driven* programs. This means that when an app is running, it waits for specific events to happen, and then it responds to those events. What do we mean by event? An event is an action that takes place, such as the user clicking a button, or sliding his or her finger across the device's screen. An incoming text message is also an event, as well as when the user tilts or shakes the phone. When you are creating an app, you decide which events the app will respond to, and then you write the code that executes when those events take place. (Obviously, there are limitations to the events that the emulator can respond to, because it isn't a physical device. For example, emulators can't receive incoming phone calls, and they can't be tilted or shaken.)

Recall that the Hello World app has a Button component named `ButtonDisplayMessage`, and a Label component named `LabelMessage`. We want the app to display *Hello World* in the label when the user clicks the button. So, we need a block that executes when the user clicks the `ButtonDisplayMessage` component.

Assuming the HelloWorld project is currently open in the Blocks Editor, notice that the Blocks column has entries for `Screen1`, `LabelMessage`, and `ButtonDisplayMessage`. This is shown in Figure 1-59. Because you want to create a block that executes when the `ButtonDisplayMessage` component is clicked, you need to click `ButtonDisplayMessage` entry. This causes a "drawer" to open, revealing blocks that are related to the `ButtonDisplayMessage` component, as shown in Figure 1-60.

Figure 1-59 The Component Entries in the Blocks Column (*Source:* MIT App Inventor 2)


Blocks
⊟ Built-in
Control
Logic
Math
Text
Lists
Colors
Variables
Procedures
⊟ Screen1
LabelMessage
ButtonDisplayMessage
⊞ Any component