Android Apps with App Inventor

This page intentionally left blank

Android Apps with App Inventor

The Fast and Easy Way to Build Android Apps

Jörg H. Kloss

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data

Kloss, Jörg H.
Android Apps with App inventor : the fast and easy way to build android apps / Jörg H. Kloss.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-321-81270-4 (pbk. : alk. paper)
1. Application software–Development. 2. Android (Electronic resource) 3. Open source software. 4. Smartphones. 5. Mobile computing. I. Title.
QA76.76.A65K614 2012
005.3—dc23

2011047948

ISBN-13: 978-0-321-81270-4
ISBN-10: 0-321-81270-0
Text printed in the United States on recycled paper at Edwards Brothers Malloy in Ann Arbor, Michigan.
Second printing, September 2012

Editor-in-Chief
Mark Taub

Acquisitions Editor
Trina MacDonald

Development Editor
Songlin Qiu

Managing Editor
John Fuller

Project Editor
Anna Popick

Copy Editor
Jill Hobbs

Indexer
Jack Lewis

Proofreader
Lori Newhouse

Publishing Coordinator
Olivia Basegio

Cover Designer
Chuti Prasertsith

Compositor
Vicki Rowland

Translator
Almut Dworak

Contents at a Glance

Contents

Preface by Hal Abelson

Following is the original preface to this book by Dr. Hal Abelson, Professor of Electrical Engineering and Computer Science at Massachusetts Institute of Technology (Cambridge, Massachusetts), leading member of the Google App Inventor Team, and director of the new MIT Center for Mobile Learning supported by Google.

People have been doing personal computing since the 1980s. But today's mobile applications are making computing "personal" as never before. Today, we carry computers with us constantly, as smartphones and pads and the new devices that are regularly emerging. More significantly, today's personal computing is increasingly "about" us: where we live, where we work, who our friends are, what we buy, what we like, whom we talk with, and what we talk about. This personal computing is linked to global data services and information sources in a way that fundamentally transforms our experience and our perception of our world, just as television did for people beginning in the 1950s.

Television was a consumer technology. Anyone could enjoy television, but there was no way to adapt television to your personal needs, other than by selecting which program to watch from a variety of offerings from professional producers. Perhaps mobile computing will be similar, where we're all limited to choosing from among predefined applications supplied by professional developers.

When we created App Inventor at Google, we were motivated by the vision that mobile computing could be personal computing technology that you can actually personalize, by creating applications for yourself and your friends, without having to be an expert programmer. Perhaps you might create applications because you want to fulfill a special need, or learn about computing, or try your hand at distributing and selling applications, or just have fun.

App Inventor became available for general use in December 2010. It's still a beta system under development, and the Google team is working to make it more powerful and easier to use. But there is already a growing community of App Inventor users of all ages who are exploring and experiencing what it's like to make applications for themselves. Some of the things they are creating are:

- An application for sending and redeeming gift cards
- A guide to a major medical reference book
- A controller for a Lego robot
- An inventory tracker for a commercial vehicle manufacturer

- Educational programs in reading and mathematics for their kids
- Many kinds of games

You can make applications like these, too, and this book shows you how, starting with the basics of how to access the App Inventor system from the Google website and connect your mobile phone, through pointers on developing applications that use the phone's built-in accelerometer, orientation, and location sensors. Along the way, you'll get a solid introduction to creating applications with text and data and to working with images and animation. You'll learn how to control the phone's camera, how to manipulate databases on the phone and on the Web, and how to create games, send text messages and make phone calls, and manipulate maps. Each topic is accompanied by working applications and thorough explanations.

Could this be your first step toward a future in designing mobile applications? Perhaps. Even if it is not, you'll find that you can be creative and empowered with a technology that's playing an increasingly central role in your life, and the lives of us all.

—Hal Abelson
MIT Center for Mobile Learning
Google App Inventor Team
March 2011

Preface

There could not be a better time than today to start developing Android apps, for many reasons. Most importantly, developing your own apps has never before been easier than it is now with App Inventor. This development tool, which is offered by Google and Massachusetts Institute of Technology (MIT) and has been available since December 2010, is available free of charge for all to use. With App Inventor, you can develop your own apps, even if you have never programmed before, using a computer or even a smartphone. With App Inventor, you can build both small and really big apps with playful ease by assembling visual building blocks, without having to write a single line of Java code. Yet App Inventor is by no means just a toy: It is an alternative and innovative tool with which you can develop even complex and demanding apps quickly and easily, both for yourself and for other users. Take a look at the table of contents of this book, and you will be amazed to find that the early chapters of a book aimed at beginners contain instructions for developing apps in the areas of multimedia (photo, audio, video), graphics and animation, various forms of communication (speech, SMS, e-mail, web services), and even sensors (orientation, acceleration, GPS geoposition). In Figure P.1, you can see a selection of the apps you will develop in this book.

Figure P.1 Android apps developed in this book

The quick and easy start and the equally quick and intuitive development of attractive and demanding apps are the declared aims of the visual development tool App Inventor. App Inventor is aimed at a far larger target group than ordinary development tools. With App Inventor, all users of Android smartphones now have the chance to peek behind the scenes at the colorful world of apps, then have a go themselves and express their creativity by designing their own apps. Where those apps go is entirely up to the developer's individual preferences, topical emphasis, and personal motivation. You as a user can decide whether your own app is "just" a personal digital picture frame, a quiz game, a vocabulary trainer with a potentially shareable online database, or a geotracker for automatically creating a route profile while the user is hiking. The future will show to what extent the "users" of such personal apps eventually turn into "developers"—that is, whether they evolve from "passive consumers" to "active producers," thereby triggering a mini-revolution in dealing with the most modern forms of communication technology. Nevertheless, even experienced developers can profit from using App Inventor, as it enables them to produce professional prototypes and apps much more quickly and, therefore, more cost-effectively. Come and take part in these "developments" and get to know App Inventor by reading this book. You will learn to use it for your own purposes and soon appreciate its value as an immensely powerful developer tool.

Owing to its rich set of properties and features, App Inventor is in the right place at exactly the right time. Now that many billions of dollars have been invested in licenses and establishing the mobile telecommunications network infrastructure, the mobile data networks of the third generation (3G: UMTS, HSDPA) and fourth generation (4G: LTE) are available almost anywhere and at any time, with a data flat-rate tariff often being used as the basis for fast data services, mobile Internet access, and web services. In turn, the new developments by manufacturers of mobile devices keep coming in a rush. Moreover, after a deluge of new smartphones with incredible technical specifications for domestic use, the next generation of tablet PCs is being embraced by a growing number of customers who are willing to pay for them. The providers of online services and web services are also eager to make use of the new mobile possibilities and to offer the increasingly communicative Web 2.0 users mobile extensions such as location-based services and proprietary apps on the growing app market. The competition between the mobile operating systems appears almost calm and seems to be more or less settled, the "top dogs" have long since been pushed out, and, after an initial neck-and-neck race, the triumph of Android as the operating system of choice for smartphones now seems certain. With its open approach, the resulting flexibility, its free availability, and its integrative access to the entire range of features of the manifold Google Services, Android has qualities that the other mobile operating systems lack.

Despite the impressive technological advancements, users nowadays are no longer only interested in pure technical features and details. Although the Internet first started to develop thanks to just a few technologically minded enthusiasts, it has long since been transformed from a mass medium for consuming news, information, and entertainment (Web 1.0) to an active, commonplace form of communication between people (Web 2.0). Today the focus is not so much on the technology, but rather on the communication,

creativity, and individuality that people can apply and express with it. This trend partly explains the increasing willingness of users to invest time and effort into creating their own profiles in social networks, to establish their own blogs, and participate in chats or online games, but also to invest money in the form of fixed and mobile telecommunications charges and the newest and most fashionable hardware. Regardless of which forms of expression users' individuality and creativity may take in the age of digital and networked communication, App Inventor offers entirely new possibilities for exercising them. Previously, users were able to move only within the predetermined limits set by the hardware manufacturer, the platform operator, and the app developer; now they can at least overcome the latter by using App Inventor, thereby gaining a piece of freedom and independence—a factor that should not be underestimated. Even if you do not have such ambitious aims, you can still have fun when developing your own apps with App Inventor. After reading this book and working your way through the many example apps, you will see the colorful app world with different eyes. And almost without noticing it, you will have become a developer of Android Apps. So, what are you waiting for?

This page intentionally left blank

Acknowledgments

I would like to thank everyone who has supported me during the creation of this book, directly or indirectly. This includes not least the members of the Google App Inventor Team, whose incredibly dedicated work has made such a fascinating developer tool as App Inventor and, therefore, many impressive apps—and this book—possible in the first place. I am especially grateful to Hal Abelson, Professor at the legendary MIT, central impulse giver within the Google App Inventor Team, and director of the MIT Center for Mobile Learning, for his inspiring and courageous work on this and the previous projects and for providing the preface to this book.

For the English-language revised and updated edition of this book, I would like to thank my colleagues in the German publishing house, Brigitte Bauer-Schiewek and Angelika Ritthaler, and in the United States, Jill Hobbs, Trina MacDonald, Anna Popick, and Songlin Qiu. I am particularly grateful to Almut Dworak for the excellent translation work and, beyond that, for the helpful comments and feedback on the book.

My thanks also to those who had so much patience with me while I was writing this book: my parents, my sister, Maximilian, Benedikt, and, above all, Alexandra.

—Jörg H. Kloss
January 2012

This page intentionally left blank

About the Author

Jörg H. Kloss has worked for many years with innovative information and communication technology, including its development, programming, and use in both private and professional areas. His private-sector beginnings with the Amstrad CPC and the programming language Basic were followed by deeper explorations at university in the area of artificial intelligence and computer linguistics, working in Pascal, C, C++, and Java, but also in specialized languages such as Lisp and Prolog. Mr. Kloss was one of the early pioneers of virtual reality (VR), augmented reality (AR), and interactive 3D worlds on the Internet. He began development work on commercial VRML-based online information systems in the mid-1990s, has worked at the renowned German VR lab of the Fraunhofer Institute for Industrial Engineering (FhG-IAO) and the American VR-Entertainer StrayLight, and was president of the European division of the VR Alliance of Students and Professionals (VRASP). In addition to numerous presentations, contributions, and other publications, Mr. Kloss has written two books that were published (in German) by Addison-Wesley: *VRML97: Der neue Standard für interaktive 3D-Welten im World Wide Web* (*VRML97: The New Standard for Interactive 3D Worlds in the World Wide Web*; 1998) and *X3D: Programmierung interaktiver 3D-Anwendungen für das Internet* (*X3D: Programming Interactive 3D Applications for the Internet*; 2010).

After developing early industrial projects based on 3D multiuser worlds for an international media house as well as for remote maintenance via the powerline of a large energy supplier, Mr. Kloss focused on telecommunications for many years, taking part in innovative projects involving multimedia data and voice technologies in the areas of fixed and mobile network communications (IP, TDM, VoIP, 3G, 4G). As these technologies have converged, Mr. Kloss has dealt increasingly with the potential of mobile data networks and services in the context of mobile augmented reality, ubiquitous computing, and contextual services. He has actively taken part in the development of Android apps with App Inventor since the early closed-beta phase.

This page intentionally left blank

Introduction

This book is a compendium, a practical course book, and a comprehensive tutorial in one, offering a collection of example projects for smaller and larger applications (apps) for Android devices. As a compendium, it addresses, introduces, and demonstrates more or less comprehensively every single area and almost every component of the App Inventor development tool as it was available at the time when the book was written. Consequently, this book can be used as a reference work even by experienced developers who are looking for specific instructions and information about a certain functional area. Presenting examples from a wide variety of topics, it also serves as a practical course book on the general development of apps for mobile devices with their specific multimedia, communication, and sensory properties as well as system elements that often remain uncharted territory even for the experienced PC programmer. Along with the basic aspects of application development, program structures, and functional elements, the example projects demonstrate approaches and solution strategies for the typical problems that can arise in the context of mobile applications.

As a comprehensive tutorial, this text is aimed mainly at beginners and their needs. Both the structure of this book and the development tool App Inventor are written with beginners in mind, with a clear focus on practical application. If you are a newcomer to programming in general or to app development for mobile devices in particular, specifically for Android smartphones, or if you are simply adopting the development tool App Inventor, the introductory chapters of Parts I and II of this book will provide you with the level of knowledge you need and guide you step by step through the development of Android apps with App Inventor. The many accompanying example projects and apps illustrate and extend what you have learned, invite you to experiment and try things out for yourself, and provide a starting point inspiring you to creatively develop your own apps. You will learn progressively in line with the sequential structure of the book's chapters, the topics and functional areas addressed, and the example projects we develop, all of which usually build on the knowledge gained in the preceding chapters. Along with covering the many functional areas and elements, the book also discusses basic methods of program development and explains how to use App Inventor's online resources, thereby preparing you to undertake your own development work in the future. In the process, the perspective gradually changes from the initial perspective of the beginner looking at individual components and their functions to the view of the developer focusing on the actual tasks the app performs and strategies to implement them with App Inventor.

Structure and Overview

If you take a quick look at the table of contents, you will see that this book is divided into five parts. These parts are not so much devoted to different topics, but rather reflect the intended evolution of the reader from the beginner working with App Inventor for the first time to the developer of advanced and complex apps—which we certainly hope will occur while you are reading this book and working through the chapters. This structure emphasizes the tutorial nature of this text; thus we recommend reading and working through the chapters in order. Even if you have previous knowledge of and experience with App Inventor, you should at least skim through the first few chapters to make sure you have the knowledge base that is essential for understanding the topics covered in later chapters.

Part I: Preparing Your First App

There is no way around installing the software required for App Inventor. The first chapter, "Preparation and Installation," guides you through the sometimes bumpy and not always obvious procedure of checking and setting the required system parameters on your computer, the download and installation of the App Inventor Setup Software, the obligatory registration for the online development platform, and the setup of the development parameters on your smartphone. After successful setup, you will explore the development environment of App Inventor in Chapter 2, "The Development Environment," where you learn how to use the program, explore its areas of application, and encounter its development elements in the two central AI interfaces, Designer and Editor. You will also discover how to integrate your smartphone into the development environment and what to do if you are having start-up problems. In Chapter 3, "Developing Your First App," you at last begin developing an app—that is, you design the user interface and develop the functions of your first app, "LaughBag." Once you have added a custom default icon to your app, you will discover different options for installing it on the smartphone or exporting it as APK file. This information lays the foundation for all further app projects.

Part II: Easy Projects as a Warm-Up

Before you develop your next app, Chapter 4, "Basic Terms and Central Concepts," introduces key ideas such as properties, events, and methods. Chapter 5, "The AI References," walks you through the current components, blocks, and concepts of App Inventor and those expected to included in future versions of the software. Equipped with these fundamentals, in Chapter 6, "Graphical User Interface," you use the Designer to create the UI of a demo app, becoming familiar with and actively using components such as buttons, text boxes, and check boxes. In Chapter 7, "Multimedia," you explore the topic of multimedia and its components by taking photos and looking at them in a demo app, creating a voice recording, playing audio and video files, and making the smartphone vibrate. Next, in Chapter 8, "Example Project: Creating a Media Center,"

you expand this demo project in the form of a media center, an optically elaborate and ergonomically designed multimedia app with multiple screens.

Part III: On the Way to Becoming an App Developer

After your quick trip through the colorful world of graphical user interfaces and multimedia functions and now that you have acquired a good sense of how easy it is to create apps with the components of App Inventor, Chapter 9, "Program Development Basics," leads you more deeply into the development of apps with blocks and block structures. A comprehensive overview provides key details about data types, data structures, and control structures, with which you can implement every conceivable functionality using App Inventor. Quick demo apps show you how to create colors; process numbers; check logic states; edit texts and strings; use variables, procedures, and lists; and control the program flow with branches and loops. Next you will find tips on program development in the discussion of App Inventor's Editor component, followed by sample projects in which you implement a traditional calculator, a number guessing game, and a vocabulary trainer as apps. Next, Chapter 10, "Storage and Databases," explains how to save data locally on your smartphone or online on a web server and how to load them from there. To practice these skills, you expand the vocabulary trainer by developing a master and a client app with a common online database and vocabulary in the cloud.

Part IV: Developing Attractive Apps

Building on the foundations of your newly acquired developer knowhow, we then turn to the really interesting apps and more challenging areas of app development. Chapter 11, "Graphics and Animation," dives straight into the topic of graphics and animation—a rather advanced topic, but one that App Inventor makes it easy to cope with. After a brief introduction to the subject area, you develop a drawing program app, in which you can draw objects on the smartphone using your finger; the app even includes an undo function. Next you learn to animate graphic objects and use collision recognition to facilitate for realistic movement simulations. To turn your new knowledge into practice, you create a 2D squash game with a scoring function and dynamic difficulty level. You also learn to use timer events for any kind of animation, develop an app for drawing keyframe animation paths with your finger and an alarm clock app that will wake you from your dreams even when it is in standby mode.

Chapter 12, "Sensors," covers a topic that is considered exotic even by experienced developers. Here, you get to know the smartphone sensors and learn about their functions and, above all, their integration into your apps. You use the orientation or position sensor and its measurements to implement a fully functioning compass app with graphical compass needle or even a graphical spirit level. You can get musical with the acceleration sensor and develop a shaker, whose sensitivity you can regulate via a slider bar, plus a balance game similar to the classical "Labyrinth." In keeping with the trend of providing location-based services, you discover ways to use the GPS sensor in your apps, by

developing a geotracker for recording route profiles that you can automatically set online in real time, plus an app for geocaching complete with compass, direction, and distance indicator to the next cache.

Of course, we won't forget the almost classical area of communication—the topic of Chapter 13, "Communication." In this chapter, we work through this subject and the associated functional areas in our large and practice-oriented final project "Driver Assistance System," whose requirements and tasks we will analyze, structure, and then implement step by step in modules, following the same development path taken by professional developers. First, you are asked to integrate a module for telephone calls via speed dialing under an ergonomic interface with multiple screens. This is followed by development of a module for fully automatic receiving, processing, and answering of SMS messages; this module enables you to read incoming SMS messages aloud via a text-to-speech option and to dictate outgoing SMS messages via a voice recognition capability. Chapter 13 then introduces a central interface concept of App Inventor's Activity Starter component for calling and integrating other apps and web services, with which you can both expand the functional range of the developer language and integrate any external services into your apps. Through various modules, you learn to integrate Google Maps with your app to find the way back to where you parked your car, or Google Navigation to navigate the car driver home or to his or her workplace with one press of a button. Using a module for sending e-mails, you can inform any passengers of your current location and the time you will pick them up. Last but not least, the other central interface based on App Inventor's Web component is introduced for exchanging data with web services via their APIs. By implementing a ticker module with the latest news and stocks data, you learn how to develop information mashups based on real-time data access to the web APIs from Yahoo and Feedzilla, whereas the websites with the original full-text news can be shown directly in your app by using AI's WebViewer component. These capabilities turn your driver assistant system into a full-fledged and powerful app for serious everyday usage.

Part V: Useful Things for the Developer

Even experienced developers—among whom you can definitely count yourself after working through Chapters 1 through 13—can always learn something new and useful that they should know and keep informed about. Chapter 14, "Special Functional Areas," reveals the application-specific components of App Inventor for communicating with Twitter, for scanning barcodes, for online voting, or for using the online database of Google's Fusion Tables. This chapter also provides an overview of the dedicated component groups for developing online multiplayer games, exchanging data via Bluetooth, controlling robots from Lego Mindstorms construction sets, or even combining App Inventor with the app development in Java via the App Inventor Java Bridge.

Chapter 15, "Tips and Tools," offers helpful tips for working with the supported media formats, using the Java console, and setting up the speech module. The Appendix describes the many project, media, and APK files available on the companion website and lists further sources of information and interesting links.

Companion Website

On the companion website for this book, you can download ("Downloads") all demo and example projects from the book, as well as all media files such as the pictures or sounds needed for the projects:

```
www.informit.com/title/9780321812704
```

Please read the chapter "On the Companion Website" in the Appendix "Additional Resources" of this book for further details about the contents of the companion website and how to use them for your work with this book.

Requirements

One of the central characteristics of App Inventor is that you do not need to meet any special requirements to be able to develop small and large Android apps with this tool. Much like this book, App Inventor is aimed primarily at beginners in app development; no previous knowledge in programming Android smartphones, either general or specialized, is required to use this tool. If you are interested in smartphones, apps, and mobile data services, and you use them regularly to check your e-mails and update your social networks, then you should already have the prerequisites and—above all—the motivation to take the next step, have a look behind the scenes of the colorful app world, and start developing your own apps. Whether these are small helper applications or gimmicks, your own SMS manager or location-based games, or useful apps for work, everyday life, leisure time, or your club or group, with App Inventor you can turn your own ideas into apps without having to program a single line in Java. This book will show you how to do all that.

Of course, it would be a good idea if you had an Android smartphone at your disposal. From Android version 1.6 onward, you will be able to try out and use practically all of the functions described and apps developed in this book directly on your smartphone. Even if you do not yet have an Android smartphone, or if you have a smartphone running an older version of Android (prior to 1.6), you can still use App Inventor and the Android Emulator included with it to develop nearly all of the apps described in this book, test their functions, and then install them on a friend's Android smartphone. One part of the development environment for App Inventor runs on your local computer and another part on a web service, so that you need to have a PC or notebook and DSL access or a similar connection to the Internet available.

The software you need for App Inventor is available free of charge. To download and use the software, you simply need to register for the free service. This book shows you how quickly and easily all of this can be achieved and also guides you through the installation and the setup of the development environment.

Many apps—both in general and those described in this book—make intensive use of the Internet and online web services and will use your mobile data connection to do so. Do not forget that you are paying for this mobile data connection while you are testing and using your own apps. A flat-rate data plan with your cellphone provider can help you avoid unpleasant—and costly—surprises.

History

When it comes to the developer tool App Inventor and even the mobile operating system Android, it is difficult to talk about "history," given that this history is still rather young. In fact, you now stand at the forefront of this history and are driving this wave of innovation by developing apps for Android smartphones with App Inventor! Since Android was first released in October 2008, the Linux-based (and, in many respects, open) operating system for different mobile devices that was initiated by the Open Handset Alliance (OHA)—an alliance of producers, Net companies, and service providers within the telecommunications sector—and marketed by Google has developed from a niche product to a market leader. In the wake of the incredible increase in sales revenues, rapidly increasing market share, displacement of former market leaders, and practically exploding Android market, more and more producers and providers are announcing devices and services for Android, as they do not want to miss the boat.

App Inventor at Google

Good job—we now have App Inventor! With it, you may be able to actively participate in this growth process, if you have the ambition to do so. Nevertheless, it seems to be pure coincidence that free access to App Inventor was provided at the same time as the hype surrounding Android and mobile data services ramped up. When App Inventor was announced in July 2009 as an experimental teaching and learning tool for a select circle of American institutes of higher education (MIT, Harvard, and so on), the developer tool was mainly aimed at giving pupils and students easy access to programming in general and mobile devices in particular, while taking into consideration the most modern forms of information and communication technology such as social networking, location-based services, and web services within a cloud. Almost exactly one year later, App Inventor at Google was announced to the public in July 2010 and made available to interested developers as a closed beta version, albeit after they submitted an application and access was approved by the Google App Inventor Team. The period during which these would-be developers had to wait for their access being granted was marked by impatient debates in which some applicants compared the process to having to wait for Santa Claus and the long-awaited Christmas present (see Figure I.1)—an attitude that reflects the strong interest in a developer tool such as App Inventor.

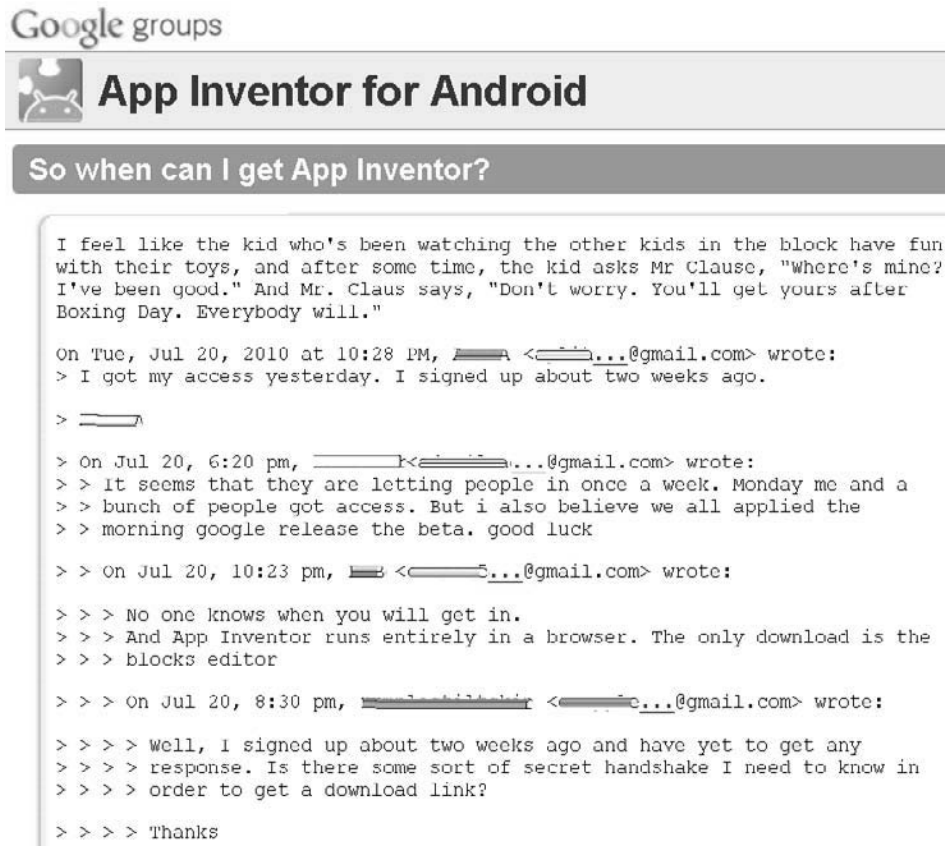


Figure I.1 Impatiently waiting to be granted access to the former closed beta version

Once access was finally granted (see Figure I.2), a new phase of intensive testing of the development environment, functions, and capacity of App Inventor began. This phase, under real conditions of use and load, was marked by adaptations, optimizations, and a vivid exchange between the Google App Inventor Team and the App Inventor closed beta users.

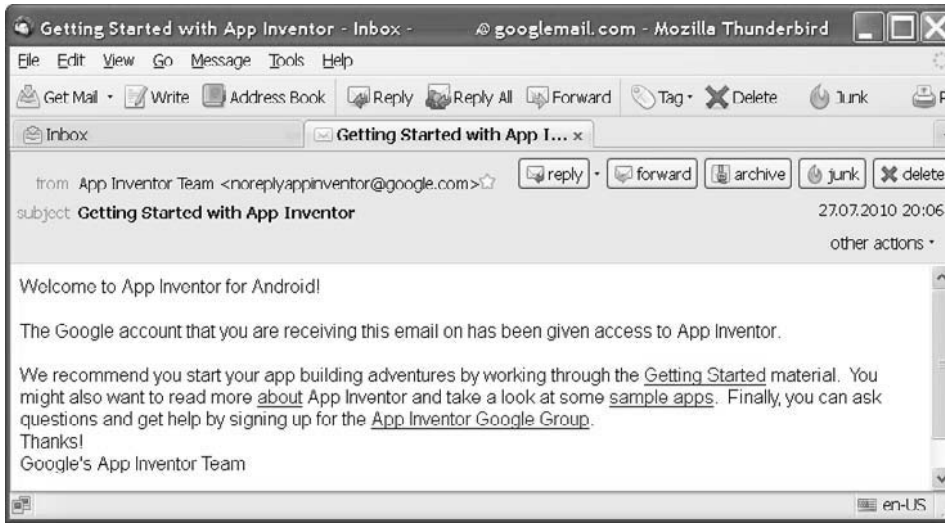


Figure I.2 Admission to former closed beta version

Over time, the development platform was tested extensively under realistic conditions, subjected to certain optimizations, and finally moved on to a stable basis version. On December 15, 2010, the Google App Inventor Team announced entry into the open beta phase and decided to open the development platform to any interested user without any access restriction (see Figure I.3).

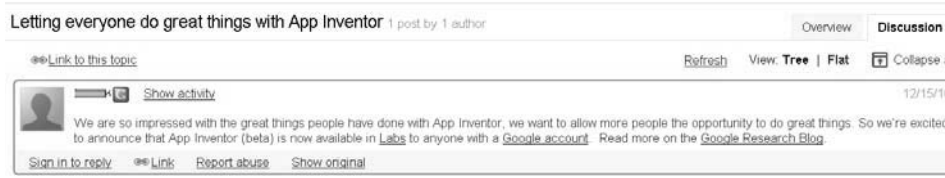
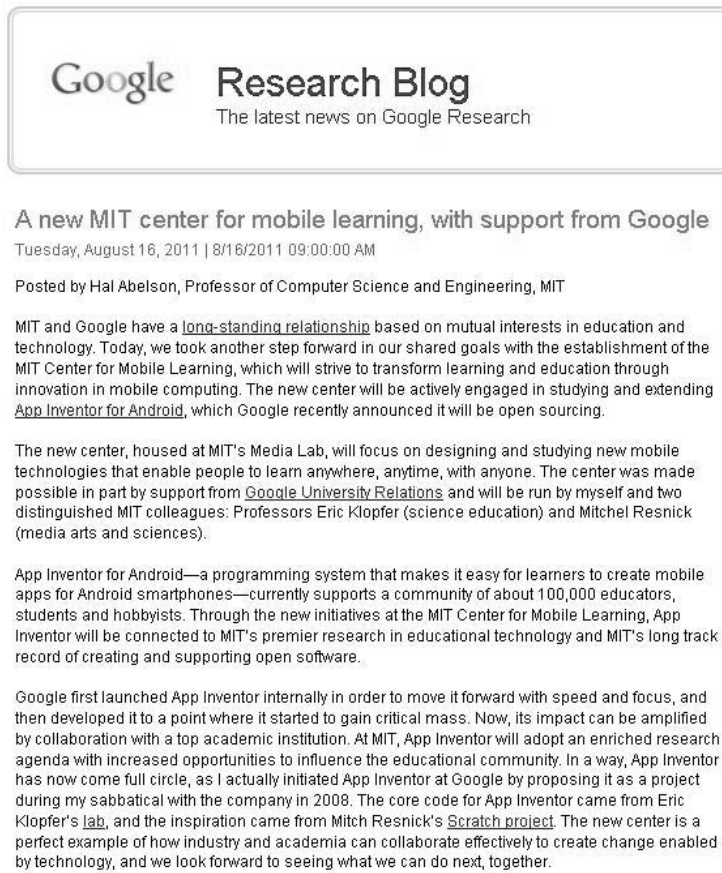


Figure I.3 Start of the open beta phase with access to anyone

Since that day, there has been even more buzz in the already turbulent Android market, and developers and users alike are feverishly working on new Android apps with App Inventor all over the world. The Google App Inventor team has been continuously expanding the developer platform and adding more and more powerful and extensive functions to App Inventor. During the year 2011, App Inventor became established as a real alternative to the Java-based Android Software Development Kit (SDK), and on the various Android markets you will find more and more apps that have been developed with App Inventor.

Open Source and App Inventor at MIT

With the year 2012, Google heralds a new era for the popular developer tool App Inventor. As with the open operating system Android itself, Google is now offering the platform for visual app development for Android as an open-source version, free of charge (see the first clause in Figure I.4). Similar to the successful concept whereby manufacturers can adapt the mobile operating system to their Android hardware, the development environment App Inventor can now also be run on different platforms by independent providers and can be adapted and developed further depending on the desired focus. For the users of App Inventor, this results in the availability of alternative platforms on which they can develop their apps depending on their personal preferences. This move will increase the multitude of options even further and ensure the development of more new features of App Inventor in the future.



Google Research Blog
The latest news on Google Research

A new MIT center for mobile learning, with support from Google

Tuesday, August 16, 2011 | 8/16/2011 09:00:00 AM

Posted by Hal Abelson, Professor of Computer Science and Engineering, MIT

MIT and Google have a long-standing relationship based on mutual interests in education and technology. Today, we took another step forward in our shared goals with the establishment of the MIT Center for Mobile Learning, which will strive to transform learning and education through innovation in mobile computing. The new center will be actively engaged in studying and extending App Inventor for Android, which Google recently announced it will be open sourcing.

The new center, housed at MIT's Media Lab, will focus on designing and studying new mobile technologies that enable people to learn anywhere, anytime, with anyone. The center was made possible in part by support from Google University Relations and will be run by myself and two distinguished MIT colleagues: Professors Eric Klopfer (science education) and Mitchel Resnick (media arts and sciences).

App Inventor for Android—a programming system that makes it easy for learners to create mobile apps for Android smartphones—currently supports a community of about 100,000 educators, students and hobbyists. Through the new initiatives at the MIT Center for Mobile Learning, App Inventor will be connected to MIT's premier research in educational technology and MIT's long track record of creating and supporting open software.

Google first launched App Inventor internally in order to move it forward with speed and focus, and then developed it to a point where it started to gain critical mass. Now, its impact can be amplified by collaboration with a top academic institution. At MIT, App Inventor will adopt an enriched research agenda with increased opportunities to influence the educational community. In a way, App Inventor has now come full circle, as I actually initiated App Inventor at Google by proposing it as a project during my sabbatical with the company in 2008. The core code for App Inventor came from Eric Klopfer's lab, and the inspiration came from Mitch Resnick's Scratch project. The new center is a perfect example of how industry and academia can collaborate effectively to create change enabled by technology, and we look forward to seeing what we can do next, together.

Figure I.4 App Inventor with Google and MIT (http://googleresearch.blogspot.com/2011/08/new-mit-center-for-mobile-learning-with.html, as of August 17, 2011)

As one of the first providers, the Massachusetts Institute of Technology (MIT) is now making the new App Inventor available online to the public on its systems, thereby consistently continuing the tasks of the previous Google App Inventor Group (see the announcement in Figure I.4). The release process followed the original by having started with an experimental version followed by the stable public version. The cooperation between Google and MIT, which has existed since the earliest days of App Inventor, will continue in the future as well, in form of the jointly founded and financed Center for Mobile Learning (CML) at the MIT Media Lab (Figure I.5).

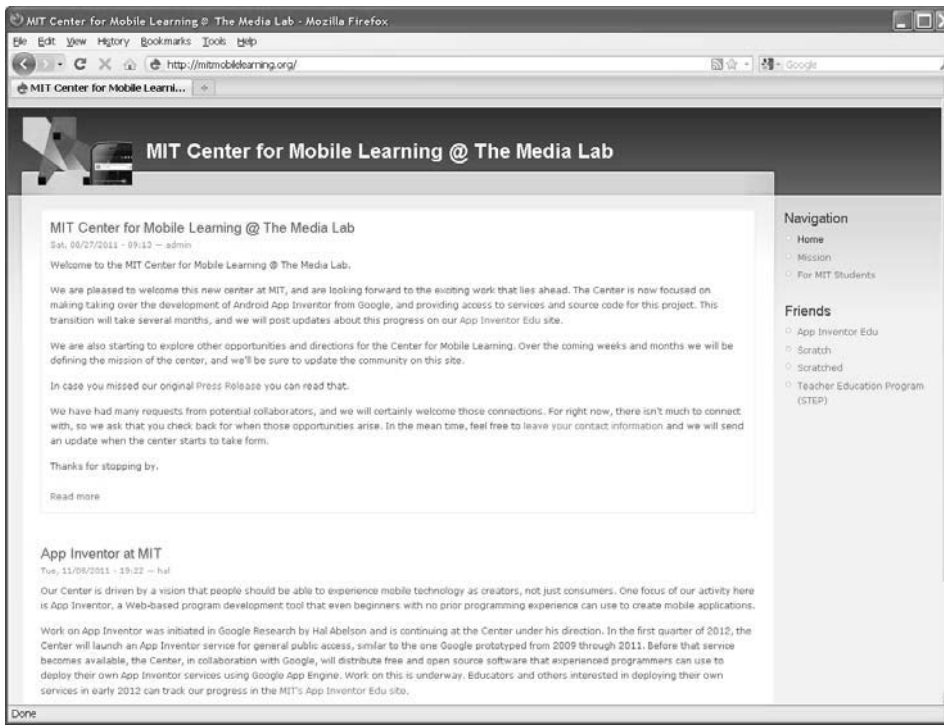


Figure I.5 App Inventor at the Center for Mobile Learning at the MIT Media Lab

This venture also provides for continuity in terms of personnel, notably Dr. Hal Abelson, who as initiator of the former Google App Inventor Group is now running the new CML, which is responsible for operating and further developing App Inventor. Figure I.6 shows the new central website for App Inventor at MIT, from where you can start to explore documentation and tutorials (Explore), special information for educators, and even curriculum materials (Teach), and—of course—also launch App Inventor (Invent).

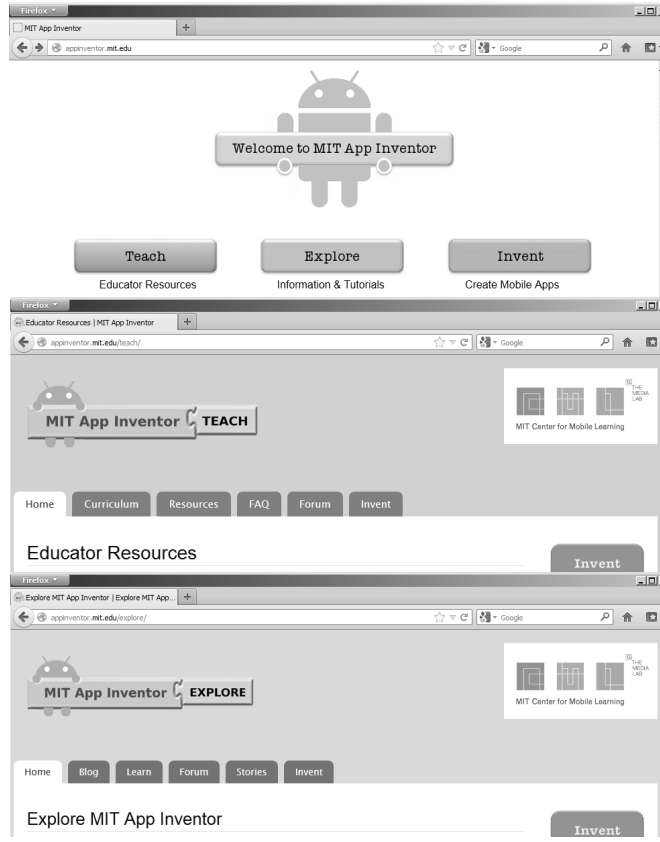


Figure I.6 Central information website for App Inventor at MIT (http://appinventor.mit.edu/)

The success story of App Inventor for visually developing Android apps continues and is expected to gain further momentum. You can become a part of this story by starting to develop apps for Android, the most widely used mobile operating system. Read this book to find out how you can use the new App Inventor to turn your ideas into reality and create your own exciting, impressive, and unique apps!

This page intentionally left blank

Chapter 3

Developing Your First App

Finally! In this chapter you will have a chance to develop your first very own Android app with App Inventor. Now that you have fulfilled all the preparation and installation requirements mentioned in the previous chapters and created your first Project "HelloAndroidWorld," all that effort is ready to pay off. And as you know, you learn best while having fun. For that reason, our first app will be all about laughter. You may remember the "laugh bag"—a popular gag toy of the 1970s that is still around today. This canvas bag contained a little plastic box inside it that would laugh like mad when you pressed on it. The kids in the 1970s had great fun with it, so why should it be any different in our age of the smartphone?

Developing the laugh bag app will introduce you to all the basic elements of app development with AI. In AI Designer, you will put together the app's graphical user interface from various components, integrate different media types, connect the components together in the AI Blocks Editor, and develop the app functions by combining the blocks to create block structures. In addition to testing the app within the AI development environment, we will prepare it as independent app so we can upload it to any Android device using a variety of export options. We will also address any problems that might crop up during app development with AI and present solutions or workarounds. By the end of the chapter, you will have taken a great step toward becoming an app developer with AI and will have acquired sufficient knowledge to be able to develop simple apps independently.

The step-by-step guide to the development process provided in this chapter reflects the most basic approach to creating an app. With simple apps, it is indeed possible to create the complete app interface in a single step and then to complete all of the block structures in a second step. For more complicated app projects, however, you will likely need to jump back and forth between the Designer and Blocks Editor and, therefore, between the individual working steps, or to run through the development cycle repeatedly for the various functional components and block structures. In the course of the book, you will become familiar with both approaches and learn to make seamless transitions between them.

Creating the Project "LaughBag"

In Chapter 2, we created our first project titled "HelloAndroidWorld." This project was automatically saved on the provider servers when you left the AI development environment. If you now reopen App Inventor with your user data, the status of your latest project is loaded and displayed automatically. Start AI by going to the start page in your web browser and sign in. It makes sense to save the web address as a bookmark in your browser to speed things up if you will be frequently developing apps with AI.

> **Sign In and Start AI Designer**
>
> Start the AI Designer by logging in to the website of your AI provider.
>
> The links to the experimental and public versions of MIT AI are:
>
> `http://experimental.appinventor.mit.edu`
>
> `http://beta.appinventor.mit.edu`

Click on the button labeled "Sign in" so the web browser begins loading AI Designer. On loading, you may briefly see the project overview before the view changes to the central interface with its five panels, into which the most recently edited project ("HelloAndroidWorld") is loaded. We recommend that you now also start the AI Blocks Editor by clicking the button labeled "Open the Blocks Editor" and connect it to your Android smartphone via the USB cable by clicking the button "Connect to phone" (see the description of this process in Chapter 2).

> **Start AI Designer, Blocks Editor, and Phone Connect Immediately**
>
> If you always start the AI Blocks Designer in addition to the AI Designer when developing apps, and you integrate your smartphone as well, you can follow all visible development steps you carry out in the Designer on the smartphone screen.

If your smartphone is connected straight away to the AI development environment, you can always see the following development steps on your smartphone screen and try them out. Alternatively, you can start the Blocks Editor and smartphone at a later stage; we will remind you of this possibility at a later point.

To create a new project in AI Designer, go to the project overview by clicking on "My Projects," where you will see the project "HelloAndroidWorld" listed. Create a new project titled "LaughBag" by clicking on the New button. Enter the project name "LaughBag" in the pop-up window and confirm by clicking OK, as shown in Figure 3.1.

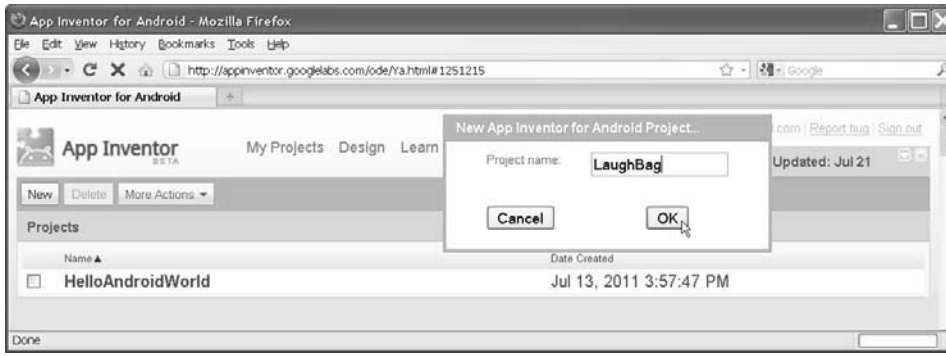


Figure 3.1 Create the new app project "LaughBag" in AI Designer

The AI Designer now creates the new project, saves the default configuration on the remote provider servers, and loads it together with the component groups into the now opening interface. The status bar at the top informs you of this process with short status messages such as "Save..." and "Load...." As our first project, LaughBag appears in the familiar default configuration. Apart from the component groups in the Palette, the Viewer shows only the starting component "Screen1," which appears as the empty screen area and is selected by default in the Properties panel.

As you can see in Figure 3.2, you can now change the app title from "Screen1" to "LaughBag" by editing the Title option accordingly. The app title will also appear on your smartphone when you use the app later. Press your computer's Enter key, and the title displayed in the Viewer also changes.

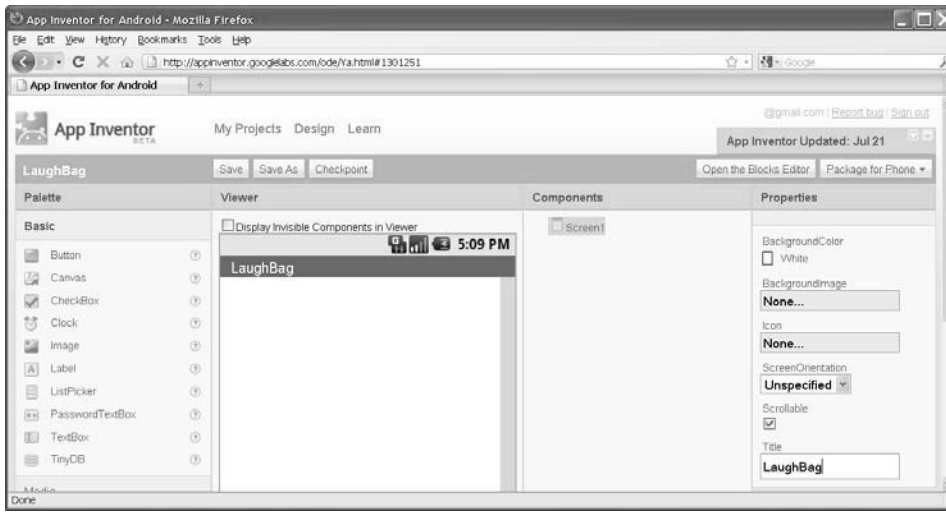


Figure 3.2 Default configuration for the project LaughBag

Checking Your Smartphone

Have you already connected your smartphone via the AI Blocks Editor? If so, you can check whether the app title has changed.

We will leave the other settings of the Screen component unchanged. The default setting "Unspecified" in the ScreenOrientation box will later automatically adapt the screen orientation horizontally or vertically depending on how the user is holding the smartphone. Setting the screen to vertical (Portrait) or horizontal orientation (Landscape) is recommended only in special situations. To make sure all user elements of the app are visible even on small screens, you should normally also leave the screen area set to "Scrollable."

Designing the User Interface

Once the project has been created and the app title changed to LaughBag, the creative part of the app development can begin. It is not a coincidence that the AI interface responsible for this task has the name Designer, because it is where you design the user interface for the future users of your app. As mentioned in Chapter 2, "The Development Environment," the user interface not only enables the user to enter input, but also provides interactive output of the relevant results to the user. The elements of the user interface are visual (e.g., text, buttons, images), of course, but also they allow for multimodal input (e.g., microphone, camera, accelerator, location sensor, GPS) and output (e.g., sound, vibrations, movies), sometimes by accessing remote sources (e.g., web services, SMS, Bluetooth). The multitude of components AI provides for this purpose has now become so great that it is difficult to categorize them comprehensively, and the Google developers are fervently doing their best to further expand the functionality of AI.

Consider Ergonomics

The significance of the design task for an app's success cannot be emphasized enough, as the user friendliness or ergonomics is a decisive factor in determining the ultimate success of an app—which, after all, is nothing more than a small mobile software program. The more intuitive, effective, and useful an app is for an intended task in the special situation of mobile use, the more the users will use the app, recommend it to other users, and download it. Try to consider this point while you are designing your apps, and imagine the situation and requirements of your target group. For example, if your app is aimed at drivers, the buttons and text labels should be big enough to read and use comfortably and—above all—safely during a car ride.

If you are interested in finding out more about app ergonomics, you can research this topic further on the Web. A whole range of recommendations and suggestions can be found online for the design of Android apps icons alone.

In this book, we will focus less on the creative or ergonomic aspects and more on the functional variety of the AI components. In turn, you should not have too high aesthetic expectations regarding our laugh bag—even the original devices from the 1970s were not very impressive in that respect.

Inserting the "Label" Component

The user interface of the LaughBag app will be fairly simple and consist of only two components, aside from the screen component. First, we want a brief text to describe how to use the app ("Please press the bag!"). Second, we want the bag itself to have several properties: It should look like a bag, and you should be able to press it to make it laugh. These properties may sound rather complicated, but they are actually just the same as for any button you know so well from a Windows or app environment. Thus the user has to be able to press or tap the button, and this button gets its individual look because you attach a text label to or image positioned on the button like a sticker.

> **"There Is More Than One Way to Skin a Cat"**
>
> Don't worry—no cats are harmed in this book. We just want you to be aware of this fact, especially as a beginner in app development. There are always multiple ways of realizing an app. Most apps fulfill a more or less useful purpose. How you achieve that goal largely depends on your creativity as a developer. AI offers a huge collection of tools that you can use to build your app. Whether you realize the laugh bag as an elaborately animated 3D graphic with haptic feedback or much more simply as a button with a picture on it is entirely up to you and your level of motivation. Both variations fulfill their primary purpose: The laugh bag laughs if you press it. If you want to demonstrate the particular capacity of your high-end smartphone with Android 2.3 or even 4.0, a 3D laugh bag will certainly more impressive than a 2D button, but the button will run without limitations even on simple smartphones with Android 1.5. You have to weigh the resources and effort involved against the desired effect.

Our modest intent is to create a manageable app for beginners that can run on as many Android devices as possible, so we use only two components in designing the visual appearance of our LaughBag app:

- *Label*: A text field that can receive any text and be placed onto the screen like a label.
- *Button*: A button that can also have a text and/or an image in addition to its switching function.

> **Further Information in the Component Reference**
>
> Further information on functions and properties of all components can be found online in the Component Reference:
>
> `http://experimental.appinventor.mit.edu/learn/reference/components/`

Together with all the other components, you will find the Label and Button in the AI toolbox—that is, in the Palette panel on the left in AI Designer. The two components are part of the "Basic" group. If it is not yet open, please click on the group name to display the components contained within it. Button is in the first position, Label in the sixth position.

We want the message "Please press the bag!" to appear in our LaughBag app, just below the title bar. We will first place the label in the Viewer, though we can move the components at a later stage if required. Grab a label in the Palette by clicking on the name or the icon of the component "Label," then hold the mouse button and drag the label into the Viewer. While you are dragging the mouse, the pointer changes from the usual mouse pointer icon to a text field—the label—containing the default text "Text for Label1," as shown in Figure 3.3.

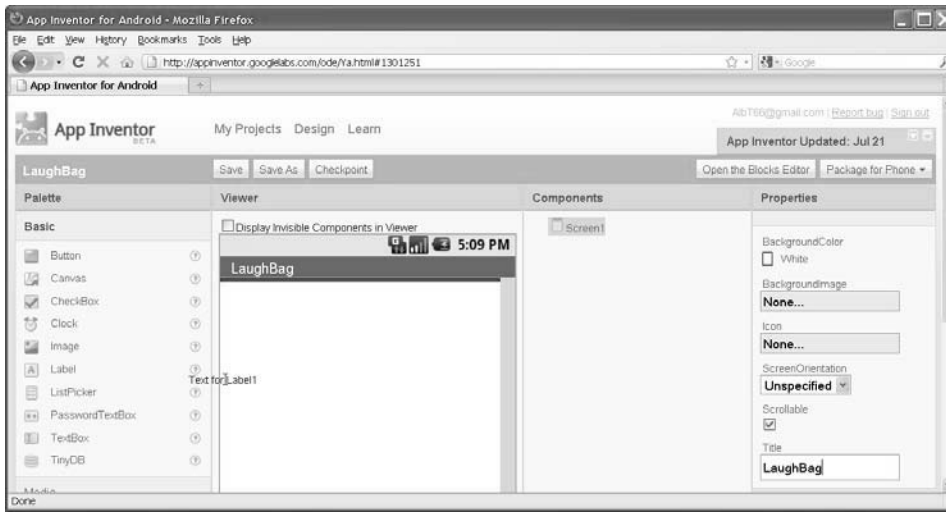


Figure 3.3 Drag a new label from the Palette to the Viewer

If you drop the label in the Viewer by releasing the mouse button, the label slots in automatically just below the title bar. Now you have created your first additional component in the LaughBag app: You have created an object of the component type "Label."

An Aside: The Distinction Between Component and Component Type

Let's remind ourselves once more of the distinction between the abstract component in the Palette and the specific object in the Viewer. If you drag a component from the Palette to the Viewer, you do not actually move this component, but rather create a specific object of the type of this component, the so-called *component object*. In theory, you can create an unlimited number of objects of the same component type and use them in your app. The distinction is similar to that between a class and an object in object-oriented programming.

Checking on Your Smartphone

If you have connected your smartphone, the new label with the default text "Text for Label1" will be visible on the screen as well. The same is true for all other visible expansions or changes you make in AI Designer, which will also become visible on the connected smartphone in debugging mode.

That is not the only reaction in the AI Designer interface, however. Once you create the label, it appears not only in the Viewer (and the connected smartphone), but also in the panels Components and Properties. As you can see in Figure 3.4, the new label appears as a component object in the Components panel, it has the name "Label1," and it is hierarchically subordinate to the central starting component "Screen1." The name and index number of "Label1" are assigned automatically by AI when a new component object is created. If you were to drag another label component into the Viewer, the newly created second label object would get the name "Label2," and so on. Feel free to try it. The second label will be positioned below the first in the Viewer, just as in the Components panel.

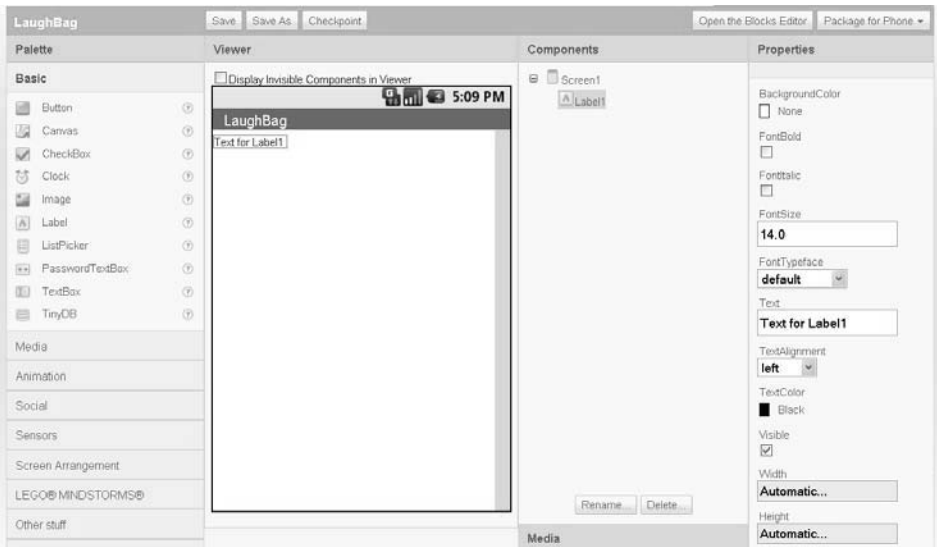


Figure 3.4 The label in the panels Viewer, Components, and Properties

At this point, you should delete all surplus labels, except for the first label. To do so, highlight them one by one by clicking on them in the Viewer or under Components and then clicking the Delete button in Components. Confirm that you really want to delete them by clicking OK. Before you continue, your workspace should once more look like that shown in Figure 3.4.

Assigning Component Names

Before we turn to the label's properties, we want to give this object a memorable name. If you assign memorable names, it becomes much easier to keep track of things when you work with the AI Blocks Editor later. If you are designing complex apps and are using many labels or other objects of the same component type that have names differing only by their index number ("Label1," "Label2," "Label3," and so on), it will be really hard to tell them all apart. It helps if the object has a name that reflects its function within the app.

> **An Aside: Assigning Names in Programming**
>
> In programming, it is common practice to assign descriptive names to objects, constants, functions, and variables, so as to make the rather abstract and cryptic program code clearer and easier to understand in case later editing becomes necessary. A good naming convention can also be an essential aspect of good and effective app development.

We want to change the name of the label from "Label1" to "Message." Click on "Label1" with the mouse to select it under Components, and then click on the Rename button. This opens a pop-up window where you can enter the "New name" and confirm it by clicking OK (see Figure 3.5). In the Components panel, the label is now listed under its new name "Message."

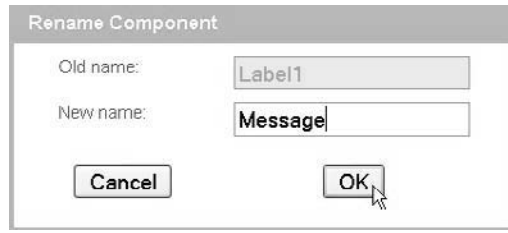


Figure 3.5 Assigning a memorable name to the new label

Setting Properties

To view and change the properties of a component, you first need to select the component by clicking on it in the Viewer or the Components panel. The selected object appears bordered in green in the Viewer and with a green background in Components. The properties of the selected component are shown in the Properties panel. Even a simple component such as the label has an impressive list of properties. For the text field, for example, you can specify or choose its Alignment, BackgroundColor, FontBold, FontItalic, FontSize, FontTypeface, and TextColor.

In addition, you can specify other properties, such as the label size, separately for the width and height of the text field. You can choose whether the label size should

automatically adapt to the text dimensions (Automatic) or the dimension of the parent object "Screen1" (Fill parent) or whether the label size will be specified explicitly in pixels (Pixel). You can also choose whether the label or the text it contains should be "Visible" (or not). You might wonder what would be the point of defining a text that is then not visible. As we have already mentioned, the component properties can be changed not just during development in AI Designer, but also later during the app's execution (runtime) on a smartphone, dynamically within the program logic defined by the block structures. It is possible to not show a message in a label when the app is first run, and then display it only when a certain event occurs—for example, tapping a button.

Finally, the label also has a text field ("Text"), where you can enter the actual label text. Here you should replace the default text "Text for Label1" with our message "Please press the bag!" If you check the new message in the Viewer or on the smartphone, it will not yet look very spectacular. Feel free to set a snazzy background color, change the font color, and perhaps increase the font size. The other settings can be left with their default values. In the Viewer, your app should now resemble the one shown in Figure 3.6.

Figure 3.6 Changed properties and new design for the label "Message"

Adding the Interactive Component "Button"

Let's move on to our second component, with which we want to complete the user interface of our LaughBag app. As mentioned earlier, we want a button with a picture on it to represent the interactive laugh bag.

Interactive Components

The "Button" component is an interactive element of the user interface, unlike the label. The button receives a user action (an event), such as tapping the button, and the app processes the event and reacts with a result (laughter). In addition to the button, AI provides many other interactive components—for example, selection elements such as check boxes and lists, and sensors that react to movements by the user.

The "Button" component can also be found in the component group "Basic" in the AI Designer Palette. Grab the "Button" component and drag it to the Viewer. Again, you can see the default setting for the "Button" component while you are dragging—namely, a button with the text "Text for Button1." While you are dragging the button to the Viewer, the screen area also shows a narrow horizontal blue bar (see Figure 3.7, below the label). This bar indicates where the selected component will be placed once you drop it in the Viewer. For example, if you were to drag the component "Button" above the label, the position bar would jump above the label. If you then drop the component, it would be placed between the title bar and the label. For our app, you should drag the component below the label to place the button below it. You can grab the objects in the Viewer below the title bar later and rearrange them at will.

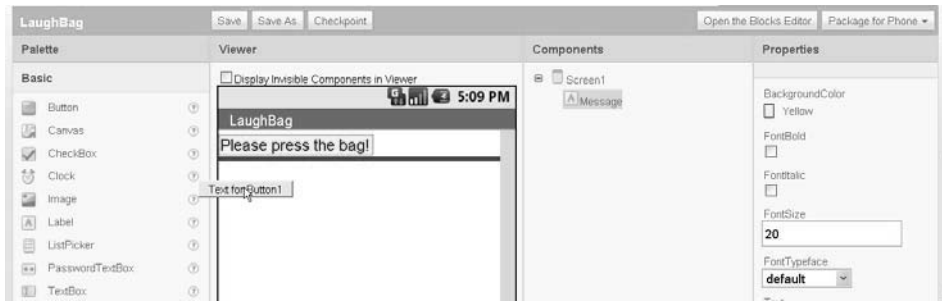


Figure 3.7 Dragging the "Button" component from the Palette to the Viewer

Just as before with the label, you now have created a second component object of the type "Button" in the Viewer. After you drop it, the button is automatically selected. The Viewer shows the graphic representation of the button with a green border, and you can also see the new button on your smartphone. Now we want to customize the button for use in our LaughBag app. First we will give it a memorable name. As shown in Figure 3.8, the button is also a child of the parent object "Screen1" in the Components panel, and again the button has a default name with an index number, "Button1." Just as before, you can click the Rename button and change the default name "Button1" to "LaughButton."

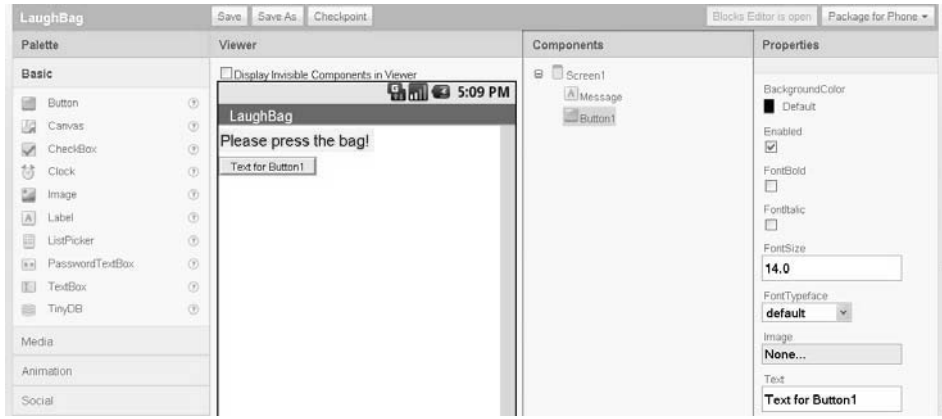


Figure 3.8 Button displayed in Viewer, Components, and Properties

In the next step, we again examine the properties of the LaughButton. They are mostly the same as those of the label. Once more, we have options for setting the text alignment, background color, bold and italic font, font size, type, and color. We can again set the button size (i.e., width and height) and decide whether to make it visible. In the text field "Text," you can change the text; let's replace the default text "Text for Button1" with "Press me!" Perhaps you would also like to edit the font and set it to bold and italics. If you wanted the button to have no text on it at all, you would need to leave the text field empty.

In addition to the familiar properties, this button has two new qualities. First, it contains an additional check box with the label "Enabled." Similar to the case with "Visible," you can use this check box to decide whether the button should be enabled when the app is run—that is, whether it should respond to being pressed. Of course, we do want the button representing the laugh bag to respond if the user taps it, so we leave the "Enabled" check box checked. The second new property is the "Image" field, whose name reminds us of our initial intention of adding an image to the button. At the moment, the field shows "None," indicating that the button does not yet have an image. Let's change that now.

Uploading and Integrating Media Files

As mentioned previously, we will design our interactive laugh bag as a button with an image of a bag on it. As the LaughButton is already positioned in the Viewer and, therefore, in the app, we now need simply choose an image for it. We need a suitable image file with the picture of a bag. If you happen to have a great photo of a pretty bag in electronic form, feel free to use it. But make sure you meet the file format requirements, so as to ensure the image file will be supported by AI and the Android devices running the app.

Image Formats

For more information on image file formats supported by Android, refer to the "Image Formats" section in Chapter 15, "Tips and Tools."

In the more likely case that you do not have a suitable image file available, you can use the image file `laughbag.jpg` located in the `/MEDIA` directory on the companion website for this book. Feel free to use it as part of this example app.

On the Companion Website: Media Files for All Example Apps in the /MEDIA directory

All audio, image, and video files used in our examples can be found on the companion website in the `/MEDIA` directory.

To place the image from the file `laughbag.jpg` onto the LaughButton, you first have to make it accessible for your development work by uploading it into the AI IDE (development environment) and the remote provider servers (Upload). Click in the LaughButton Properties on the "Image" field to open the selection list shown in Figure 3.9. As you have not yet uploaded any image files in your current app project, the list is still empty and "None" is available for selection. You need to find the desired image file on your local hard disk (previously downloaded from the companion website) by clicking the Add button. This opens the Upload File pop-up window (shown in Figure 3.9). Click the Search button, to access your local file directory—for example, via Windows Explorer. Go to the image file `laughbag.jpg` and select it by clicking the Open button. The file name and path then appear in the Upload File window, and you can confirm your choice with OK.

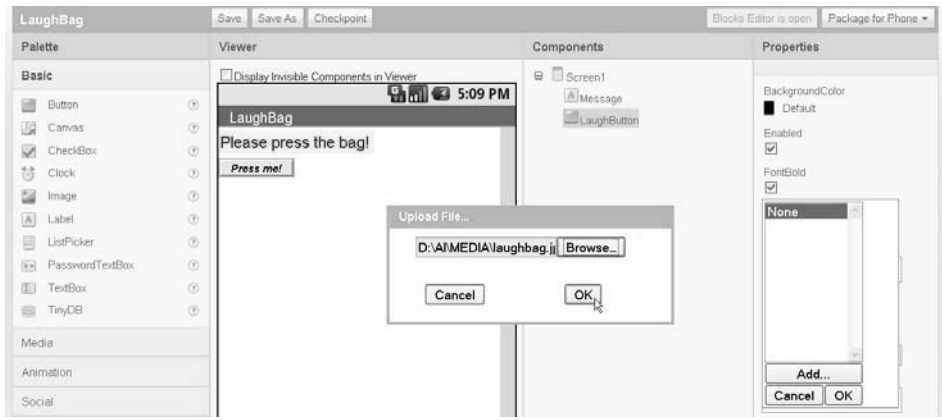


Figure 3.9 Uploading the image file laughbag.jpg to AI IDE

This starts the image file upload, indicated by the AI status message "Uploading laughbag.jpg to the App Inventor server." Once the upload has been completed, our laugh bag

in all its beauty appears in the Viewer (and on the connected smartphone), as shown in Figure 3.10. The name of the image file now assigned to the button is displayed in the LaughButton Properties in the "Image" field. The image file `laughbag.jpg` is still available for general use in the current LaughBag project, as indicated by the corresponding list item in the Media panel. If you decide, for example, that you want to assign the same image file to another button, the image file will appear in the selection list of the "Image" field concerned, so you do not need to upload the file again, but instead can reference it within the project to reuse it again and again. If you wanted to delete the image file from the AI IDE or the provider servers, you would have to click on it in the Media panel and select "Delete" in the pop-up menu. The file would be deleted and the LaughButton would be displayed without an image on it.

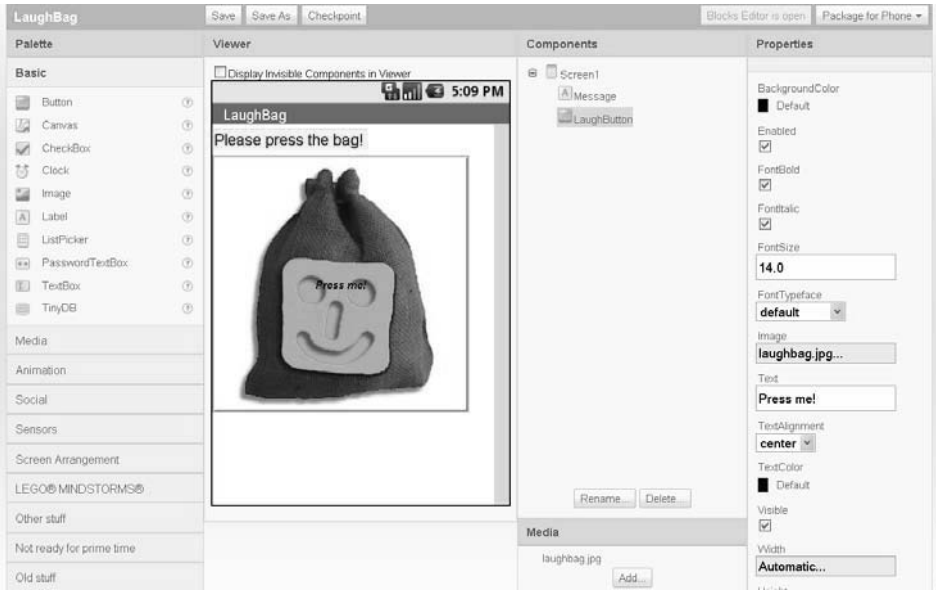


Figure 3.10 The LaughButton button with the image laughbag.jpg

Optimizing the App Design

This step basically concludes the optic design of our LaughBag app. We say "basically" because we still want to do a bit of fine-tuning. Take a look at the user interface in the Viewer and on the smartphone: The Message is all the way over on the left side of the screen, as is the LaughButton. It would look better and more professional if both elements were centered in the middle of the screen, not only in the static Viewer, but above all on the differently sized screens of the various Android smartphones. You can guess how we can optimize the optic design. Click on the label in AI and go to its Properties

to change the setting for the horizontal Width property to "Fill parent." Now the label width automatically matches the width of the parent object "Screen1" and consequently fits the screen width of the relevant smartphone. You can check this immediately in the Viewer and on your smartphone. The text is still in the left corner of the label, however. Center it by changing the Alignment property to "center."

Texturing

In computer graphics, an image that is applied to a 2D or 3D object is sometimes referred to as a *texture* and the process of doing so as *texturing*.

We now want to optimize the button in the same way. Select it and change the Width property to "Fill parent." As the underlying image has a smaller width than the now enlarged screen-wide button in the Viewer and on the smartphone, the image plus label is automatically centered on the screen, thanks to the default setting "center" for Alignment. The optically optimized user interface of our LaughBag app now looks neat and tidy on the smartphone, as you can see in Figure 3.11.

Figure 3.11 Optimized user interface on smartphone LG P500

If You Cannot See the Image

If the bag image should not be visible on your systems, there is no need to panic. Unfortunately, with the many different computer systems and Android smartphones around, it sometimes happens that the image is either displayed in the Viewer only or the smartphone only, or on neither. For example, the image is displayed on the smartphone LG P500 with Android 2.2 or the default emulator of AI with Android 2.1, but does not display on the HTC Tattoo with Android 1.6. The Google developers are aware of this problem and working to fix it; temporary solutions are available on the AI "Troubleshooting" website (see "I set the image property of a button to an image file, but nothing shows on the phone"):

`http://experimental.appinventor.mit.edu/learn/troubleshooting.html`

Even if an image still fails to display on your development systems at this stage, chances are that it will display correctly on your smartphone in the finished app later. Just be patient, and wait until we reach the later section where you learn how to load and run the app on your smartphone. At that point, you should have the pleasure of seeing your image. Until then, you can simply imagine the image in your development environment. As mentioned previously, improvisation and patience are virtues that are most appropriate during app development in general and in the AI beta phase on the highly dynamic Android operating system in particular.

Non-Visible Component "Sound"

Even if the optic design of our LaughBag app already looks rather convincing with just two components and one media file, we are still missing something important. Right—what good is having a laugh bag without laughter? We need to add another media file to our app, an audio file with some nice loud laughter. Once more you will find a sample file `laughter.wav` on the companion website in the directory `/MEDIA`. As with all media files, you have to make sure that the audio file's format is supported by AI or Android.

Audio Formats

For more information on the audio file formats supported by Android, please go to the section "Audio Formats" in Chapter 15, "Tips and Tools."

To insert the audio file into your app, you need to drag another component into the Viewer. The "Sound" component for playing audio files can be found in the "Media" component group. Open it by clicking on the group name in the Palette. Grab the "Sound" component and drag it into the Viewer as before. If you drop the component with the default name "Sound1" in the Viewer, it does not appear within the represented screen area, but rather below it. As a *non-visible component*, the sound object is not displayed in the visible area of the Viewer, but simply listed in the special area "Non-visible components"; see Figure 3.12 (the bottom edge of the Viewer).

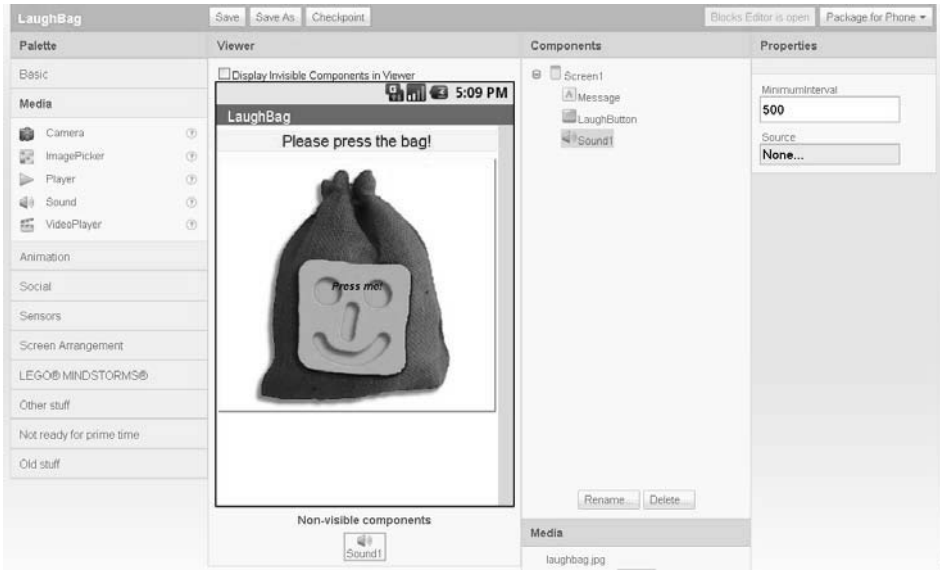


Figure 3.12 Listing the non-visible component "Sound" in the Viewer

Even though the component "Sound1" is naturally not displayed in the visible area of the Viewer (and not on the smartphone either), it does appear in the other panels just like any of the visible components. Under Components, it is subordinate to "Screen1"; under Properties, its specific properties are listed as usual, if the component object "Sound1" was selected and highlighted with a green border (see Figure 3.12). You can change the settings accordingly, just as described earlier in this chapter. Right now, you should change the name from "Sound1" to "Laughter" in Components.

The number of properties of the sound object "Laughter" in Properties is manageably small. You can use the MinimumInterval property to set the playback time of the audio file in milliseconds. If the length of the audio file is shorter than the interval, the file is played repeatedly. As a general rule, the playback time and interval time should match each other as closely as possible, unless you want to achieve a corresponding acoustic effect. We can fine-tune this feature later if necessary.

More interesting for our purpose is the second property of "Laughter" for uploading and integrating the audio file into the app by specifying the audio source in the field "Source," which currently—similar to the case for the image file—contains the value "None." In fact, you use exactly the same process for integrating any media file, whether it is an image or audio file. So, click on "Source" and you will see the list of available media files. This time you can readily see the file `laughbag.jpg`, as the selection list does not distinguish between the media types. Use the Add button to choose a file on your local hard disk and upload it into the AI IDE. Proceed in exactly the same way as when you uploaded the image file, but this time choose the audio file `laughter.wav`

in the directory /MEDIA of this book's companion website. If all went well and no errors occurred during the upload, you should now see the file laughter.wav in the Media panel and the "Source" field in Properties, as shown in Figure 3.13.

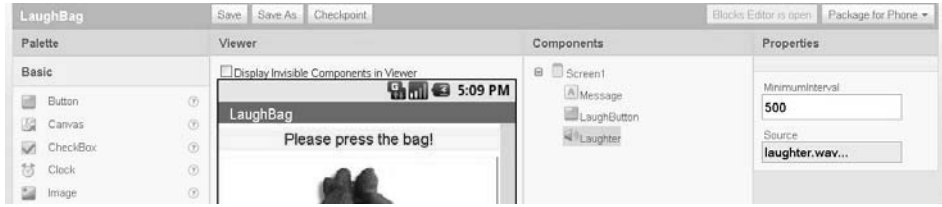


Figure 3.13 Successfully integrated audio file laughter.wav

Now the design of the user interface of our LaughBag app with components is complete. Of course, if you now try to press the LaughButton in the Viewer or even on your smartphone, nothing happens. The reason is that the objects of your app user interface are still unconnected and without functionality. To change this situation, we need to go to the second AI interface, the Blocks Editor.

Developing App Functionality

In this step, we will determine the tasks we want the component objects we placed in the AI Designer to have within our app. To do so, we will connect the active components (button and sound) with each other in the AI Blocks Editor and combine them into an interactive functionality (laughter when the button is pressed). This description of the development work we are about to undertake may sound a little pompous, but is the basic element of each and every app development process with AI, regardless of how complex and complicated, or simple and basic, the resulting app may be. If you fully understand this step, you will have mastered the basic principle of app development with AI and be ready to tackle bigger projects in the future.

This step takes us into a development area that comes close to classic app development with a programming language such as Java. Up to now, you have created the objects of the user interface in AI Designer and optically arranged the visible objects accordingly. Now you will access the functions of these objects (component-specific blocks) in the AI Blocks Editor and use them to design part or all of the app's functionality (block structure). In addition to the object functions, you can use general functions (generic blocks) for completing and developing the application logic and app functionality. Essentially you will start "programming" an app straight away, without using a programming language in the narrower sense. Do not feel put off by this process as a beginner; on the contrary, you should enjoy learning the principles of object-oriented programming as if in passing. Perhaps you would like to be able to develop apps in a programming language such as Java one day—in that case, your experience with AI will prove very useful.

The development process here will not get especially complicated. If you have not yet started the Blocks Editor, please do so now by clicking the button labeled "Open Blocks Editor" in the AI Designer. In the block selection area on the left, select the My Blocks tab. Now you can see the components with their component names that you dragged into the Viewer and renamed with a memorable name in the AI Designer (see Figure 3.14). For example, you can see the label "Message," the sound "Laughter," and the button "LaughButton." You can also see the screen component with its still unchanged default name "Screen1," plus a "My Definitions" area where you can enter your own definitions.

Figure 3.14 The component objects in the AI Blocks Editor under My Blocks

Just as described in Chapter 2 using the example of the "Screen1" component, you can now display the available function blocks for each of the custom component objects you created by clicking on the object name. Remember the "Visible" property in the Message label? In the preceding section on setting the properties of objects in AI Designer, we pointed out that the default settings there for app runtime can be dynamically changed with function blocks of the same name. In Figure 3.15, you can see the corresponding blocks `Message.Visible` for these dynamic changes.

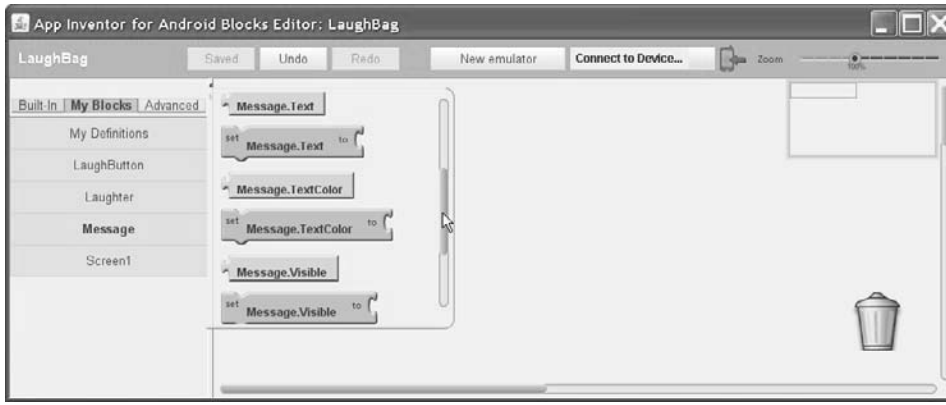


Figure 3.15 Properties of component object label "Message"

Create Interactive App Logic

For now, we want to leave the label text of Message in our LaughBag app unchanged and will not yet use any of these function blocks. Instead, we want to add an interactive function to the button LaughButton. We want to receive user input (i.e., tapping the LaughButton) and respond with a reaction by the app (i.e., playing the sound Laughter). We can express this function of the LaughButton in words as follows:

> When user presses LaughButton, play the sound Laughter!

As a beginner without programming knowledge, you may find it hard to believe, but this is already an instruction similar to programming, in the form of an informal *pseudocode*.

We will now implement this function in our LaughBag app to instruct the app to play the Laughter if the user presses the LaughBag. In programming, you generally talk about an event handler—a routine that is carried out only if a certain *event* occurs. In our example, pressing the button is the *event*, the instruction "If click, then [action]" is the *event handler*, and playing the sound is the *action*.

Events and Event-Driven Programs

The term *event* is an important keyword for a basic principle of object-oriented programming, *event control*. In contrast to the classic sequential program flow, an event-driven program waits for input or events of various kinds and reacts with an appropriate function. A program or an app can react to different kinds of events, such as haptic, text, acoustic, or other sensory input, as well as to input from other applications, such as phone calls, SMS messages, e-mails, or news from web services such as Twitter.

To express the function instruction described earlier in the visual developer language AI, we need just two component-specific blocks (see Figure 3.16). The first block forms

the frame of the event routine and contains the instruction: When the LaughButton receives a click-event, then do something. The second block forms the action and performs the task: Open the audio player and play back the audio file Laughter.

Figure 3.16 The two components LaughButton.Click and Laughter.Play

To create a continuous instruction sequence, the two individual blocks now have to be connected with each other; that is, the action has to become part of the event routine. That can happen only if the syntactic rules of our visual development language allow it. In our case, you can see this result immediately. The executing component `Laughter.Play` fits like a puzzle piece into the calling component `LaughButton.Click`. This results in the instruction sequence or the block structure shown in Figure 3.17.

Figure 3.17 The block structure for the instruction sequence of the pseudocode

That's it! By connecting these two blocks, you have sufficiently described the functionality of the LaughBag app. That step may seem trivial at first, but only because the visual development language AI hides the complexity behind the block functions. As an app developer using AI, you do not need to worry about the complicated process with which your Android app loads an audio file, plays it, and outputs it via the system loudspeakers. You do not need to write a program routine in which you supervise the touch screen area of the graphically represented button for touches and then combine this event with the audio player. Thanks to the high level of abstraction of the visual description language, you can fully concentrate on describing the functionality of your app and leave the program- and system-technical implementation almost entirely to AI and Android. Even this basic step can become quite challenging, however, as you will notice later during the other projects described in this book.

An Aside: The Power and Abstraction of the Visual Development Language AI

Of course, this convenience comes at a price in terms of the flexibility afforded for function design. The Java programmers among the Android developer population, for example, might argue that they would like to decide themselves which audio player they access and in which

way. It is important to recognize that *all* development languages, with their different levels of abstraction, have specific advantages and disadvantages. AI has a very clear advantage: You can very quickly and simply develop appealing apps on your own. Despite their occasional complaints about AI's limitations on design, the Java developers resort to using class libraries that offer prefabricated objects and functions for use in their program code.

App development is also usually subject to a cost–benefit relation. If you can develop the "same" app more quickly and easily with AI, then you should make use of this advantage. Conversely, if you have a specific requirement that cannot be implemented with even a creative use of the AI components, then you will have to accept the need to put in more effort and become familiar with Java programming. Before doing so, however, you should think twice and remember the saying "There are many ways to skin a cat": Perhaps there is a way in AI after all to implement the desired app functionality. AI is much more powerful and more flexible than this first example might suggest, and its functionality is constantly being expanded. Don't forget that AI is only in its infancy (i.e., in the beta phase), yet already offers an impressive range of functions that continues to grow exponentially. You will see for yourself later on what is already possible with AI, even if it is just the "tip of the iceberg."

Implementing Functional Block Structure

Now that we have developed the program logic of our app with the block structure described previously, we can specifically implement it within our LaughBag app. You first need to drag the blocks mentioned earlier, one after the other, into the AI Blocks Editor. Let's start with the interactive component object "LaughButton." Open the available blocks by clicking on the corresponding object name in the block selection on the left. This opens the selection menu shown in Figure 3.18, in which you can see the suitable instruction block `LaughButton.Click` at the top.

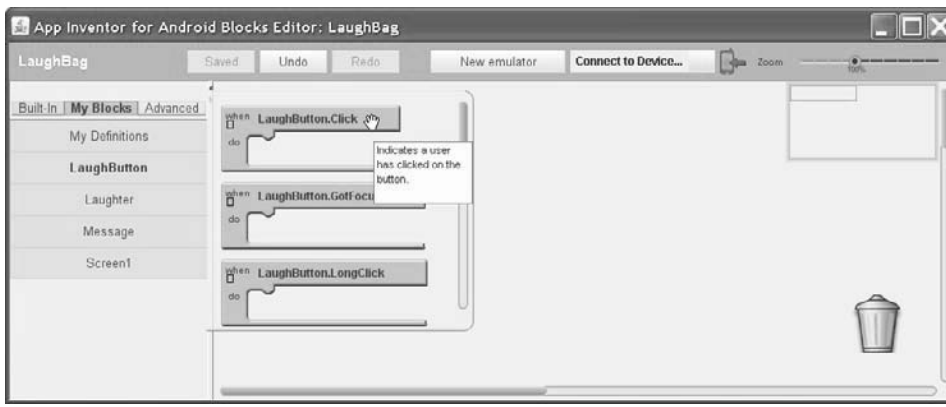


Figure 3.18 Selecting the function block LaughButton.Click in the AI Blocks Editor

If you touch this block with the mouse pointer, a brief description of its functions pops up. As soon as you grab the block with the mouse pointer, the selection menu becomes hidden and you can now drag the selected block into the Editor and drop it in the place you want. Then select the function block `Laughter.Play` in the same way. Open the block selection of "Laughter" and search for the right block—to facilitate the search, you can use the scroll bar on the right-hand side of the selection menu to scroll the selection up and down. In third position, you can see the function block `Laughter.Play` (see Figure 3.19); grab it and drag it into the Editor.

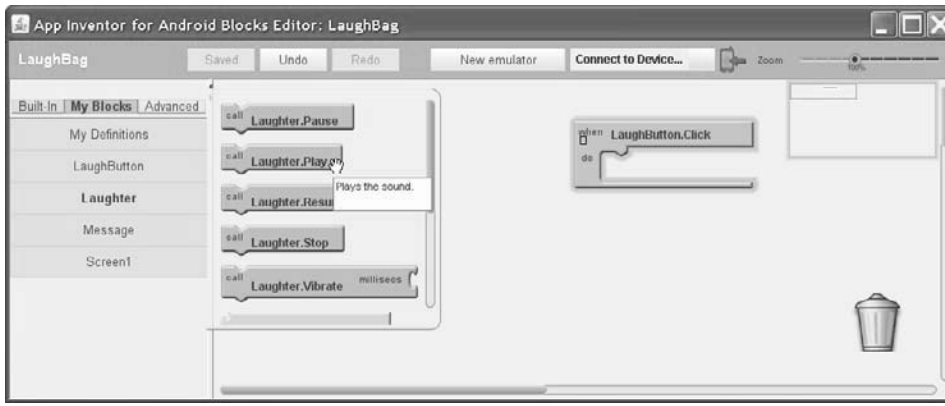


Figure 3.19 Adding the function block Laughter.Play to the LaughBag app

You can "temporarily" drop the function block `Laughter.Play` anywhere in the Editor and then drag it somewhere else later or directly connect it to the instruction block `LaughButton.Click` to form the desired block structure.

Color Coding of the Different Block Types

You have probably noticed that the blocks in the block selections have different colors. The color coding indicates the type of the relevant block. For example, all instruction blocks are green and the function blocks are purple. The different colors make it easier to keep track of things when you are developing apps with complex block structures.

To connect the two selected blocks, drag the block `Laughter.Play` to the correct "docking place" of the block `LaughButton.Click`, then drop it. If the two "puzzle pieces" were sufficiently close together (and provided the syntax is right), `Laughter.Play` will audibly click together with the instruction block `LaughButton.Click`, as shown in Figure 3.20.

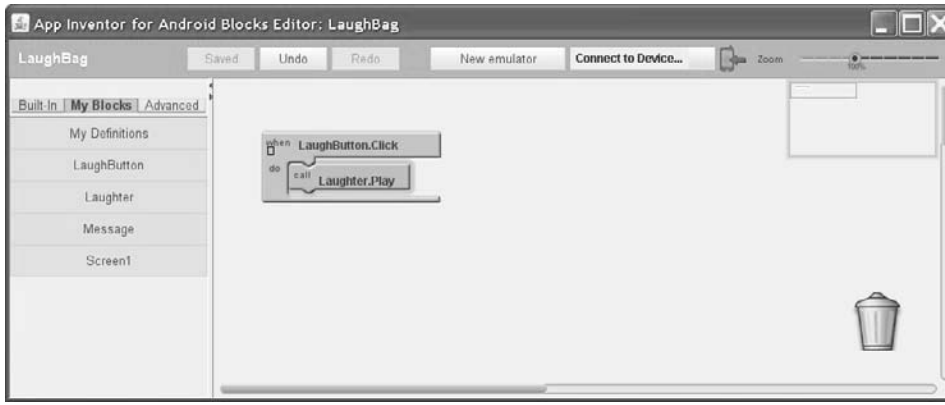


Figure 3.20 Finished implementation of the block structure in Editor

This completes the implementation of the functionality of our LaughBag app. Now you can check that it really does work, provided your smartphone (or even the emulator) is connected. Go to your smartphone and press the laugh bag; you should hear loud laughter every time you press this button.

Save Project Locally

As soon as you have finished developing your AI project, you should immediately save it. As described earlier in the context of the AI IDE, AI saves your project automatically on the provider's servers each time you close the AI IDE. This ensures that you always see the most recent version of your project when opening the AI IDE. You also have the option of saving the current project stage in AI Designer via three buttons—Save, Save As, and Checkpoint—on the provider's servers in the previously described variations. These saved versions of your project are available only online within your personal account in the AI IDE, however, and using them requires that the provider's servers be functioning without errors. If you want to use your AI projects under other accounts or make them available to third parties—for example, as a didactic model, for discussing possible solutions or as a basis for their own projects—then it makes sense to save the project on your local hard disk. You also reduce the risk of losing your projects when you maintain a backup copy in your own sphere of influence.

To save one or more projects together with all components and block structures in the AI Designer and Blocks Editor, you need to go to the project view of AI Designer by clicking the My Projects button. There you can mark the projects to be saved locally with a green check mark. For now, check the LaughBag project, as shown in Figure 3.21, and then click the More Actions button and select the menu item "Download Source" below it.

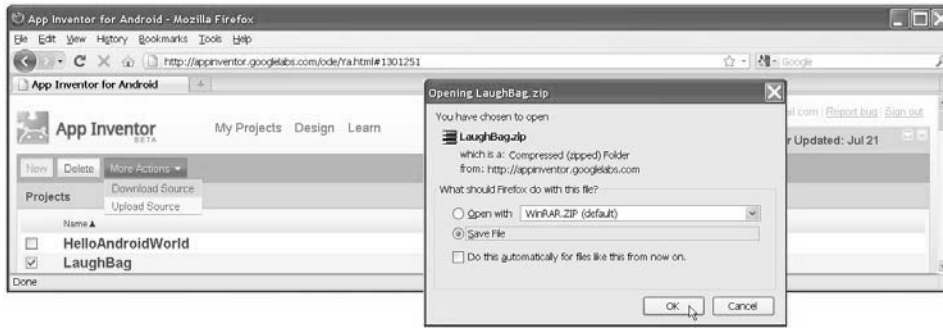


Figure 3.21 Downloading the marked project to hard disk

This series of steps opens the download window shown in Figure 3.21, which asks what you would like to do with the generated project file `LaughBag.zip`. Choose the option "Save File," click OK, and enter the desired location on your hard disk where you want to save the project. Now the project file is safe and sound on your local hard disk.

Downloading All of Your Projects at Once

At the end of 2011, an additional button labeled "Download All Projects" was added to the Google AI's My Projects overview menu. With it, all projects created by a user can be downloaded at once and saved locally as one large ZIP file. The resulting file, `all-projects.zip`, contains all projects in turn as individual ZIP files. After downloading and unzipping the collective file `all-projects.zip` to the local hard disk, you can then upload the individual project ZIP files one by one to the development environment of another AI provider (for example, MIT AI) and process them further as part of your project.

Project Files from This Book on the Companion Website in the /PROJECT Directory

You can find all project files from this book on the companion website in the `/PROJECT` directory (see the link in the Introduction). The website also includes the current file `LaughBag.zip`. You can upload all of the AI projects described in this book directly into the AI IDE after downloading them from the companion website, without having to input the interface components and block structures yourself. Nevertheless, you should not underestimate the benefit of the learning experience you achieve when you recreate the app step by step.

In case the block structures we develop later in this book become so big that they cannot be adequately printed in this book, be aware that you can access the companion website and upload the block structures from the project files into your own AI Blocks Editor. You can then study them in their entirety and develop them further for your own purposes if you wish.

If you should wish to edit the project later under a different account, you can upload it to the corresponding provider's server. Once more, uploading projects in AI happens

via the More Options button in AI Designer. Click on the option "Upload Source," and then click on "Choose File" and select the desired local project file with the extension `.zip` in the file manager. Click OK, as shown in Figure 3.22, to upload the project from your hard disk to the provider's servers and open it as the current project.

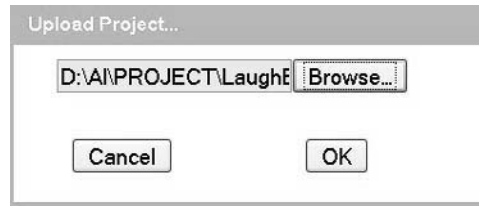


Figure 3.22 Uploading a locally saved project in AI

In this way, you can make your project files available to other AI developers as well, so that they can upload your project to their AI IDE for their own use. It is common practice for AI developers to share their projects and swap opinions on block structures, difficult problems, and common solutions. This cooperative attitude makes it also possible to offer tried and tested block structure functions to other users, which other developers can then use in their own projects as ready-made building blocks.

The file extension `.zip` indicates that the project file is not a single file, but rather a file archive. Surely you have a program installed on your computer that can extract zip files, such as WinRAR (`www.win-rar.com`). Double-click on the project file `LaughBag.zip` in your File Manager and take a look at the contents of the project archive.

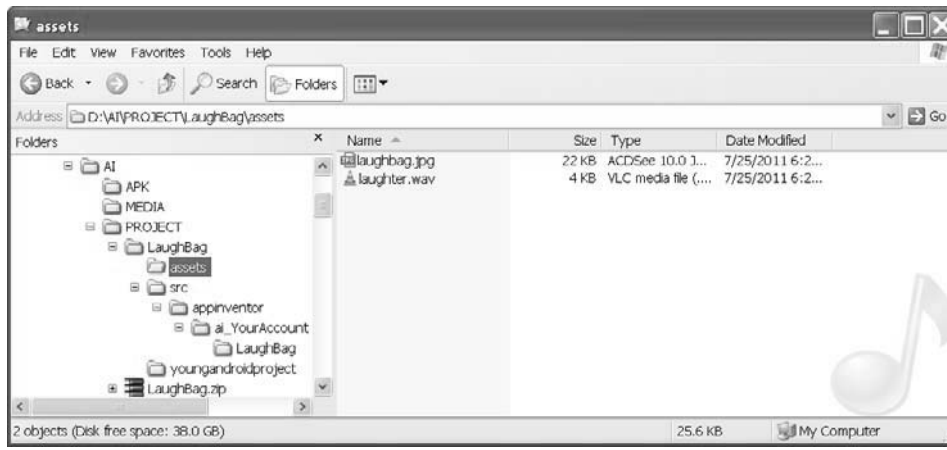


Figure 3.23 Directory and files in project archive LaughBag.zip

In Figure 3.23, you can see the directories and projects of the project archive `LaughBag.zip`. The actual source code with the system-specific description of the interface design and functionality of your app is hidden away in three files in the subdirectory `/src/appinventor/ai_YourAccount/LaughBag`, although on your computer the name of your actual Google account will appear instead of the placeholder `ai_YourAccount`. The directory `/assets` contains all of the media files used in your project and shown in Figure 3.23. You can't really do much with these individual files outside of the AI IDE, but the directory structure gives you an additional impression of how AI internally manages your app projects.

If There Is No Laughter

If the final function test of your app was unsuccessful, first check the obvious sources for the absence of sound. Have you set the volume on your connected smartphone high enough (or the volume of the computer, if you are using the emulator)? If yes, check the setting from the previous section on correctly integrating an audio file into the AI Blocks Editor or the app. Can you see the file name `Laughter.wav` in the button Source property, and is the file still located in the directory specified? Or are you using another sound file that may not fulfill the format requirements of AI or Android and, therefore, cannot be played because it is not supported? Even if you can't see an image yet (see the problem description in the "Optimizing the App Design" section), the sound should be audible if you press the button.

It may be small consolation in this situation, but you are not alone with your problem. We based our first app deliberately on the official beginner's project "HelloPurr" described on the AI online pages. This project uses the same components and blocks for its app in which you can press the image of a cat to hear a meow sound. If you are trying this app, you can assume that you would encounter the same problems as with our LaughBag app.

> **Analogous Beginners App on AI Website**
>
> If you want to compare projects, the analogous beginners project "HelloPurr" can be found at:
> `http://experimental.appinventor.mit.edu/learn/setup/hellopurr/hellopurrphonepart1.html`

Even if you do not want to test the project "HelloPurr" yourself right now, you can still use the troubleshooting tips if you are having any problems with our LaughBag app. Check the AI Forum or the "Troubleshooting" page if there is no sound.

> **AI Troubleshooting**
>
> This page contains "Working with Sounds and Images," which includes the entry "I set the source property of a Sound or Player component, but there's no sound when I tell the phone to play." This entry, which is found at the following address, provides even more help:
>
> `http://experimental.appinventor.mit.edu/learn/troubleshooting.html#ImagesSounds`

In the document "No Meow for Hello Purr" the Google AI Team also described some steps for problem solving. If you are still having problems, try going through the following steps in order and see if one of them helps:

1. Click the button "Connect to device" in the AI Blocks Editor with your smartphone connected and test the app again.
2. Unplug the USB cable from your smartphone and then plug it back in. Now click the button "Connect to device" and try the app again.
3. Close the AI Blocks Editor, and then start it again from AI Designer. Now click the button "Connect to device" and try the app again.
4. Delete the audio file `laughter.wav` from the AI Blocks Editor, by clicking on the file name in the Media panel and then selecting the "Delete" option. Now load the audio file into the project again as described earlier. Disconnect the smartphone, and then reconnect it to the computer and the AI Blocks Editor.
5. Reboot your smartphone by switching it off completely (not just changing to standby mode) and then restarting it. Now reconnect it to the AI Blocks Editor.
6. Try a different USB connection mode. For information on how to do this, refer to the section on setting up the Android device in Chapter 1. Connect your smartphone to the AI Blocks Editor in the different modes.

Even if none of these steps solves your problem, there is still a chance that you will both see the LaughBag image and hear the laughter sound when you later download the app to your smartphone and run it as an independent app. This advice applies, for example, to the smartphone HTC Tattoo, which we also used for testing the apps in this book. Just be patient and don't give up! The next few sections will show you how to download the LaughBag project as an independent app to your smartphone.

Creating and Installing the App

If you have taken a break at some point during the previous development steps of the LaughBag project and shut down your computer or disconnected your smartphone from the PC and the AI IDE, you may have noticed something. First, there is the reassuring fact that the most recent version of the app project is displayed when you restart AI Designer and Blocks Editor and reconnect your smartphone, even if you did not save it explicitly—you can thank the automatic saving function of AI for that benefit. Second, you may have noticed that the LaughBag app was nowhere to be found among the other apps on your smartphone. That is because our LaughBag app up to now has existed only within the corresponding project on the AI IDE or the provider's servers. Now we need to create it explicitly as an independent app and download it to the smartphone to make it a "proper" app. AI offers three alternative methods for doing this, which we describe in the following sections:

- Direct installation on a smartphone
- Online installation via a barcode
- Downloading of the APK file to the computer

All three approaches can be accessed in AI Designer by clicking the button labeled "Package for Phone" and selecting the relevant option from the pop-up menu.

Direct Installation on a Smartphone

Let's start with the most direct option of creating and downloading the LaughBag app to your smartphone. The key requirement is that your smartphone be correctly connected to the AI IDE via the Blocks Editor. Of course, the app project you want to turn into an app also has to be selected for editing or active in the AI Designer. To create the app and at the same time download it to your smartphone, click the Package for Phone button in AI Designer. In the pop-up menu, choose "Download to Connected Phone" (see Figure 3.24) to start the download process.

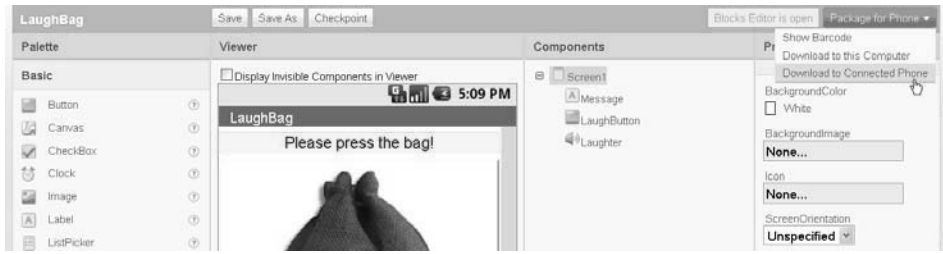


Figure 3.24 Selection of an app for direct installation on a connected smartphone

Now AI starts creating the app and downloads it to your connected smartphone. The progress of this process is indicated with the status messages "Packaging" and "Downloading to phone." After a certain processing time and many messages on data transmission in the browser's status line, the extra window shown in Figure 3.25, together with an acoustic signal, announces that the download and installation of the app on your smartphone were successful and are complete.

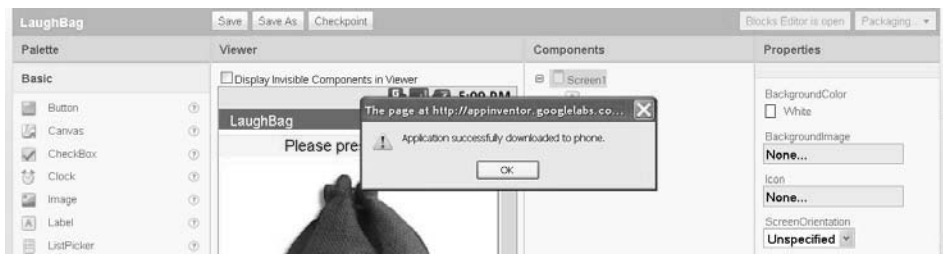


Figure 3.25 Message on successful app installation in AI Designer

Before you can now go looking for the app on your smartphone, you first need to close the still active AI development environment or display of the app project on your smartphone. You can do so with the options described earlier. Press the menu button on your smartphone; select the only menu item that appears, "Stop this application" (see Figure 3.26); and confirm your choice by selecting "Stop and exit" or, alternatively, by just unplugging the USB cable.

Figure 3.26 Closing the connection between the AI development environment and the smartphone

Now you can go to the app overview of your smartphone and search for the app with the title "LaughBag." Figure 3.27 shows an example of the app on the LG P500 smartphone, where we used the manufacturer-specific option of putting our LaughBag app into a separate category "AI Apps" (which will hold our future AI projects), just to make things clearer. If you now select the "LaughBag" app, it will start just like any other app on your smartphone. You can confirm that the app really does run independently of the AI IDE by noting that the USB connection symbol is absent in Figure 3.27 on the left-hand side of the top status line, whereas the connection icon is present in Figure 3.26.

Figure 3.27 The independent LaughBag app on the smartphone

The icon for the LaughBag app shown in Figure 3.27 is the default icon for all apps created with AI. If you would like to replace the small default icon depicting two little androids with your own custom app that matches the theme of your app, we will show you how in the next section.

A Matching Icon for Your App

Normally it would be sufficient to mention this subject in one or two sentences in the context of interface design with AI Designer—from today's point of view. Until the AI update of November 10, 2010, it was not officially possible to replace the default AI app icon with a custom icon. Many heated debates focused on the possibilities of retrospectively unpacking the project files, editing them, and adding another icon in the most adventurous ways. After all, how could people create more or less professional apps, and in some cases market them, if the app could not even have a suitable and appealing icon, just like every other app?

Against this backdrop, one of the early AI beta users had requested the inclusion of this feature by the Google developers. Feature requests are generally submitted via the "Issues List" introduced by the Google developers to enable them to more quickly and clearly structure and edit the numerous requests for improvement and bug reports submitted by AI users in the beta phase. The desire for the option of assigning custom app icons, for example, is listed as Issue 43 under the title "Add the ability to change the icon

of apps." The numbers under which the issues are listed (IDs) do not indicate their order of priority, but simply the order in which they are addressed.

Bug Reports and Feature Requests in the "Issues List"

The Google developer team provides the "Issues List" as a collection point for all open technical matters regarding the AI beta software:

`http://code.google.com/p/app-inventor-for-android/wiki/ReportingBugs`

Beta users are encouraged to report any AI bugs in a standardized form and to submit feature requests for AI using this list. Before starting a new issue, the AI beta user should check that the topic has not already been raised by another user and perhaps been discussed and resolved in the Google Group. The topics raised here should be of general interest and not concern individual problems, such as questions on the setup of your own smartphone—that is what the AI Online Documentation and the Google Group are for. The more focused the "Issues List" requests, the more readily the Google developers can concentrate on the important topics and continue developing AI, without getting sidetracked with minor issues. For that reason, each issue is evaluated by the Google developers, processed in various stages, and its status documented as follows:

Open issues

New	Issue has not yet been reviewed or queued
Investigating	Further information is required
Noted	Issue has been noted but not yet been accepted for work queue
Accepted	Issue has been accepted and will be worked on soon
Started	Work on this issue has begun
Testing	Issue is resolved and being tested to be published in the next new AI release

Closed issues

Fixed	Issue has been resolved
Invalid	This was not a valid issue report
Duplicate	This report duplicates another issue report
Won't fix	The Google team is not working on this issue
Forum	Refer this issue to the Google Group

Issue 43 now has the status "fixed." If you are curious and want to read up on the suggestions other AI beta users came up with in the past in connection with this feature, however, you could set the filter to "All issues" and search for "43" or "icon" on the "Issues Search" page.

Searching Issues and Checking Status

You can search open issues with various filter options and submit new issues via the "Advanced Search" page:

```
http://code.google.com/p/app-inventor-for-android/issues/
advsearch
```

You can find Issue 43 directly at this page:

```
http://code.google.com/p/app-inventor-for-android/issues/
detail?id=43
```

Since November 10, 2010, Issue 43 has had the status "Fixed" (see Figure 3.28). As a general rule, fixed issues are announced as new features in the next official update of App Inventor in the AI Forum. Given this fact, you do not need to search the "Issues List" regularly to find out about new features.

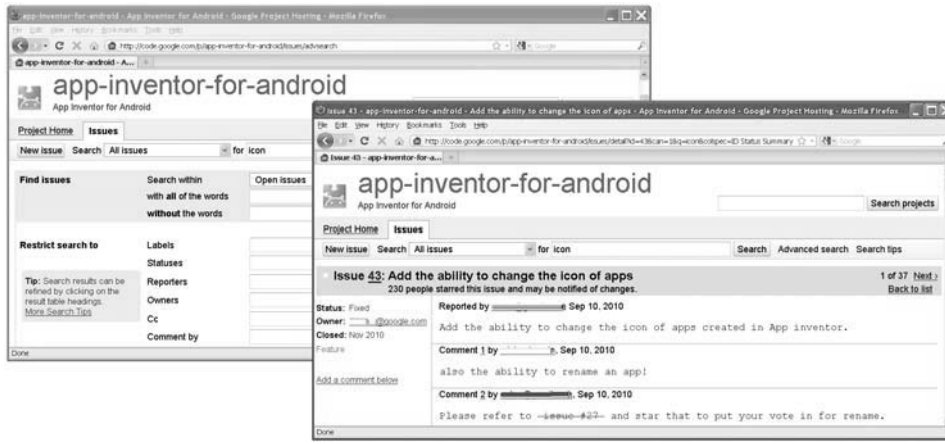


Figure 3.28 Status of Issue 43, "Change the icon of apps"

Now that Issue 43 is fixed, you can add a custom icon to your app really easily. Of course, you still have to keep your masterpiece within the specified format. Apart from guidelines on file formats, a detailed set of instructions in the form of "Icon Design Guidelines" applies to all developers of Android apps.

"Icon Design Guidelines"

A very detailed description of the requirements for format and design of icons for Android apps can be found in the Android Developers Forum:

```
http://developer.android.com/guide/practices/ui_guidelines/
icon_design.html
```

The "Icon Design Guidelines" contain not only templates and examples, but also recommendations for resolutions (in pixels) for different icon types such as a launcher icon, menu icon, and so on. Try to follow these guidelines when designing your app icons to give them a professional look. You can even use the templates provided in professional graphic programs such as Adobe Photoshop and Adobe Illustrator. Even if you are not a graphic designer, it can make sense to provide your images in the correct format so as to optimize their representation on the smartphone. Almost every graphics program offers the option of cropping your images and reducing them to a specified pixel size. Thus, if you immediately reduce the image you want to use as icon to the recommended size of 48 × 48 pixels, you reduce both the memory requirements and the risk of distorting the icon on the smartphone through automatic display adjustment in Android. For the file formats, the same requirements as specified in Chapter 15, "Tips and Tools," also apply.

Designing a Custom Icon with a Graphics Program

For our LaughBag app, it makes sense to use the existing laugh bag image for the icon as well. We want the icon to stand out a bit from the mass of other icons, however, and the bag should have a greater contrast to the white background.

Using the graphics program Corel Paint Shop Pro, shown in Figure 3.29, we edited the image from left (the original image) to right (the edited result). In the first step, we selected the outline of the bag with the "Magic Wand" and colored the background green using a "Gradient." To the resulting image we then added the optical appearance of a button with the menu sequence Effects > 3D Effects > Button. We then made the button round with Effects > Geometric Effects > Circle, and selected the background again with the "Magic Wand" to color it red.

Figure 3.29 Designing the default icon using the laugh bag image

We chose to use the graphics file format PNG (*Portable Network Graphics*), which allows us to define transparent areas, enabling us to achieve smoothly rounded transitions between the originally rectangular or square icon and the screen background. In Corel Paint Shop Pro, we used the menu sequence File > Export > PNG Optimizer to select the Red color values (RGB 255/0/0) as the transparent image area, so that only the round button shown in Figure 3.29 on the right remains visible. We then reduced the image from the original 288 × 288 pixels to the recommended default icon size of 48 × 48 pixels via the menu items Image > Resize, and saved it as `laughbag_icon.png`.

If you do not want to design your own icons in a graphics program, you can use special programs or web services for creating buttons. The website "Glassy Buttons," for example, offers many options for creating and downloading custom buttons or icons in any sizes.

> **Website "Glassy Buttons" for Creating Custom Icons**
>
> The free button generator on the "Glassy Buttons" website (`http://www.netdenizen.com/buttonmill/glassy.php`) lets you create attractive buttons and icons with gloss effects. You can choose from many settings to customize the size, gradients, text labels, and many other characteristics of images, and upload images as textures. The finished button can then be downloaded in the file format PNG and JPG (packaged as a ZIP file) and used directly as icon subject to the conditions of use.

Assigning an icon to your app is just as simple as integrating the LaughBag image into our project and follows the same process. We could have done this in step 2 of designing the user interface described earlier, but then you would not have seen the default icon. With app development, it is quite common to skip back and forth between development steps and to correct, adapt, or expand work done in previous steps. To change the icon, please reconnect your smartphone to your computer and the AI IDE by plugging in the USB cable and clicking the button labeled "Connect to phone." Then go to AI Designer, and select the component object "Screen1." In its properties, you can see the property Icon in last place. As seen earlier with the two media files, its (default) value is "None." Now load a suitable image file for the default icon into your LaughBag project by clicking on "None" and choosing the image file on your computer via the Add button in the pop-up menu. Of course, you can once again find an example file on the companion website for the book in the directory `/MEDIA`—namely, the just-created image `laughbag_icon.png` (shown in Figure 3.29). When you select it, the image name once again appears in the Media and Properties panel, but the image itself is not displayed (see Figure 3.30).

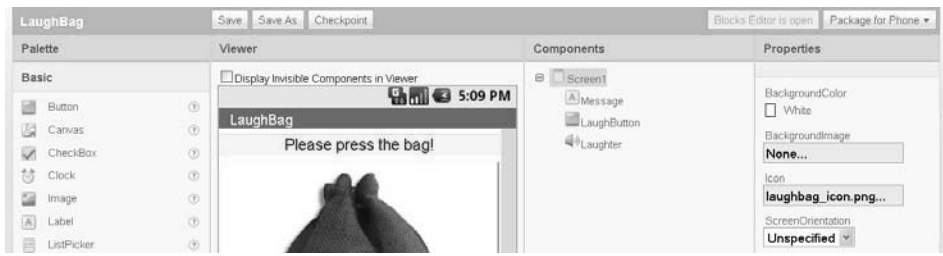


Figure 3.30 Adding an image file for the default icon of the LaughBag app

At this point, you have added your own default icon for the LaughBag project. Now you just have to download the app once again to your smartphone and install it. Remember to delete or deinstall the old LaughBag app from your smartphone before you install the new version with the default icon.

Deleting/Uninstalling AI Apps on Your Smartphone

Proceed in the same way as when deleting or uninstalling other apps. Drag the default icon of the app you created with AI into the waste basket, or select Settings > Applications > Manage > LaughBag > Uninstall, as shown in Figure 3.31.

Figure 3.31 Uninstalling the LaughBag app with the default icon

In the uninstall dialog shown in Figure 3.31, the file size shown for our LaughBag app is an impressive 4.10MB. This is the file size of the *installed* app on the smartphone, which is usually much larger than the app installation file you downloaded earlier.

Once the old app with the default icon is deleted, you can install the edited LaughBag app. Proceed as described earlier in the section on direct installation on the smartphone. After you have successfully completed the installation, the LaughBag app appears in your smartphone's app overview with the custom default icon shown in Figure 3.32.

Figure 3.32 The LaughBag app with custom default icon on the smartphone

You can now use your LaughBag app just like any other app on your smartphone. In addition to the app icon in the app overview (Application menu), you can create a widget (link icon) for it directly on the home screen or the other panels.

Correct Reproduction of Image and Sound in the Independent App

If reproducing the laugh bag image or playing back the sound has been causing problems up to now, you may now both see the image and hear the laughter sound in the independent app. This is the case, for example, with the HTC Tattoo with Android 1.6: Whereas neither the image nor the sound works during the app development process, the independent app works perfectly on the smartphone. If you experience the same issue with your development environment and smartphone, then you can assume that in your future development work you will not be able to check and use certain components directly in the AI IDE, but that they will work correctly on the smartphone later as independent apps. This makes developing a bit more awkward, but at least not impossible.

Assigning custom icons for apps developed with AI is, of course, possible at stages other than during the direct installation. Assignment of the default icon occurs independently of the installation process. By assigning an image in the default component "Screen1," the app automatically gets the corresponding default icon.

Online Installation via a Barcode

For the second method for downloading and installing our LaughBag app, you do not need a USB cable connection between your computer or the AI IDE and your smartphone. To demonstrate this the approach, please unplug your smartphone's USB cable from the computer now. If you have already installed the LaughBag app on your smartphone in the step described in the previous section, please delete it so we can be sure that you download the app without the USB cable this time. Keep the AI Blocks Editor and AI Designer open, as you will need the block structures to generate your LaughBag app.

To use the barcode approach, you need an Internet data connection on your smartphone (and on your computer, of course), either via WLAN and your WLAN router or directly via the mobile data net of your cellphone provider in form of GPRS (*General Packet Radio Service,* which offers up to a 172 Kb/s download rate with GSM channel bundling), EDGE (*Enhanced Data Rates for GSM Evolution,* which offers up to a 473 Kb/s download rate with GSM channel bundling), UMTS (*Universal Mobile Telecommunications System,* which offers a 384 Kb/s download rate in the 3G net), or the speedy HSDPA (*High-Speed Downlink Packet Access,* which offers up to a 7.2 Mb/s download rate in the 3.5G net). You also need to have an app for reading barcodes on your smartphone.

Barcodes, QR Codes, and Barcode Scanners

A barcode is generally a method of encoding data using bars (lines) that can be read and processed by optic reading devices. You are certainly familiar with the 1D codes used on almost all product packaging, such as in the supermarket. 2D codes are becoming increasingly more common on the Internet in form of QR codes (quick response codes), which are used to encode web addresses and the like with a *barcode generator*. The encoded information can then be read and decoded by a smartphone, allowing direct access to the website. To use QR codes, you first need to install a *barcode scanner* as an app on your smartphone; such an app can check the filmed or photographed camera image on the smartphone for a QR code and then process the code.

To use the AI option for installing your app online, you need to have a barcode or QR code scanner installed on your smartphone. If you do not yet have one, go to the Android Market and search for "barcode"; then choose one of the many free scanners to install on your device. We used the barcode scanner ixMAT by ZXing, but Google Goggles also works well for QR codes.

(Continues)

Figure 3.33 QR codes for downloading ixMAT (on the left) and Google Goggles (on the right)

Of course, you cannot do anything with the two QR codes shown in Figure 3.33 unless you already have a barcode scanner installed on your smartphone. If you do have one, these two QR codes will give you direct access to the appropriate app from the Android Market. We provide a quick description of how to use the barcode scanner for downloading AI apps using the ixMAT app as an example.

If you have fulfilled the requirements for a mobile Internet connection and a barcode scanner installed on your smartphone, you can start with the online installation of the LaughBag app. Open the menu in AI Designer by clicking the Package for Phone button and this time choose the option "Show Barcode," as shown in Figure 3.34.

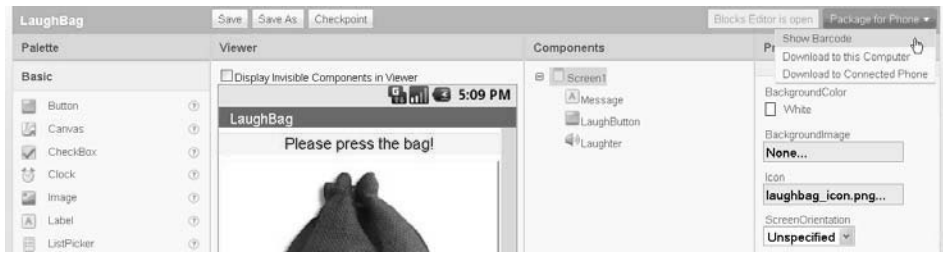


Figure 3.34 Selection for online installation of the LaughBag app on the smartphone

After displaying the usual status messages "Saving" and "Packaging," AI opens the window titled "Barcode link for LaughBag" and shows a QR code (see Figure 3.35). This QR code encodes the download link under which AI offers the generated LaughBag app for download. Unlike the case with the direct download described in the previous section, AI has now saved your app on a Google server from which you can download the app.

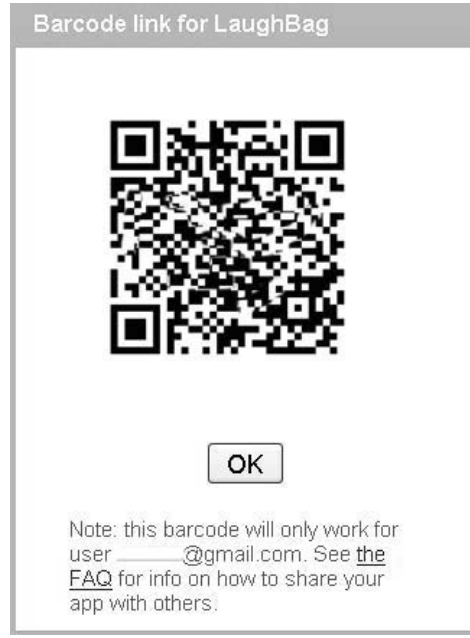


Figure 3.35 QR code or barcode containing a download link for the LaughBag app

Note that the QR code shown in Figure 3.35 was deliberately distorted before being printed in this book, so that your barcode scanner can no longer read it. Instead, you must scan your *own* barcode generated under your personal account and now displayed by AI. You can also take a "photo" of your QR code (under Windows with the keyboard shortcut Alt + Print) or save it for later downloads. You could also pass the QR code on to third parties, who could use it to download your LaughBag app to their own smartphone and install it; however, the requirement in this case would be that you make the login data for your own Google account (i.e., the account in which you are working with AI) available to others. For security reasons, this practice is not recommended, so this download option is not a suitable choice for making your future apps available to the public. We will discuss possible alternatives in the next few sections.

To get the app onto your smartphone via the QR code shown in AI, you need to scan the barcode with the barcode scanner installed on your smartphone. Start the scanner app and hold your smartphone's camera at an appropriate distance in front of your computer screen (or a printout). If you can see the barcode clearly and in its entirety in the view finder, follow the instructions of the scanner app. In case of Google Goggles, for example, you need to take a photo of the QR code by pressing the appropriate keys

before the image analysis is started. In contrast, ixMAT processes the running camera view automatically and quickly indicates the recognition and decoding of the QR code with a signal tone (see Figure 3.36). If your smartphone has a low camera resolution without autofocus, try enlarging the QR code a little to make it scan correctly.

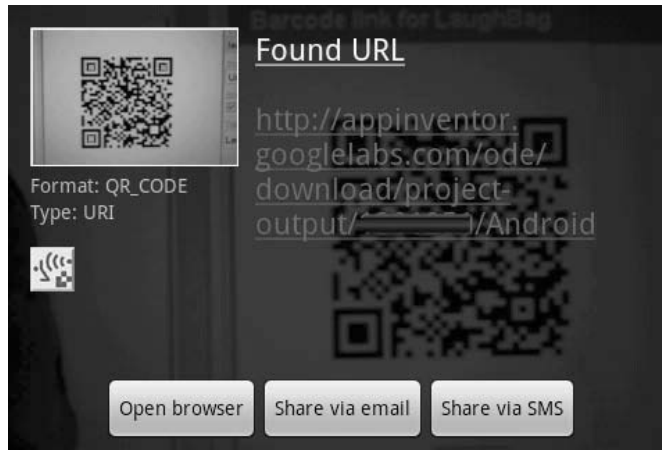


Figure 3.36 Decoding the QR code with the barcode scanner ixMAT

If the QR code was successfully recognized, the decoded download link is displayed in the scanner; Figure 3.36 shows the display in ixMAT. Depending on the feature range of the barcode scanner used, it will offer different options on how to proceed with the link. With ixMAT, you can choose to click "Share via email" or "Share via SMS" to send the link to other e-mail addresses or phone numbers, if appropriate (see the earlier cautionary comments). For our purpose, we want to use the decoded web link to "Open browser" so as to display the download page for the LaughBag app. Your smartphone then opens the web browser and takes you to the HTTPS-secured login page for your Google account, as shown in Figure 3.37 on the left. Enter the same login data as for your login to the AI development environment, and then press the Sign In button.

Figure 3.37 Log in to the Google account and download the LaughBag app

Once you have successfully logged in, the download of the LaughBag app to your smartphone commences. The progress of the download is indicated by the corresponding download icon in the smartphone's status line. If you drag down the status line, you can see a confirmation after successful download telling you that the application file `LaughBag.apk` has finished loading, as shown in Figure 3.37 on the right under "Notifications."

APK Files

An Android file has an extension of `.apk`. The file extension APK stands for *Android Package*. As the term "package" indicates, the app not only comprises a single file, but also forms an archive of several files. This organization explains why the AI status message says "packaging" during the download process: The generated app files are combined together into a "package" before being downloaded. The archive format resembles the Java archive format JAR (Java Archive) and is also used for app development in Java. We will briefly discuss the contents of the APK archive in the following section on downloading the APK file.

Now click on the file `LaughBag.apk` in the "Notifications" section, or follow the other necessary steps for installing the downloaded application file `LaughBag.apk` on your smartphone.

Allow "Unknown Sources"

One requirement for the installation of the downloaded APK file is that you have checked "Unknown sources" in Settings > Applications with a green check mark, to allow app installation of sources from outside of the Android Market. As you have downloaded your LaughBag app from your Google account on a Google server, this setting must be enabled.

Installation takes place via the steps shown in Figure 3.38, just as with other apps. Once you have selected the file `LaughBag.apk`, you are notified of the app's access rights before you start the actual installation by clicking on the Install button. After a brief installation period, you can "Open" the LaughBag app directly from the installation confirmation by clicking on the corresponding button, or go to the application overview of your smartphone to start it.

Figure 3.38 Installation of the downloaded LaughBag app on the smartphone

This completes the online installation of the LaughBag app using the second approach (barcode). But wait, there's more: AI offers a third alternative of downloading and installing your app.

Downloading an APK File

As third alternative, you can download your LaughBag app as the file `LaughBag.apk` to your computer and get it to your smartphone via this "detour" to install it. Considering the ease of the other two options, this approach seems rather awkward, but it is the easiest option for passing your app on to third parties without having to go via the Android Market or giving others access to your Google account for the barcode installation. It enables you to offer the file `LaughBag.apk` online on your web server for downloading or to send the file to others via e-mail so that they can also download your LaughBag app to their smartphones.

To try out this method, you again need to uninstall the LaughBag app from your smartphone as described earlier. At first you will not need to connect the smartphone to your computer for downloading the app, but later it will need to be connected when you copy the APK file from the computer to your smartphone. You do not need a mobile

Internet connection for this installation method, as all data required for the app are available locally on your computer and are copied over to your smartphone via USB. Start the download of the file `LaughBag.apk` to your computer by clicking the Package for Phone button in AI Designer, but this time choose the option "Download to this Computer" (see Figure 3.39).

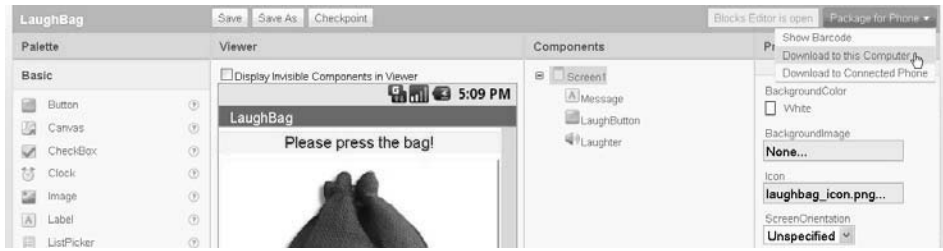


Figure 3.39 Option to download the file LaughBag.apk to the computer

After displaying the usual status messages "Saving" and "Packaging," the system window shown in Figure 3.40 pops up to ask if you want to open or save the file. Choose the option "Save file," click OK, and then select the target directory on your local hard drive to save the file `LaughBag.apk`.

Figure 3.40 Pop-up window for downloading the file LaughBag.apk

After the download is complete, you will find the file `LaughBag.apk` in your specified target directory. The companion website also contains this file, in the directory `/APK`. In contrast to the app's installation size of 4.10MB as shown in Figure 3.31, the downloaded APK file is only about 1.15MB in size.

We now want to copy this APK file from the computer to the smartphone. First you need to ensure the smartphone is connected to the computer via USB. Enable USB

debugging and turn on USB storage. To check this status, you can open the file manager on your computer and determine whether your smartphone (or, more specifically, its SD card) is registered or listed as an additional hard drive in addition to the other drives (e.g., `C:/`).

An SD Card Is Obligatory

Having an SD card on your smartphone is a requirement for working with AI—although, of course, it is valuable for many other reasons. The other installation methods also download the apps created with AI to the SD card of your smartphone. The relevant APK files are then usually placed in the SD card's `/downloads` directory. The media files of the installed AI app, referred to as *assets*, can be found in the directory `/AppInventor/assets`. Take a look at your own directories: You may even still see the files `laughter.wav`, `laughbag.jpg`, and `laughbag_icon.png` used in the previous installations. Not all traces vanish after you uninstall a program.

If you cannot see your smartphone in the file manager, you can enable USB storage explicitly on your smartphone by pulling down the status bar, clicking on "USB connection," clicking the button "Turn on USB storage," and confirming your choice by clicking OK (see Figure 3.41 using the example of LG P500 from left to right). Now your smartphone—or, more correctly, its SD card—should appear as a separate drive in the file manager.

Figure 3.41 Enable the USB connection to copy the APK file

Once your smartphone is connected to the computer as a USB storage device, you can easily copy the downloaded file `LaughBag.apk` to the SD card of your smartphone (see Figure 3.42). Go to the download directory in the file manager (for example, `E:/AI/APK`), copy the APK file (Copy in Windows or use the keyboard shortcut Ctrl + C), go to the smartphone's desired target directory in the file manager (such as `F:/downloads`), and paste the copied file there (Paste in Windows or use the keyboard shortcut Ctrl + V). Remember this directory, as you will later need to access it on your smartphone and select the APK file to install it.

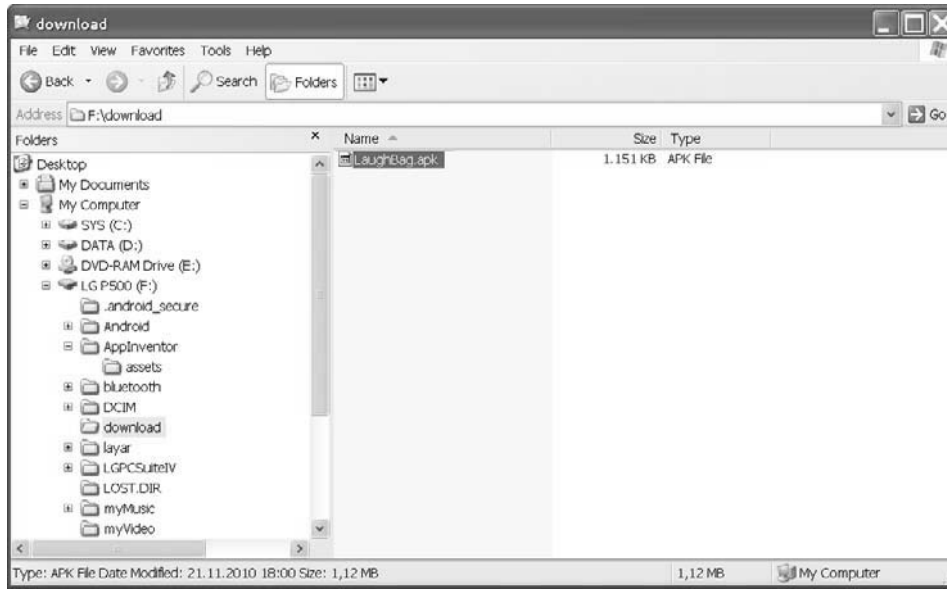


Figure 3.42 Copying the file LaughBag.apk to the SD card of an LG P500 smartphone

Once you have copied the file `LaughBag.apk` onto the SD card of your smartphone, you can disconnect the smartphone from the computer. Close the USB connection by clicking on the button "Turn off USB storage" (see Figure 3.41 on the right), or unmount the drive from your computer just like any other USB device (in Windows, use the icon "Safely remove hardware," which is found in the status bar).

Now you have the APK file on your smartphone, but you can neither start it nor see it in the app overview. Before the app can appear in the application overview, you must first install it. The APK file is merely the installation file. If you tap on an APK file on your smartphone, Android automatically installs the app archived within that file. But how can you find the file `LaughBag.apk`, which you copied into the `/downloads` directory on your smartphone's SD card using the file manager? To do so, you need an app that lets you access the directories and files on your smartphone and its SD card in the same way as the file manager does on your computer.

File Manager for Android Smartphones

To access the files and directories of your smartphone and its SD card, you need an additional app, similar to the file manager on a computer. The current Android smartphones do not usually include a mobile file manager, so you need to obtain one from the Android Market—for example, by searching for "Explorer." The free version AndExplorer, by Lysesoft, is a good choice.

If you have an Android Explorer, such as the AndExplorer program created by Lysesoft, installed on your smartphone, you can start it now to find the copied file `LaughBag.apk` on your SD card. After starting AndExplorer and pressing the SDCard button, you should see the same directories (shown in Figure 3.43 on the left) as were visible in the file manager on the computer (shown in Figure 3.42). Now you can simply press the directory name `/download` to go to that directory; there you will find the copied file `LaughBag.apk`. Click on the file name to start the installation process, as shown in on the right-hand side of Figure 3.43.

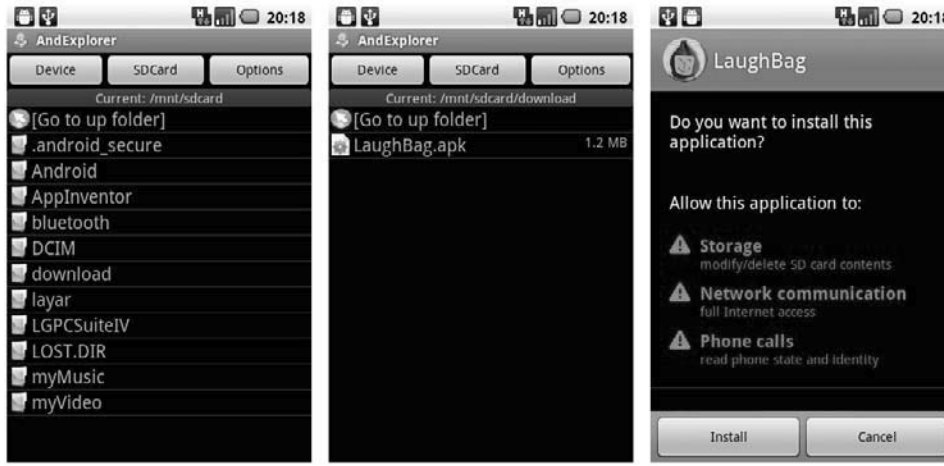


Figure 3.43 Installing the file LaughBag.apk using the AndExplorer file manager

As described in the previous section, an APK file is an archive of several files and directories. With a program for unpacking—for example, 7-Zip (`www.7-zip.org`)—you can take a closer look at the contents of this archive. Figure 3.43 shows the unpacked directory structure of the file `LaughBag.apk`. It is a bit bigger than that of the project file `LaughBag.zip` (shown in Figure 3.23), but still resembles it. In addition, the APK file now includes the Java-type files of an Android app—for example, the Android Manifest in the XML file of the same name, various meta files, and the integrated Java classes in `classes.dex`, plus the three media files as *assets* in the directory `/assets` (see Figure 3.44).

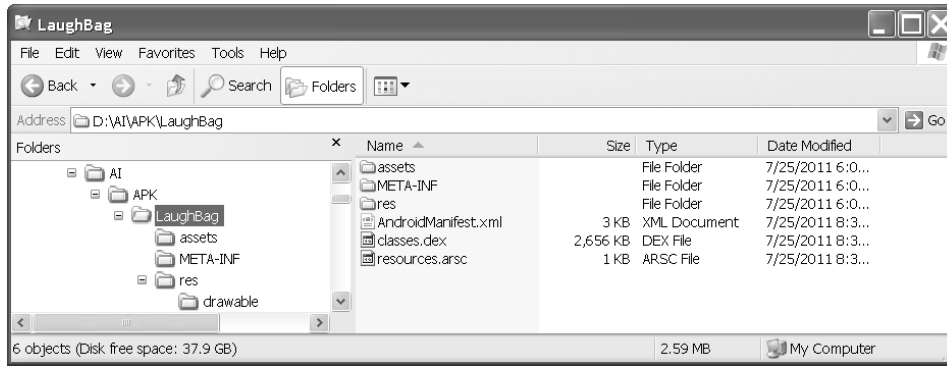


Figure 3.44 Directory structure of the unpacked app archive file LaughBag.apk

Now that you have used all three methods for downloading and installing the LaughBag app on your smartphone, it is entirely up to you which installation method you decide to use and when. If you want to test your independent app quickly during your development work, the direct installation is certainly the fastest option. If you are using several smartphones for testing, the quickest way of getting the app to the various Android devices is probably using the QR codes with online installation. If you want to make your app available to third parties for testing or general use, however, the best option is downloading the APK file.

Google Play and other Android Markets

In the context of exporting your own app as an APK file, you are probably wondering whether you could offer apps created with AI within Google Play (formerly known as Android Market), where apps are also available as APK files. Any developer who wants to distribute an app on Google Play first has to register, regardless of which development language has been used or whether the app is free or sold for a fee.

Registering as a Developer on Google Play

Before you can distribute your apps on Google Play, you first need to register and pay a one-time fee (currently $25). You can register online at this website:

`http://market.android.com/publish/signup`

For further help, refer to the Help page on the registration process:

`http://market.android.com/support/bin/answer.py?hl=en&answer=113468`

Before you think about registering to publish the apps you have developed with AI on Google Play, please read the rest of this section first.

On the Help page mentioned in the note, you can read a statement explaining the motivation behind this financial "hurdle": "We charge this fee to encourage higher-quality products on the market (e.g., less spammy products)." Just like other distributors of competing platforms for marketing apps, Google is trying to keep the quality of apps offered on Google Play as high as possible. Whereas some competitors require each of the apps to undergo a complicated approval procedure with more or less transparent evaluation criteria, the requirements for Google Play are not yet as restrictive. Despite the registration fee, the number of apps available on Google Play continues to grow rapidly, and it is becoming increasingly difficult to find a suitable app among the multitude of available apps. To keep the growing Google Play attractive to consumers as well as developers, Google is trying to limit the excessive proliferation of apps.

Against this backdrop, it should not be surprising to find that for a long time, apps created with AI were not intended or officially allowed to be distributed on Google Play. AI is aimed mainly at beginners and seeks to didactically provide basic methods of developing mobile apps; it is not targeted toward professional Java developers who want to develop commercial apps for Google Play. The prospect of hundreds of additional "HelloWorld" apps inundating Google Play understandably did not seem to be a good idea. Some of the discussions in the AI Forum, therefore, debate the question of whether professional apps can be created at all under the current limitations of AI and whether keeping them off Google Play altogether might be justified. Also, you should not forget that AI is only in the beta stage.

As there was no built-in export-to-market function integrated into AI for a long time, some clever alternative tools have been developed. You can still use the search term "market" in the AI Forum to find tips about publication tools such as Marketizer and AppToMarket, which have been available online and sometimes even free. With these tools, you are still supposedly able to sign the APK files created with AI relatively simply and convert them to compatible APK files, which you can then upload to Google Play with a valid registration.

> **Tools for Converting AI Apps for Google Play**
>
> According to the product information, the following tools enable you to convert, sign, and publish the APK files created with AI under a valid registration on Google Play:
>
> AppToMarket: `http://amerkashi.wordpress.com/`
>
> Marketizer: `http://www.taiic.com/marketizer/`

Since the release of AI Version 125 (see `http://beta.appinventor.mit.edu/ReleaseNotes.html`) from May 6, 2012, apps created with AI can be uploaded to Google Play. There is a complete description about how to download your apps from AI and upload them to Google Play on the documentation sites of the MIT AI.

How to Upload AI Apps to Google Play

You can find a complete description of how to prepare and upload your AI apps to Google Play at the MIT AI documentation website:

`http://beta.appinventor.mit.edu/learn/reference/other/appstoplay.html`

We should also mention that apart from Google Play, a growing number of alternative or additional online platforms are emerging on which you can offer your apps either free or for a charge. Please refer to the terms and conditions on the respective websites for details.

Alternatives to Google Play

In addition to or as an alternative to the official Google Play, you can offer your apps among all the others on these platforms:

Amazon Appstore for Android: `http://www.amazon.com/mobile-apps/b?ie=UTF8&node=2350149011`

AppBrain: `http://www.appbrain.com/`

GetJar: `http://www.getjar.com/`

Yet Another Android Market: `http://yaam.mobi/`

Regardless of where and how you want to offer your AI apps, it is essential to keep your target audience in mind. The success of an app depends not solely on marketing, but primarily on a good idea and its appealing and appropriate implementation—which is what we will concentrate on in the rest of this book.

This page intentionally left blank

Index