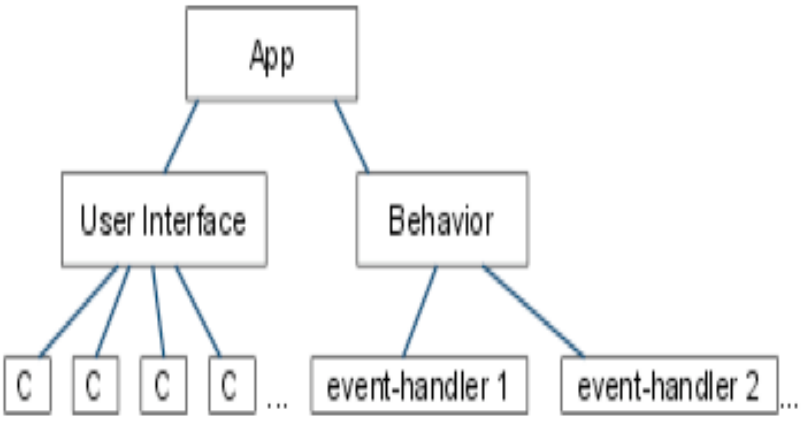
Table of Contents

Understanding an App's Architecture

This chapter examines the structure of an app from a programmer's perspective. It begins with the traditional analogy that an app is like a recipe, then proceeds to conceptualizing an app as a set of components that respond to events. The chapter also examines how apps can ask questions, repeat, remember, and talk to the web, all of which will be described in more detail in later chapters.

Many people can tell you what an app is from a user's perspective, but understanding what an app is from a programmer's perspective is more complicated. Apps have an internal structure that must be understood in order to create them effectively.

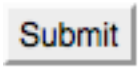
One way to describe an app's internals is to break it into two parts, its user interface and its behavior, sometimes denoted as an app's *look and feel*. Roughly, these correspond to the two main windows you use in App Inventor: you use the Component Designer to specify how the app will look, and you use the Blocks Editor to specify the app's behavior.



User Interface

The app's user interface consists of a set of components—things like the textboxes in which you type and the buttons you click.

Each component is defined by a set of properties. Most visual components have properties like Width, Height, and Alignment which together define how the component looks. So a button that looks like this to the end-user:



is internally defined as a set of properties such as:

Width	Height	Align	Text
50	30	center	Submit

The component properties are stored internally in the app's short-term memory. Each property can be thought of as a *named memory cell* holding a number or some text, something like the cells you see in a spreadsheet.

You modify component properties in the Component Designer to define the *initial* appearance of a component. Your app can also modify component properties *dynamically*—as the program

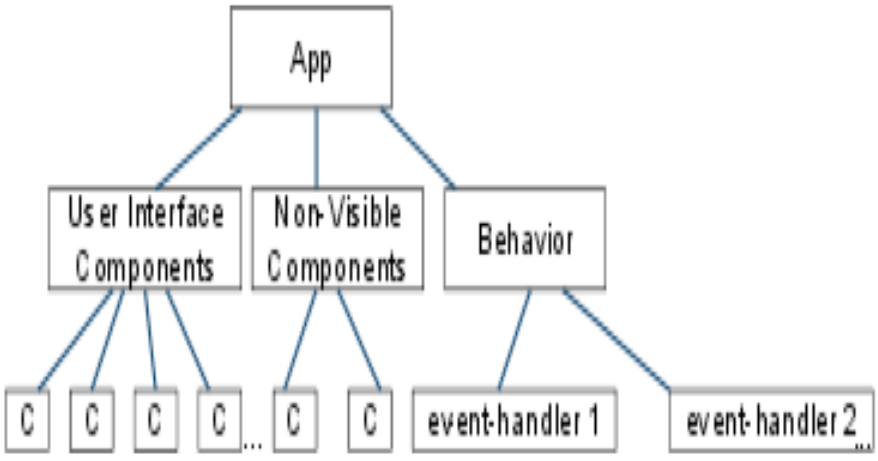
executes—with the Blocks Editor. For instance, you could have a button get larger each time it is clicked with the following blocks:

Each time the button is clicked, the value in the memory cell for the button's width increases by 3—for our sample, the value in the width box would change from 50 to 53. The end-user of the app wouldn't see this, but would see the button widen.

Non-Visual Components

It is not exactly correct to characterize an app as a set of user interface components and a set of behaviors—an app can also have *non-visible components*. A non-visible component has no appearance, so is not part of the user interface, but it does perform some functionality for the app. The (SMS) Texting component is an example of a non-visual component—the job it performs is to send and receive text messages. Other examples include the Text-to-Speech component and the TinyDB and TinyWebDB database components.

With non-visual components, the model for an app's architecture becomes:



Behavior

An app's components are straight-forward to understand. An app's behavior, on the other hand, is conceptually difficult and often complex. This section provides a model for understanding it.

Traditional Analogy: An App is a Recipe

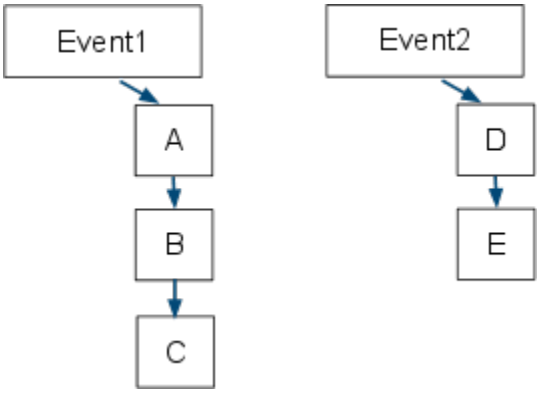
When computers were first invented, their apps (programs) were best described as recipes-- step-by-step instructions. An app specified that the computer should perform a sequence of steps in linear fashion:

A typical app might first load in a bank transaction (A), perform some computations and modify the customer's account (B), then print out the new balance on the screen.

An App is a set of Event-Handlers

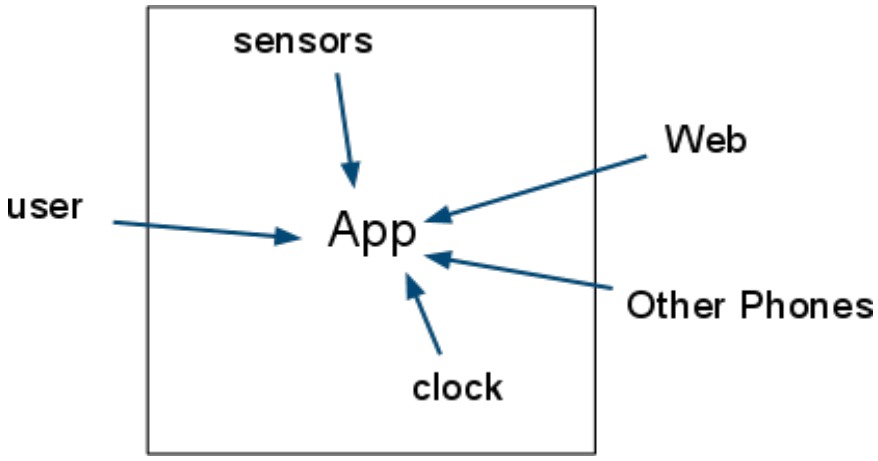
Most apps today, whether they be mobile, web, or desktop, don't fit the *recipe* paradigm. They don't just perform a bunch of instructions, they *react to events*, most commonly events initiated by the end-user of the app. The end-users act of clicking on a button is considered an *event*, and the app responds to the event by performing some operations, e.g., sending an SMS text. For touchscreen phones and devices, the act of dragging your finger across the screen is another event-- the app might respond to that event by drawing a line from your original touch to where you ended.

Apps that have a graphical user interface (GUI) are better conceptualized as a set of components that respond to events. The apps do consist of recipes-- step-by-step instructions-- but each recipe is only performed in response to some event:



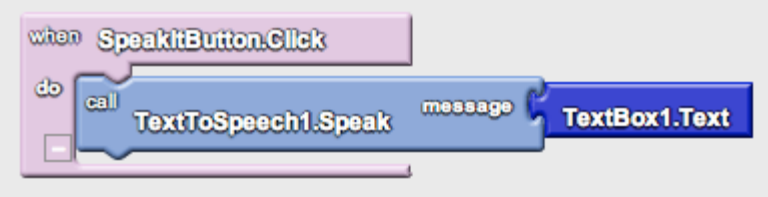
Events occur, and the app reacts by calling a sequence of functions. We call the event and the functions performed in response an *event-handler*.

End-user initiated events are just one of entity types. In general, an app will communicate with some entities within the phone (sensors, clock), and some outside the phone (the user, web, other phones):



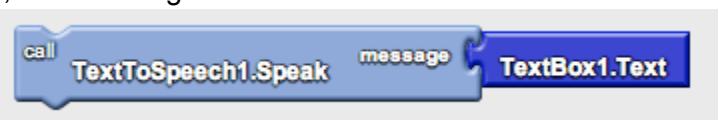
One reason App Inventor is easy to use is that you define an app's behavior using the event-handling paradigm directly. After designing the components-- the user interface-- of an app, you specify the behavior of the app in the Blocks Editor. An app's behavior consists of a set of event-handlers. You begin specifying each event-handler by dragging out an event-block, which has the form, "When <event> do".

For instance, consider an app "SpeakIt" that lets the user type in words and click a button, then speaks the typed words out loud. This application could be programmed with a single event-handler:



These blocks specify that when the event "SpeakItButton.Click" occurs-- when the user clicks the button-- the TextingToSpeech component should speak the words the user typed in the text box TextBox1. The event is "SpeakItButton.Click". The event-handler includes all the blocks above.

With App Inventor, nothing happens except in response to an event. You'll never drag function blocks into the Blocks Editor without hooking them into an event's "when-do" block. For instance, the following blocks:



don't make sense floating alone. The blocks specify that some words should be spoken aloud,

but they don't specify when such a "recipe" should be followed.

The events that can trigger activity fall into the following categories:

Event Type	Example
User-initiated event	when the user clicks button1 do...
Initialization event	when the app launches do...
Timer events	when 20 milliseconds passes do...
External events	when the phone receives a text do...

User-Initiated Events

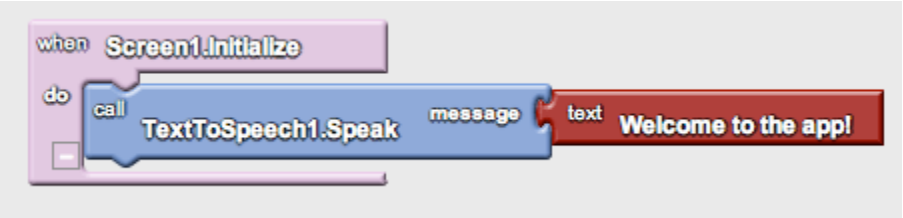
User-initiated events are the most common type of event. With input forms, it is typically the button click event that triggers a response by the app. With more graphical apps, such as drawing programs, dragging one's finger across the screen is the event.

Initialization Event

Sometimes your app just needs to perform some functions right when the app begins and not really in response to any end-user activity or other event. How does this fit in to the event-handling paradigm?

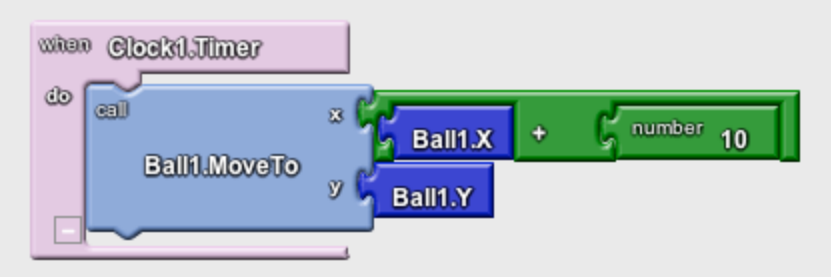
The answer is that event-handling languages, including App Inventor, consider the app being launched an event. In App Inventor, if you want some functions to be performed immediately when the app opens, you drag out a *Screen1.Initialize* event block and place some function blocks within it.

For instance, if you wanted to welcome the user with some spoken words when the app opens, you'd use blocks such as these:



Timer Events

Some activity in an app is triggered by the passing of time. An animated object is really an object that moves when triggered by a timer event. App Inventor has a clock component which can be used to trigger timer events. For instance, if you wanted ball on the screen to move 10 pixels horizontally every time interval, your blocks would look like this:



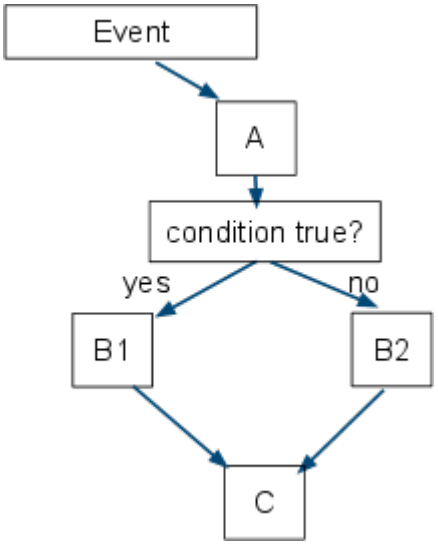
External Events

When your phone receives a text, this is an event. When your app requests some information from a web service, and that data arrives, it is considered an event. In general, you must consider your phone as an entity which interacts with other entities. With App Inventor, you'll program that interaction by specifying how your app should respond to an external event.

When you sit down to begin designing an app, one way to organize your thoughts is to think of all the events your app should respond to-- this is one way that you can breakdown a complex app into smaller conceptual parts.

An App Consists of Event-Handlers That Can Ask Questions and Branch

The responses to events are not always linear recipes, but can contain branches and loops. The app is allowed to ask questions -- to query the data within it-- and determine its course based on the answers. We say that such apps have *conditional branches:*



In the diagram, when the event occurs, the app performs operation A, then checks a condition. Function B1 is performed if the condition is true. If the condition is false, the app instead performs B2. Once either of the branches completes, the app continue on to performing function C.

Conditions are questions such as "has the score reached 100?" or "did the text I just received come from Joe". Conditions can be arbitrarily complex and include and,or, not and essentially any kind of boolean logic.

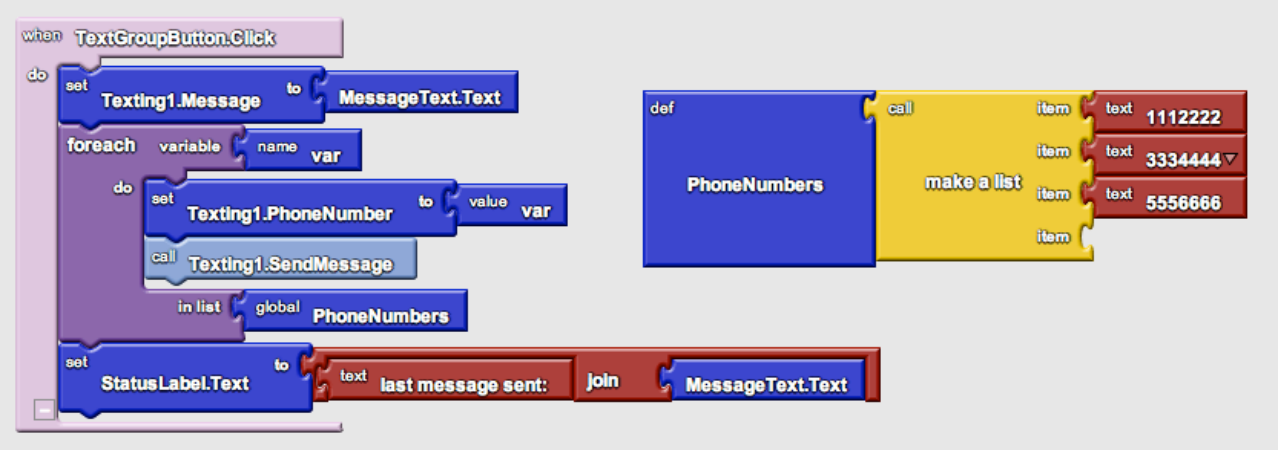
To check a condition in App Inventor, you'll use an "if-then" of "if-then-else" blocks. For instance, the following block would report a win if the player scored 100 points:

You can insert any type of condition in the test slot. If the test is true, the blocks within the then-do are executed . If it is false, the blocks are not executed.

Conditional blocks are discussed in detail in chapter X.

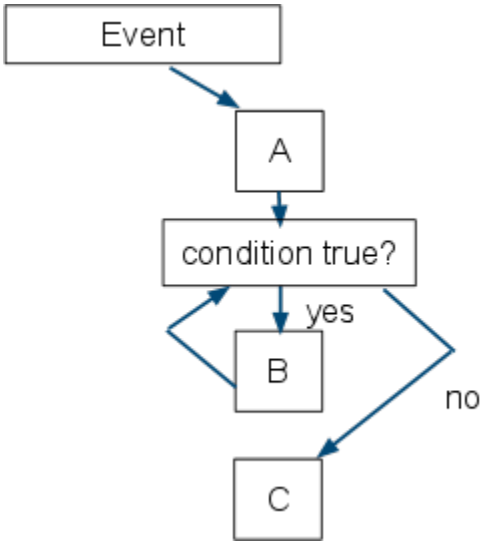
An App Consists of Event-Handlers That Can Ask Questions, Branch, and Repeat

Besides asking questions and branching based on the answer, you can also repeat operations multiple times. App Inventor provides two blocks for repeating, the *foreach* and the *while do.* Both enclose other blocks. With the foreach, all the blocks within it are performed once for each item in a list. For instance, if you wanted to text the same message to a list of phone numbers, the blocks would look like:



The blocks within the purple *foreach* block are repeated. In this case, they are repeated hree times because the list PhoneNumbers has three items. So the message is sent out to all three numbers.

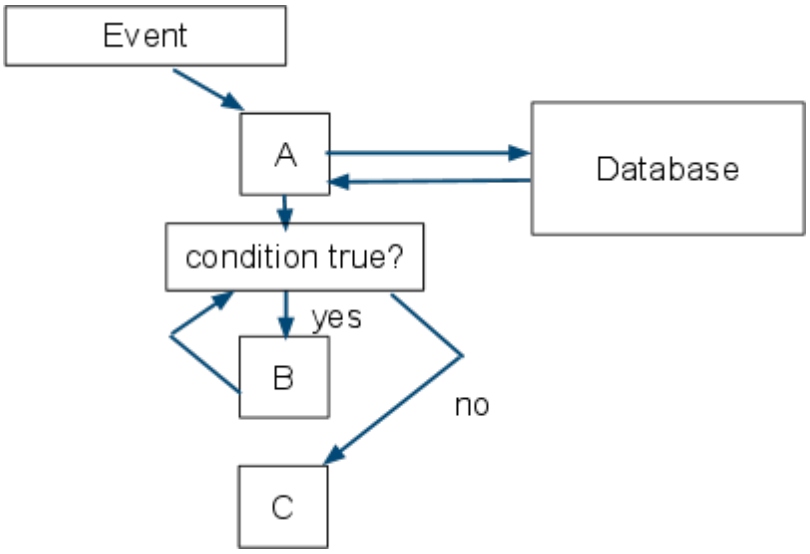
The *while do* is a more general repeat block. It is similar to an if or if-else in that you specify a test-- a question-- which determines things. In this case the condition determines whether or not the repeatable blocks within the *while do* should be executed again. If so, they're executed, and then the app performs an up-branch, or *loop,* and asks the question again:



For this diagram, when the event occurs, the operation A is executed, then a condition is checked. If it is true, B is executed. Each time B is executed, the app jumps back up and checks the condition again. This loop is repeated until the condition becomes false.

An App Consists of Event-Handlers That Can Ask Questions, Branch, Repeat and Remember Things

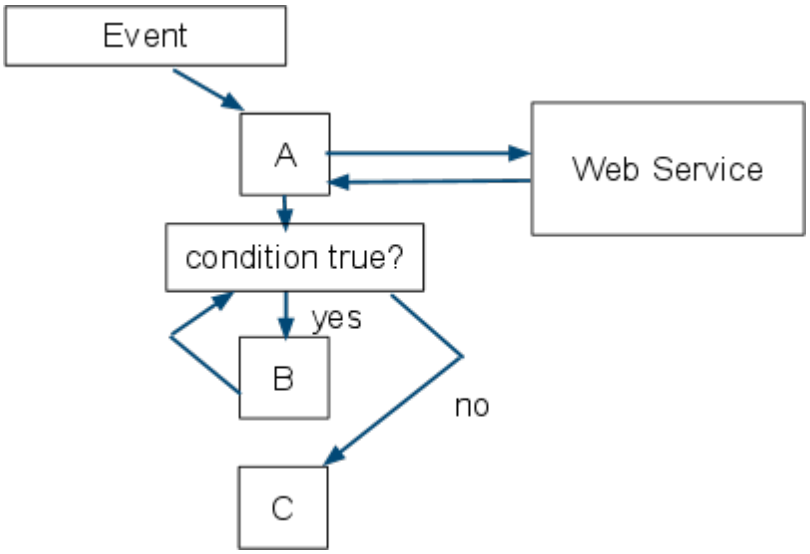
When you use apps, you expect them to remember things for you. For instance, when you enter a contact in the Contact Manager app, you expect that contact to be there the next time you use the app. Data that is remembered even after an app is closed is called *persistent data* and is stored in some type of a database:



App Inventor provides persistent memory in two ways. The TinyDB component allows you to store data persistently directly on the phone. The TinyWebDB component allows you to store data on the web, and thus share the data with other phones and apps. We'll explore the use of both of these database components and build apps such as a quiz application that lets the teacher create and maintain the list of questions.

An App Consists of Event-Handlers That Can Ask Questions, Branch, Repeat, Remember, and Talk to Web Services

Some apps use only the information within the phone or device. But many apps communicate with the outside world by sending requests to web services.



Twitter is an example of a web service that an App Inventor app can talk to. You can write apps that communicate with Twitter both to collect data to display on the phone, and to update your Twitter status. Apps that talk to more than one web service are called *mashups*.

Summary

A programmer must view an app both from an end-user perspective and from the inside-out app developer perspective. With App Inventor, you'll design how an app looks and then you'll design its behavior-- the set of event-handlers that lead an app to behave as you want. You'll build these event-handlers by assembling and configuring blocks representing events, operations, conditional branches, repeat loops, web service operations, and database operations, then testing your constructions by actually running the app on your phone. After writing a few programs, the mapping between the internal view of an app and its physical manifestation will become clear. When that happens, you will be a programmer!