CHAPTER 6

Paris Map Tour

In this chapter, you'll build an app that lets you create your own custom guide for a dream trip to Paris. And since a few of your friends can't join you, we'll create a companion app that lets them take a virtual tour of Paris as well. Creating a fully functioning map app might seem really complicated, but App Inventor lets you use the `ActivityStarter` *component to launch Google Maps for each virtual location. First, you'll build an app that launches maps for the Eiffel Tower, the Louvre, and Notre Dame Cathedral with a single click. Then you'll modify the app to create a virtual tour of satellite maps that are also available from Google Maps.*

What You'll Learn

This chapter introduces the following App Inventor components and concepts:

- The `Activity Starter` component for launching other Android apps from your app. You'll use this component here to launch Google Maps with various parameters.
- The `ListPicker` component for allowing the user to choose from a list of locations.

Designing the Components

Create a new project in App Inventor and call it "ParisMapTour". The user interface for the app has an `Image` component with a picture of Paris, a `Label` component with some text, a `ListPicker` component that comes with an associated button, and an `ActivityStarter` (non-visible) component. You can design the components using the snapshot in Figure 6-1.

The components listed in Table 6-1 were used to create this Designer window. Drag each component from the Palette into the Viewer and name it as specified.

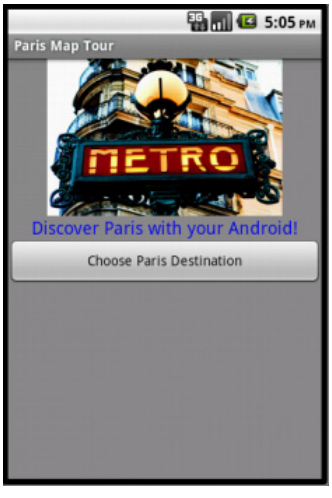

Figure 6-1. TheParis Map Tour app running in the emulator

Table 6-1. Components for the Paris Map Tour

Component type	Palette group	What you'll name it	Purpose
`Image`	Basic	`Image1`	Show a static image of a Paris map on screen.
`Label`	Basic	`Label1`	Display the text "Discover Paris with your Android!"
`ListPicker`	Basic	`ListPicker1`	Display the list of destination choices.
`ActivityStarter`	Other stuff	`ActivityStarter1`	Launch the Maps app when a destination is chosen.

Setting the Properties of ActivityStarter

`ActivityStarter` is a component that lets you launch any Android app—a browser, Google Maps, or even another one of your own apps. When a user launches another app from your app, he can click the back button to return to your app. You'll build ParisMapTour so that the Maps application is launched to show particular maps based on the user's choice. The user can then hit the back button to return to your app and choose a different destination.

`ActivityStarter` is a relatively low-level component in that you'll need to set some properties with information familiar to a Java Android SDK programmer, but foreign to the other 99.999% of the world. For this app, enter the properties as specified in Table 6-2, and *be careful*—even the upper-/lowercase letters are important.

Table 6-2. ActivityStarter properties for launching Google Maps

Property	Value
`Action`	android.intent.action.VIEW
`ActivityClass`	com.google.android.maps.MapsActivity
`ActivityPackage`	com.google.android.apps.maps

In the Blocks Editor, you'll set one more property, `DataUri`, which lets you launch a specific map in Google Maps. This property must be set in the Blocks Editor instead of the Component Designer because it needs to be dynamic; it will change based on whether the user chooses to visit the Eiffel Tower, the Louvre, or the Notre Dame Cathedral.

We'll get to the Blocks Editor in just a moment, but there are a couple more details to take care of before you can move on to programming the behavior for your components:

1. Download the file *metro.jpg* from the book site (*http://examples.oreilly.com/0636920016632/*) onto your computer, and then choose Add in the Media section to load it into your project. You'll then need to set it as the `Picture` property of `Image1`.
2. The `ListPicker` component comes with a button; when the user clicks it, the choices are listed. Set the text of that button by changing the `Text` property of `ListPicker1` to "Choose Paris Destination".

Adding Behaviors to the Components

In the Blocks Editor, you'll need to define a list of destinations, and two behaviors:

- When the app begins, the app loads the destinations into the `ListPicker` component so the user can choose one.
- When the user chooses a destination from the `ListPicker`, the Maps application is launched and shows a map of that destination. In this first version of the app, you'll just open Maps and tell it to run a search for the chosen destination.

Creating a List of Destinations

Open the Blocks Editor and create a variable with the list of Paris destinations using the blocks listed in Table 6-3.

Table 6-3. Blocks for creating a destinations variable

Block type	Drawer	Purpose
def variable ("Destinations")	Definitions	Create a list of the destinations.
make a list	Lists	Add the items to the list.
text ("Tour Eiffel")	Text	The first destination.
text ("Musée du Louvre")	Text	The second destination.
text ("Cathédrale Notre Dame")	Text	The third destination.

The `destinations` variable will call the `make a list` function, into which you can plug the text values for the three destinations in your tour, as shown in Figure 6-2.

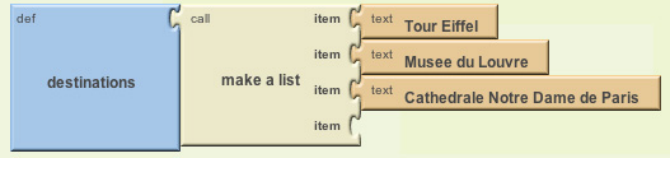

Figure 6-2. Creating a list is easy in App Inventor

Letting the User Choose a Destination

The purpose of the `ListPicker` component is to display a list of items for the user to choose from. You preload the choices into the `ListPicker` by setting the property `Elements` to a list. For this app, you want to set the `ListPicker`'s `Elements` property to the `destinations` list you just created. Because you want to display the list when the app launches, you'll define this behavior in the **Screen1.Initialize** event. You'll need the blocks listed in Table 6-4.

Table 6-4. Blocks for launching the ListPicker when the app starts

Block type	Drawer	Purpose
Screen1.Initialize	Screen1	This event is triggered when the app starts.
set ListPicker1 .Elements to	ListPicker1	Set this property to the list you want to appear.
global destinations	My Definitions	The list of destinations.

How the blocks work

Screen1.Initialize is triggered when the app begins. As shown in Figure 6-3, the event handler sets the `Elements` property of `ListPicker` so that the three destinations will appear.

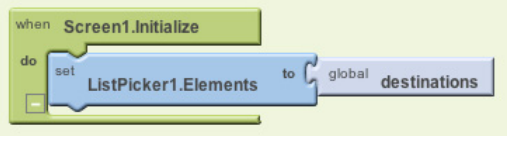

Figure 6-3. Put anything you want to happen when the app starts in a Screen1.Initialize event handler

Test your app. *First, you'll need to restart the app by clicking "Connect to Device. . ." in the Blocks Editor. Then, on the phone, click the button labeled "Choose Paris Destination." The list picker should appear with the three items.*

Opening Maps with a Search

Next, you'll program the app so that when the user chooses one of the destinations, the `ActivityStarter` launches Google Maps and searches for the selected location.

When the user chooses an item from the `ListPicker` component, the **ListPicker .AfterPicking** event is triggered. In the event handler for `AfterPicking`, you need to set the `DataUri` of the `ActivityStarter` component so it knows which map to open, and then you need to launch Google Maps using **ActivityStarter.StartActivity**. The blocks for this functionality are listed in Table 6-5.

Table 6-5. Blocks to launch Google Maps with the Activity Starter

Block type	Drawer	Purpose
ListPicker1.After Picking	ListPicker1	This event is triggered when the user chooses from `ListPicker`.
set ActivityStarter1 .DataUri to	ActivityStarter1	The `DataUri` tells Maps which map to open on launch.
make text	Text	Build the `DataUri` from two pieces of text.
text ("geo:0,0?q=")	Text	The first part of the `DataUri` expected by Maps.
ListPicker1.Selection	ListPicker1	The item the user chose.
ActivityStarter1 .StartActivity	ActivityStarter1	Launch Maps.

How the blocks work

When the user chooses from the `ListPicker`, the chosen item is stored in **ListPicker .Selection** and the `AfterPicking` event is triggered. As shown in Figure 6-4, the `DataUri` property is set to a text object that combines "http://maps.google.com/?q=" with the chosen item. So, if the user chose the first item, "Tour Eiffel," the `DataUri` would be set to "http://maps.google.com/?q=Tour Eiffel."

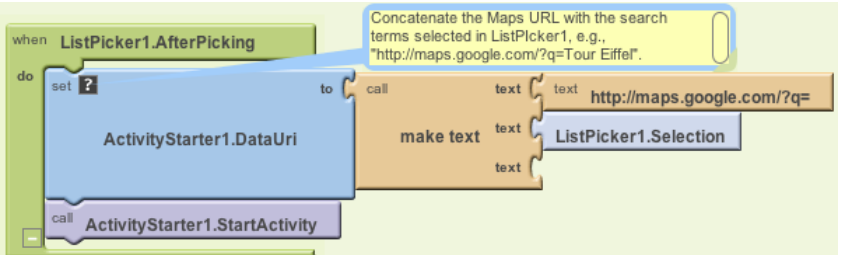

Figure 6-4. Setting the DataURI to launch the selected map

Since you already set the other properties of the `ActivityStarter` so that it knows to open Maps, the **ActivityStarter1.StartActivity** block launches the Maps app and invokes the search proscribed by the `DataUri`.

Test your app. *Restart the app and click the "Choose Paris Destination" button again. When you choose one of the destinations, does a map of that destination appear? Google Maps should also provide a back button to return you to your app to choose again—does that work? (You may have to click the back button a couple of times.)*

Setting Up a Virtual Tour

Now let's spice up the app and have it open some great zoomed-in and street views of the Paris monuments so your friends at home can follow along while you're away. To do this, you'll first explore Google Maps to obtain the URLs of some specific maps. You'll still use the same Parisian landmarks for the destinations, but when the user chooses one, you'll use the *index* (the position in the list) of her choice to select and open a specific zoomed-in or street-view map.

Before going on, you may want to save your project (using Save As) so you have a copy of the simple map tour you've created so far. That way, if you do anything that causes issues in your app, you can always go back to this working version and try again.

Finding the DataUri for Specific Maps

The first step is to open Google Maps on your computer to find the specific maps you want to launch for each destination:

1. On your computer, browse to *http://maps.google.com*.
2. Search for a landmark (e.g., the Eiffel Tower).

3. Zoom in to the level you desire.
4. Choose the type of view you want (e.g., Address, Satellite, or Street View).
5. Click the Link button near the top right of the Maps window and copy the URL for the map. You'll use this URL (or parts of it) to launch the map from your app.

Using this scheme, Table 6-6 shows the URLs you'll use.

Table 6-6. Virtual tour URLs for Google Maps

Landmark	Maps URL
Tour Eiffel	*http://maps.google.com/maps?f=q&source=s_q&hl=en&geocode=&q=eiffel+tower&sll=37.0625,-95.677068&sspn=48.909425,72.333984&ie=UTF8&hq=Tour+Eiffel&hnear=Tour+Eiffel,+Quai+Branly,+75007+Paris,+Ile-de-France,+France&ll=48.857942,2.294748&spn=0.001249,0.002207&t=h&z=19*
Musée du Louvre	*http://maps.google.com/maps?f=q&source=s_q&hl=en&q=louvre&sll=48.86096,2.335421&sspn=0.002499,0.004415&ie=UTF8&t=h&split=1&filter=0&rq=1&ev=zi&radius=0.12&hq=louvre&hnear=&ll=48.86096,2.335421&spn=0.002499,0.004415&z=18*
Cathédrale Notre Dame (Street View)	*http://maps.google.com/maps?f=q&source=s_q&hl=en&q=french+landmarks&sll=48.853252,2.349111&sspn=0.002411,0.004415&ie=UTF8&t=h&radius=0.12&split=1&filter=0&rq=1&ev=zi&hq=french+landmarks&hnear=&ll=48.853252,2.349111&spn=0,0.004415&z=18&layer=c&cbll=48.853046,2.348861&panoid=74fLTqeYdgkPYj6KKLlqgQ&cbp=12,63.75,,0,-35.58*

To view any of these maps, paste the URLs from Table 6-6 into a browser. The first two are zoomed-in satellite views, while the third is a street view.

You can use these URLs directly to launch the maps you want, or you can define cleaner URLs using the Google Maps protocols outlined at *http://mapki.com*. For example, you can show the Eiffel Tower map using only the GPS coordinates found in the long URL in Table 6-6 and the Maps `geo:` protocol:

```
geo:48.857942,2.294748?t=h&z=19
```

Using such a `DataUri`, you'll get essentially the same map as the map based on the full URL from which the GPS coordinates were extracted. The `t=h` specifies that Maps should show a hybrid map with both satellite and address views, and the `z=19` specifies the zoom level. If you're interested in the details of setting parameters for various types of maps, check out the documentation at *http://mapki.com*.

To get comfortable using both types of URLs, we'll use the `geo:` format for the first two `DataUri` settings in our list, and the full URL for the third.

Defining the dataURIs List

You'll need a list named `dataURIs`, containing one `DataURI` for each map you want to show. Create this list as shown in Figure 6-5 so that the items correspond to the items in the destinations list (i.e., the first `dataURI` should correspond to the first destination, the Eiffel Tower).

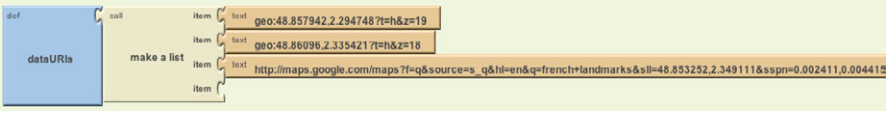

Figure 6-5. The list of maps for your virtual tour

The first two items shown are `DataURIs` for the Eiffel Tower and the Louvre. They both use the `geo:` protocol. The third `DataURI` is not shown completely because the block is too long for this page; you should copy this URL from the entry for "Notre Dame, Street View" in Table 6-6 and place it in a **text** block.

Modifying the ListPicker.AfterPicking Behavior

In the first version of this app, the **ListPicker.AfterPicking** behavior set the `DataUri` to the *concatenation* (or combination) of "http://maps.google.com/?q=" and the destination the user chose from the list (e.g., Tour Eiffel). In this second version, the `AfterPicking` behavior must be more sophisticated, because the user is choosing from one list (`destinations`), but the `DataUri` must be selected from another list (`dataURIs`). Specifically, when the user chooses an item from the `ListPicker`, you need to know the *index* of his choice so you can use it to select the correct `DataUri` from the `dataURIs` list. We'll explain more about what an index is in a moment, but it helps to set up the blocks first to better illustrate the concept. There are quite a few blocks required for this functionality, all of which are listed in Table 6-7.

Table 6-7. Blocks for choosing a list item based on the user's selection

Block type	Drawer	Purpose
def variable ("index")	Definitions	This variable will hold the index of the user's choice.
number (1)	Math	Initialize the index variable to 1.
ListPicker1 .AfterPicking	ListPicker1	This event is triggered when the user chooses an item.
set global index to	My Definitions	Set this variable to the position of the selected item.
position in list	Lists	Get the position (index) of a selected item.
ListPicker1 .Selection	ListPicker1	The selected item—for example, "Tour Eiffel." Plug this into the "thing" slot of **position in list**.
global destinations	My Definitions	Plug this into the "list" slot of **position in list**.
set ActivityStarter .DataUri	ActivityStarter	Set this before starting the activity to open the map.
select list item	Lists	Select an item from the **dataURIs** list.
global DataURIs	My Definitions	The list of `DataURIs`.

Table 6-7. Blocks for choosing a list item based on the user's selection (continued)

Block type	Drawer	Purpose
global index	My Definitions	Hold the position of the chosen item.
ActivityStarter .StartActivity	ActivityStarter	Launch the Maps app.

How the blocks work

When the user chooses an item from the `ListPicker`, the `AfterPicking` event is triggered, as shown in Figure 6-6. The chosen item—e.g., "Tour Eiffel"—is in **ListPicker.Selection**. The event handler uses this to find the position of the selected item, or the *index* value, in the `destinations` list. The index corresponds to the position of the chosen destination in the list. So if "Tour Eiffel" is chosen, the index will be 1; if "Musée du Louvre" is chosen, it will be 2; and if "Cathédrale Notre Dame de Paris" is chosen, the index will be 3.

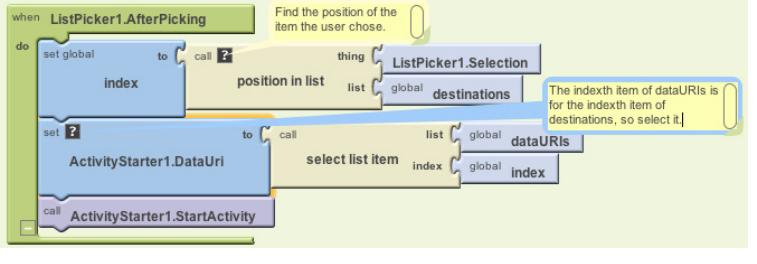

Figure 6-6. Choosing a list item based on the user's selection

The index can then be used to select an item from another list—in this case, `dataURIs`—and to set that entry as the `ActivityStarter`'s `DataUri`. Once this is set, the map can be launched with **ActivityStarter.StartActivity**.

Test your app. *On the phone, click the button labeled "Choose Paris Destination." The list should appear with the three items. Choose one of the items and see which map appears.*

Variations

Here are some suggested variations to try:

- Create a virtual tour of some other exotic destination, or of your workplace or school.
- Create a customizable Virtual Tour app that lets a user create a guide for a location of her choice by entering the name of each destination along with the URL of a corresponding map. You'll need to store the data in a `TinyWebDB` database and create a Virtual Tour app that works with the entered data. For an example of how to create a `TinyWebDB` database, see the MakeQuiz/TakeQuiz app in Chapter 10.

Summary

Here are some of the ideas we've covered in this chapter:

- List variables can be used to hold data like map destinations and URLs.
- The `ListPicker` component lets the user choose from a list of items. The `ListPicker`'s `Elements` property holds the list, the `Selection` property holds the selected item, and the `AfterPicking` event is triggered when the user chooses an item from the list.
- The `ActivityStarter` component allows your app to launch other apps. This chapter demonstrated its use with the Maps application, but you can launch a browser or any other Android app as well, even another one you created yourself. See *http://appinventor.googlelabs.com/learn/reference/other/activitystarter.html* for more information.
- You can launch a particular map in Google Maps by setting the `DataUri` property. You can find URIs by configuring a particular map in the browser and then choosing the Link button to find the URI. You can either place such a URI directly into the `DataUri` of your `ActivityStarter`, or build your own URI using the protocols defined at *http://mapki.com*.
- You can identify the *index* of a list item using the **position in list** block. With `ListPicker`, you can use list position to find the index of the item the user chooses. This is important when, as in this chapter, you need the index to choose an item from a second, related list. For more information on `List` variables and the `ListPicker` component, see Chapter 19.