CHAPTER 6

Paris Map Tour

Figure 6-1.

In this chapter, you'll build a tour guide app for a trip to Paris. Creating a fully functioning map app might seem really complicated, but App Inventor provides two high-level components to help: the `ActivityStarter`*, which makes it possible for you to launch another app from your app, including Google Maps, and the* `WebViewer`*, which shows any web page you want within a subpanel of your app. You'll explore both of these components and build two different versions of a tour guide.*

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.
- The `WebViewer` component for showing web pages within your app.
- How to use list variables to store information for your app.

- The `ListPicker` component to give the user the ability to choose from a list of locations.
- How to build a `URL` dynamically to show different maps.

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 in this first version, an `ActivityStarter` (non-visible) component. You can design the components using the snapshot in *Figure 6-1*.

Figure 6-2. The Paris Map Tour app running in the emulator

You'll need the components listed in *Table 6-1* to build this app. Drag each component from the Palette into the Viewer and name it as specified.

Table 6-1. Components for the Paris Map Tour

Component type	Palette group	What you'll name it	Purpose
`Image`	User Interface	`Image1`	Show a static image of Paris on screen.
`Label`	User Interface	`Label1`	Display the text "Discover Paris with your Android!"
`ListPicker`	User Interface	`ListPicker1`	When clicked, a list of destination choices will appear.
`ActivityStarter`	Connectivity	`ActivityStarter1`	Launch the Maps app when a destination is chosen.

Setting the Properties of ActivityStarter

`ActivityStarter` is a component with which you can launch any Android app, including Google Maps or another one of your own apps. You'll first build ParisMapTour so that the Maps application is launched to show particular maps based on the user's choice. The user can then tap 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 that would be familiar to a Java Android SDK programmer, but completely foreign to the other 99.99% of the world. For this app, enter the properties as specified in *Table 6-2*, and *be careful*—they're case-sensitive, meaning that whether a letter is uppercase or lowercase is 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 provide a URL to 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* to load into your project. Then, 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 that 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 instruct 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 by using the blocks listed in *Table 6-3*.

Table 6-3. Blocks for creating a destinations variable

Block type	Drawer	Purpose
`initialize global` ("Destinations")	Variables	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.

When you drag the `make a list` block into your app, it will have only two available sockets. You can add another one by clicking the dark blue icon it and adding a third item.

After you've done that, just create the text blocks for each destination and place them in the three sockets of `make a list`, as shown in *Figure 6-2*.

Figure 6-3. A list of three items

LETTING THE USER CHOOSE A DESTINATION

The list you just defined does not appear in the user interface—no variables do. You'll use a `ListPicker` component to display the 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 `Elements` property for `ListPicker` to the `destinations` list you just created. Because this only needs to be set once, you'll

define this behavior in the `Screen1.Initialize` event. You'll need the blocks that are 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 that you want to appear.
`get global destinations`	Drag out from variable initialization block	The list of destinations.

How the blocks work

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

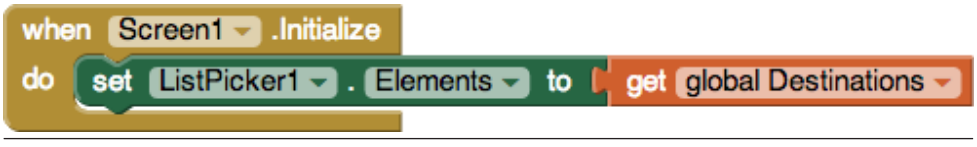

Figure 6-4. Initialize the ListPicker with the three choices when the app launches

Test your apps *Click Connect and set up live testing with your device or emulator. Then, click the button labeled "Choose Paris Destination." The list picker should appear with the three items. At this point, nothing should happen when you choose an item.*

OPENING MAPS WITH A SEARCH URL

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.

First, consider the URL *http://maps.google.com?q=Paris*. When you type this URL into the address bar of a browser, it shows a map of Paris. The "?" is common to many URLS; it signifies that a parameter is coming. A parameter is the information the website needs to process the request. In this case, the parameter name is "q", short for "query", and its value is "Paris". It instructs Google Maps what map to display.

In this app, you'll build a URL dynamically, adding the parameter value based on which location the user chooses. This way you can show different maps based on the user's choices.

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 that it knows which map to open, and then you need to launch Google Maps by 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.AfterPicking`	ListPicker1	This event is triggered when the user chooses from `ListPicker`.
`set ActivityStarter1.DataUri to`	ActivityStarter1	The `DataUri` instructs Maps which map to open on launch.
`join`	Text	Build the `DataUri` from two pieces of text.
`text` ("http://maps.google.com?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."

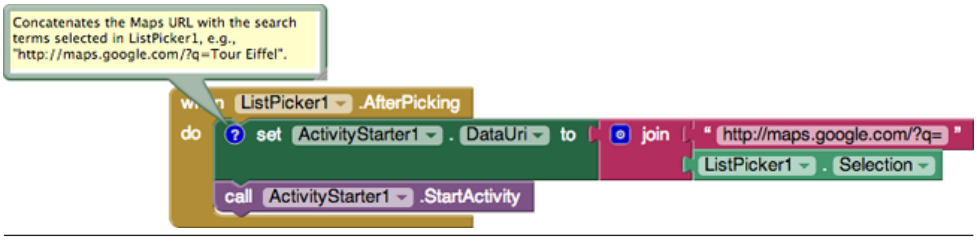

Figure 6-5. Setting the DataURI to launch the selected map

Because 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 prescribed 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? Can you get back to your app with the device's back button?*

The Complete App: Map Tour with Activity Starter

Figure 6-5 shows the final block configuration for version 1 of Paris Map Tour.

Figure 6-6. The complete Map Tour app (version 1)

A VIRTUAL TOUR WITH THE WEB VIEWER

The `ActivityStarter` is an important component because it provides access to any other app on the device. But, there is another way to build a tour guide that uses a different component, instead; the `WebViewer`. `WebViewer` is a panel you place directly within your app that behaves like a browser. You can open any web page, including a Google Map, in the viewer, and you can programmatically change the page that appears. Unlike with an ActivityStarter, your user doesn't ever leave your app, so you don't have to count on them hitting the back button to get back.

In this second version of the app, you'll use the `WebViewer` and you'll also spice up the app so that it opens some zoomed-in and street views of the Paris monuments. You'll define a second list and use a more complicated scheme to decide which map to show. To begin, 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 might want to save your project (using Save As) so you have a copy of the `ActivityStarter` 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.

Add the Web Viewer

In the designer, delete the `ActivityStarter` component. Then, from the User Interface drawer, drag in a `WebViewer` component and place it below the other components. Uncheck the `Screen1.Scrollable` property so the `WebViewer` will display pages correctly.

FINDING THE URL FOR SPECIFIC MAPS

The next 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., Street View).
5. Grab the URL. In the classic version of Maps, you click the Link button near the top right of the Maps window and copy the URL for the map. In the newer version of Google Maps you can just grab the URL from the address bar.

Use this scheme to create some cool maps of the Paris monuments and extract the URLs. *Table 6-6* provides some samples if you'd rather use them (the URLs have been shortened with the bit.ly service).

Table 6-6. Virtual tour URLs for Google Maps

Landmark	Maps URL
Tour Eiffel	*http://bit.ly/1qiEy8B*
Musée du Louvre	*http://bit.ly/1qiEVQA*
Cathédrale Notre Dame (street view)	*http://bit.ly/1qiF1YD*

To view any of these maps in a browser, paste the URLs from *Table 6-6* into the address bar.

DEFINING THE URLS LIST

You'll need a list named `URLs`, containing a URL for each of the destinations. Create this list as shown in *Figure 6-6* so that the items correspond to the items in the destinations list (i.e., the first `URL` should correspond to the first destination, the Eiffel Tower).

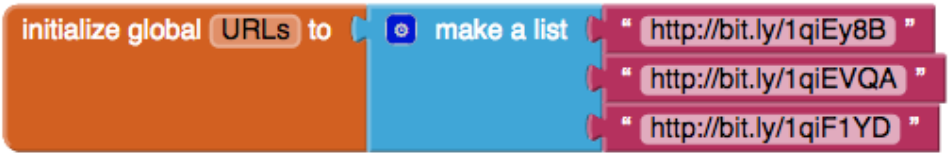

Figure 6-7. Copy and paste the URLs into the text blocks of the URLs list

MODIFYING THE LISTPICKER.AFTERPICKING BEHAVIOR

In the first version of this app, the `ListPicker.AfterPicking` behavior set the `DataUri` to a 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 app is choosing from the `URLs list` for the URL. Specifically, when the user chooses an item from the `ListPicker`, you need to know the index of the choice so you can use it to select the correct `URL` from the 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
`ListPicker1.AfterPicking`	ListPicker1	This event is triggered when the user chooses an item.
`ListPicker1.SelectionIndex`	ListPicker1	The index (position) of the chosen item.
`select list item`	Lists	Select an item from the **`URLs`** list.
`get global URLs`	Drag it from the variable initialization	The list of **`URLs`**.
`WebViewer.GoToURL`	WebViewer	Load the URL in the viewer to show the map.

How the blocks work

When the user chooses an item from the `ListPicker`, the `AfterPicking` event is triggered, as shown in *Figure 6-7*. The chosen item—for example, "Tour Eiffel"—is in `ListPicker.Selection`. You used this property in the first version of this app. However, ListPicker also has a property `SelectionIndex`, which corresponds to the position of the chosen destination in the list. So, if "Tour Eiffel" is chosen, the `SelectionIndex` will be 1; if "Musée du Louvre" is chosen, it will be 2; and if "Cathédrale Notre Dame de Paris" is chosen, it will be 3.

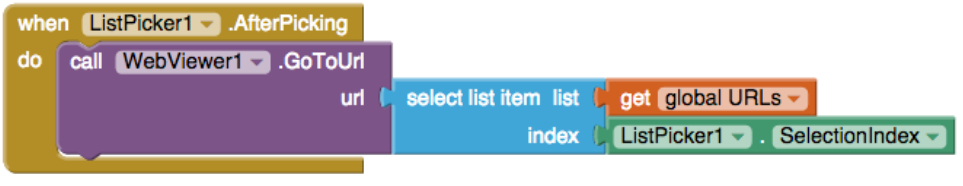

Figure 6-8. Open the selected URL in the WebViewer

You use `ListPicker.SelectionIndex` to select an item from the `URLs` list. This works because the items on the two lists, `destinations` and `URLs`, are in sync: the first destination corresponds to the first URL, the second to the second, and the third to the third. So, even though the user chooses an item from one list, you can use their choice (well, the index of their choice) to select the right URL to show.

Test your app *On the device, 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.*

The Complete App: Map Tour (Web Viewer)

Figure 6-8 shows the final block configuration for this second version of Paris Map Tour.

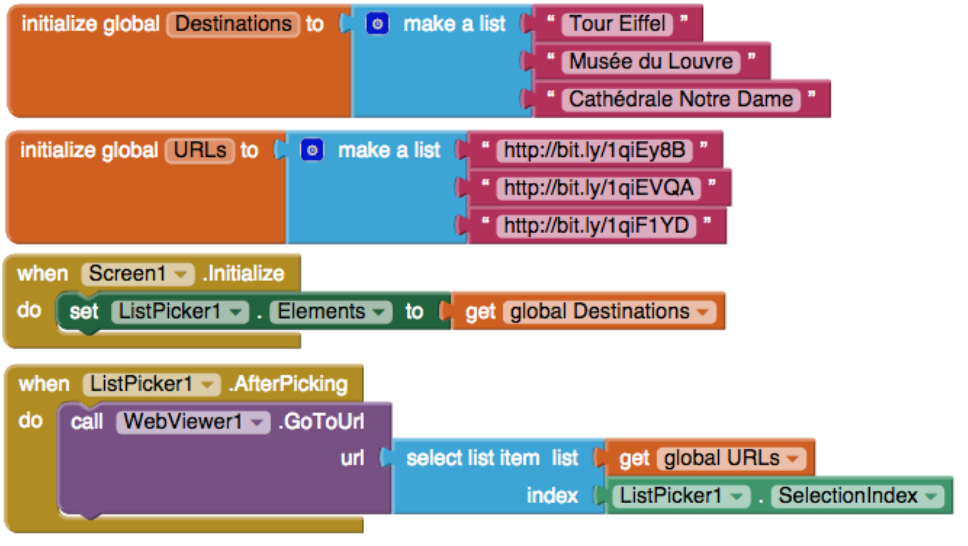

Figure 6-9. The complete Map Tour App (WebViewer version)

Variations

Here are some suggested variations to try:

- Create a virtual tour of your workplace or school, or for your next vacation destination.
- Explore `ActivityStarter` and use it to send an email or launch an app such as YouTube (see *http://bit.ly/1qiFx8Z* for help).
- Difficult: 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.

Summary

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

- You can use list variables to hold data such as 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, the `SelectionIndex` holds the position of the selected item, and the `AfterPicking` event is triggered when the user chooses an item from the list.

- The `ActivityStarter` component makes it possible for your app to launch other apps. This chapter demonstrated its use with the Google Maps application, but you can launch a browser or any other Android app as well, even another one that you created yourself.
- You can use `ListPicker.SelectionIndex` to get the position of an item that a user chooses from a list. You can then use that index to select information from a different list (whose items are synchronized with the first list). For more information on `List` variables and the `ListPicker` component, see *Chapter 19*.