Make This

Chapter 3 Building Your First App: Your Happy Place

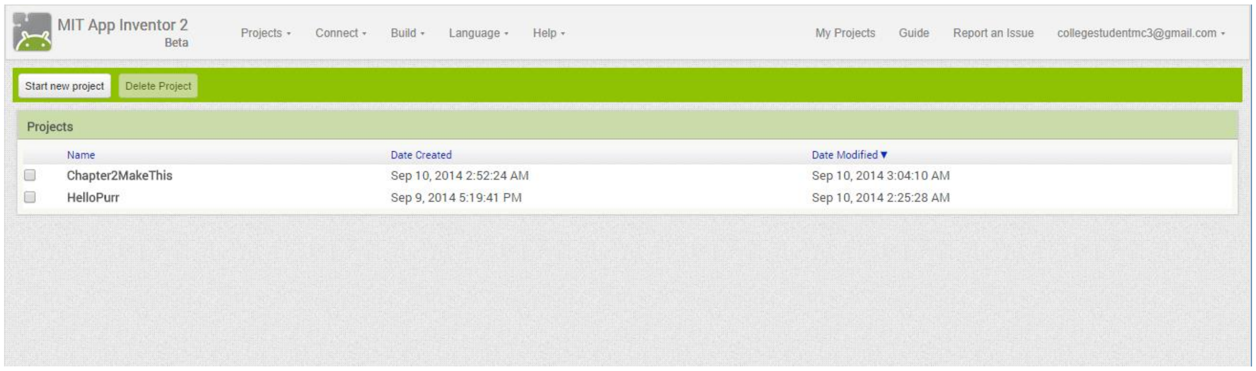
In this exercise, you will use App Inventor to build your first app. This app, called Your Happy Place, teaches the following skills:

- **Starting a new project in App Inventor**
- **Generating a list of items from which a user can pick (list picker)**
- **Viewing a web page**
- **Translation of text to speech**

Note: Before attempting this exercise, complete the Chapter 2 exercise to familiarize yourself with the App Inventor interface and get your Android device connected.

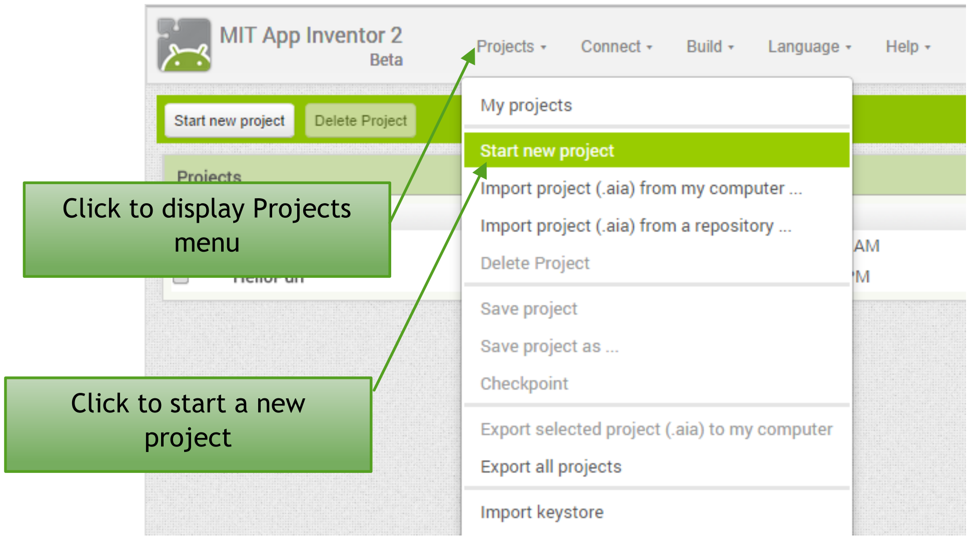
Starting a New Project

1. Navigate to http://appinventor.mit.edu/explore/. Click the **Create!** button.
2. If necessary, sign in with your Google Account. Click the **Continue** button to dismiss the dialog box. Click the **My Projects** link or click the **Projects** dropdown arrow and click **My Projects** from the dropdown menu. If you completed the Chapter 2 exercise, your screen should look like this:

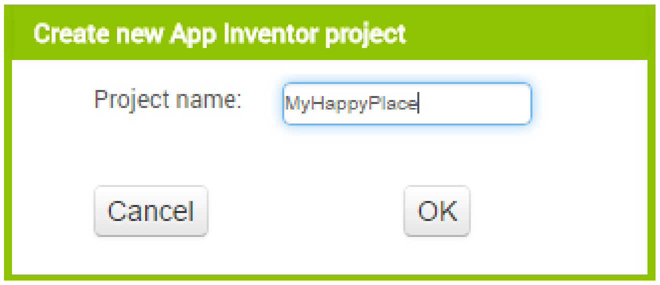


Name	Date Created	Date Modified ▼
Chapter2MakeThis	Sep 10, 2014 2:52:24 AM	Sep 10, 2014 3:04:10 AM
HelloPurr	Sep 9, 2014 5:19:41 PM	Sep 10, 2014 2:25:28 AM

3. At the top of the screen, click the **Projects** dropdown arrow to display the Projects menu. From the Projects menu, click **Start new project**. (Alternatively, you could just click the **Start new project button**.)



4. In the Create new App Inventor project dialog box, enter *MyHappyPlace* in the **Project name** box. Click **OK**.



Alert: App Inventor project names cannot contain spaces!

5. Connect App Inventor to your Android device (see instructions in Chapter 2).

App Design

In this app, you will create an interface that allows users to pick their “happy place” from a list, then display images of their happy place from Flickr.com. The app will also provide a verbal confirmation of the selection.

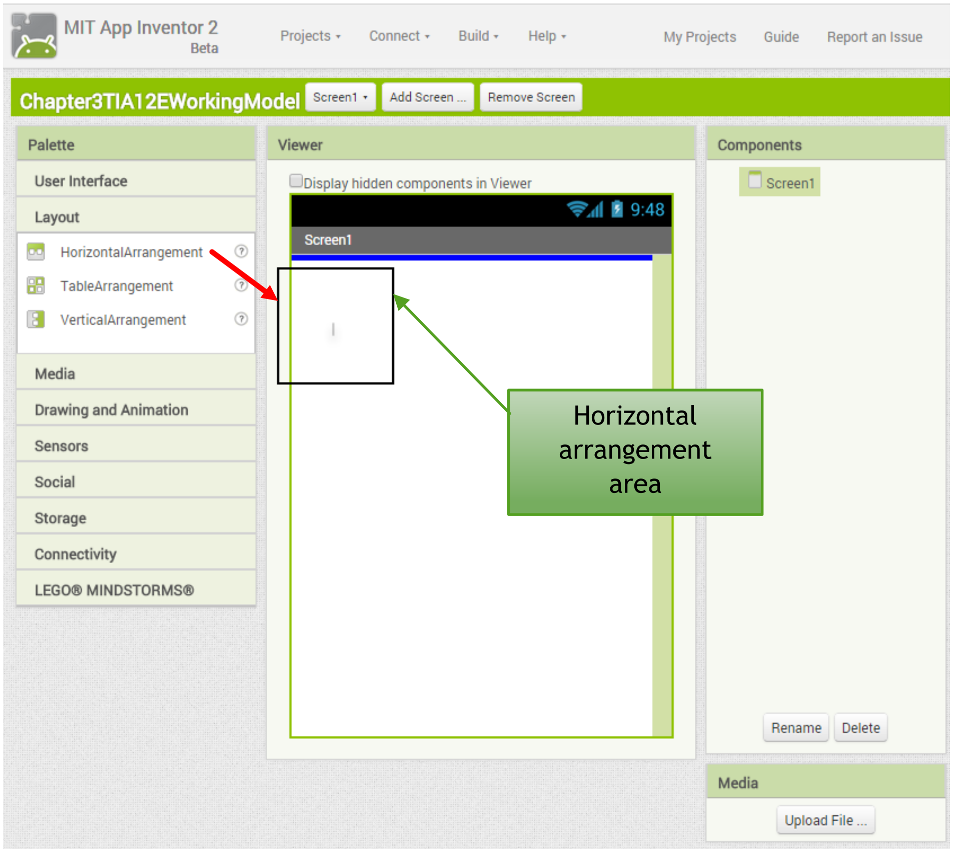
Part 1 - Creating a List

Lists are useful in many apps, such as selecting the color of your phone case when ordering it online. App Inventor has a list picker function that makes creating lists for your app relatively simple. You need to create a list of potential “happy places” from which the user of our app can select and generate the web search for images.

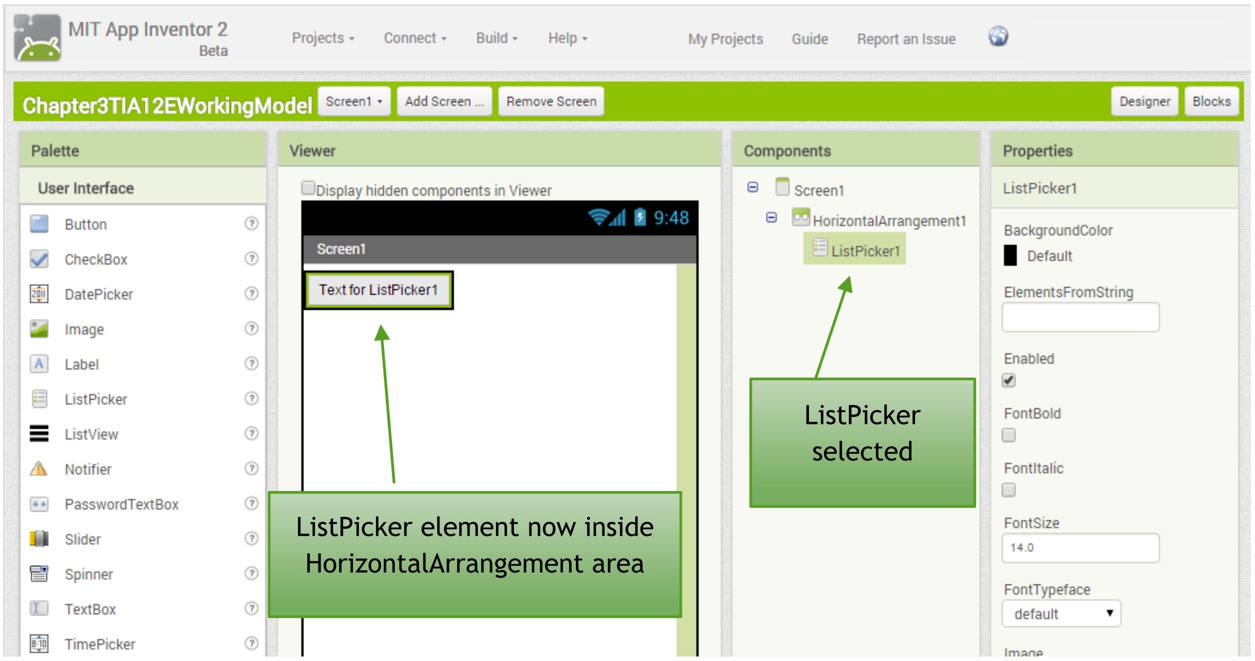
Part 1A - Creating the Screen Layout

1. In the **Palette**, select **Layout**. Click and drag HorizontalArrangement into the Viewer.

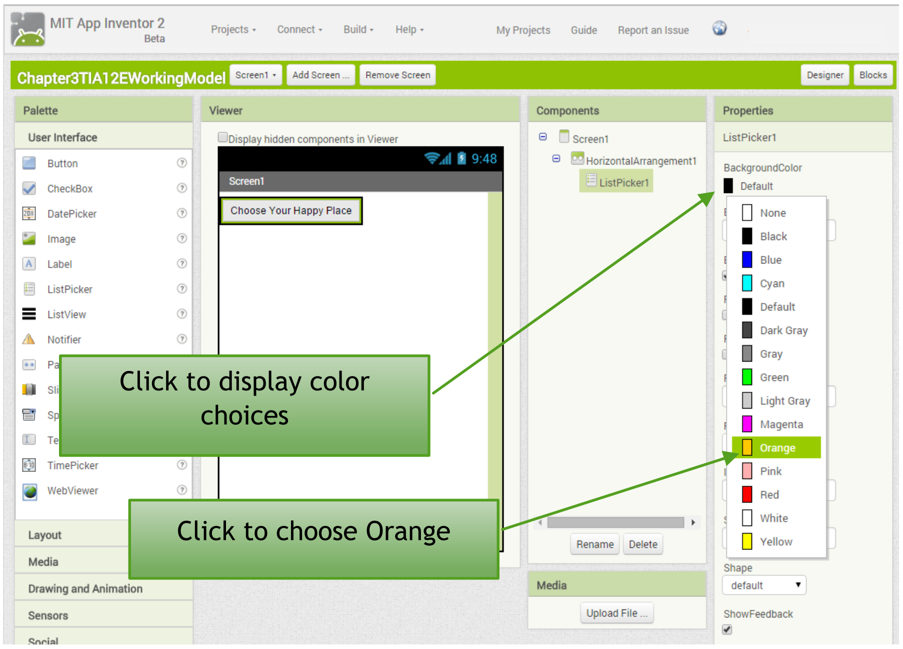
Tip: The HorizontalArrangement option creates an area on the screen in which all components you add to it will be arranged in a horizontal row.



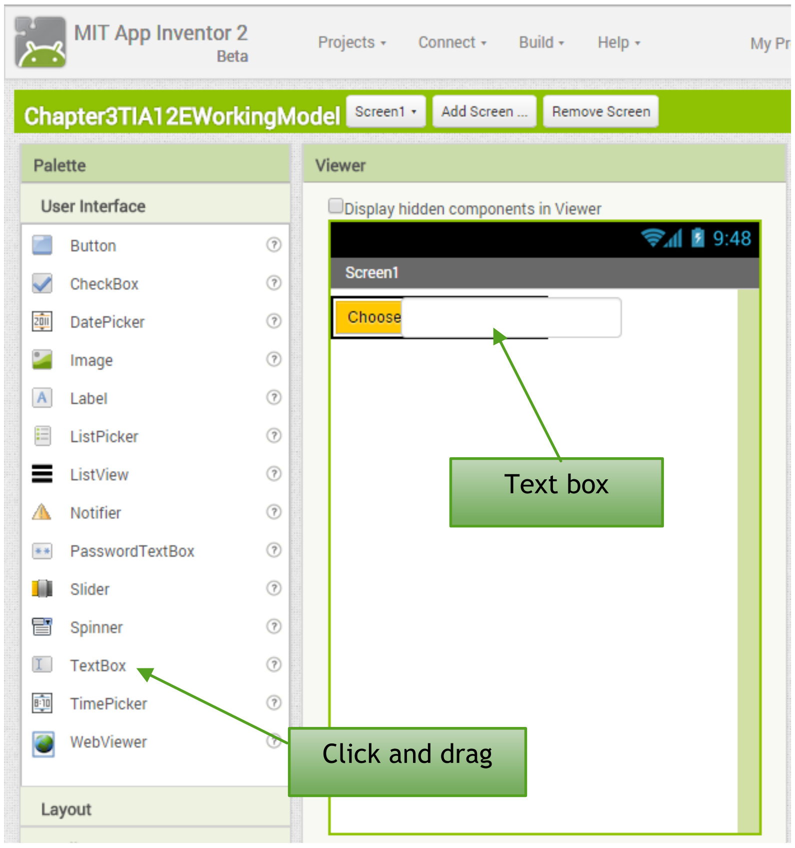
2. In the **Palette**, select **User Interface**. Then click and drag the **ListPicker** component into the HorizontalArrangement square. Your screen should look like this:



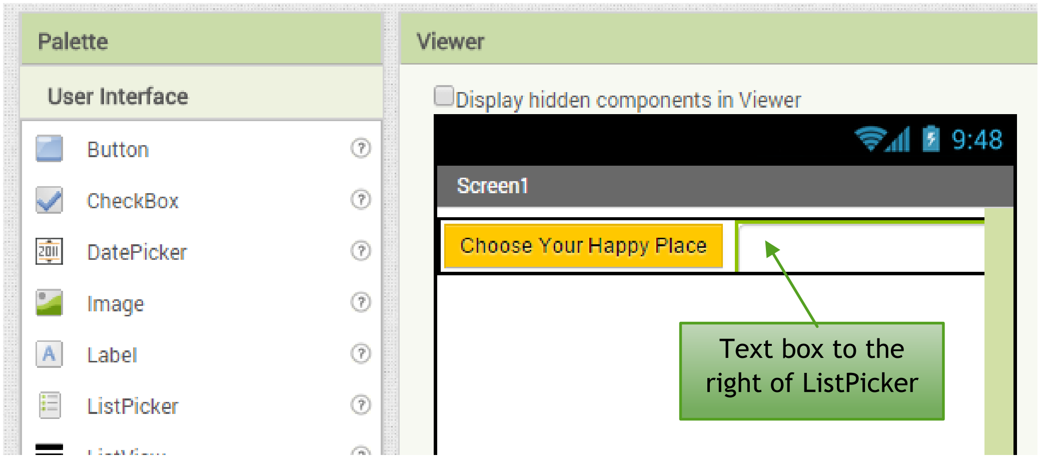
3. Make sure ListPicker1 is selected in the Components window. In the **Properties** section, under the **Text** field, replace the default text with: *Choose Your Happy Place*
4. In the **Properties** section, under the **BackgroundColor field**, click the black block to display the list of color choices. Select orange and notice the background color of the ListPicker changes to orange.



In the **Palette**, under **User Interface**, click and drag **TextBox** to the HorizontalArrangement box in the **Viewer**. Make sure to position the text box directly inside the HorizontalArrangement area overlapping the Choose Your Happy Place button, as shown below.

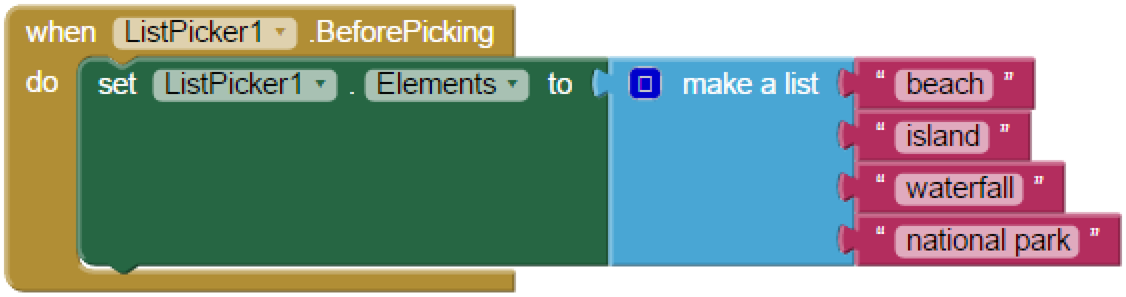


Note: Ideally, the TextBox should be positioned directly to the right of the ListPicker as shown below. If it is above or below the ListPicker, click the TextBox and move it to adjust its position. Or just choose a placement that suits your taste!

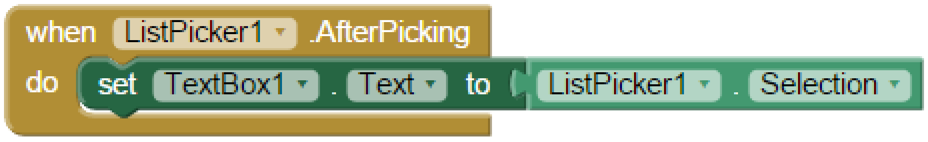


Part 1B – Using Blocks to Program Behaviors

You are now ready to work with the blocks that will control the action of our list picker. You need to assemble these blocks for the list picker:

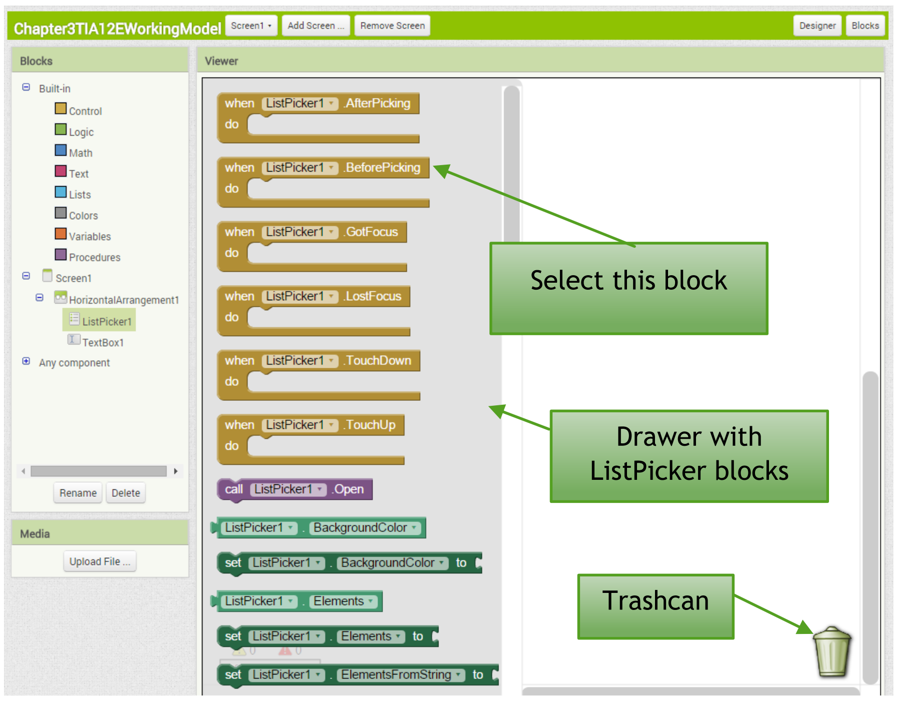


```
when ListPicker1 .BeforePicking
do  set ListPicker1 . Elements to  make a list  " beach "
                                                " island "
                                                " waterfall "
                                                " national park "
```



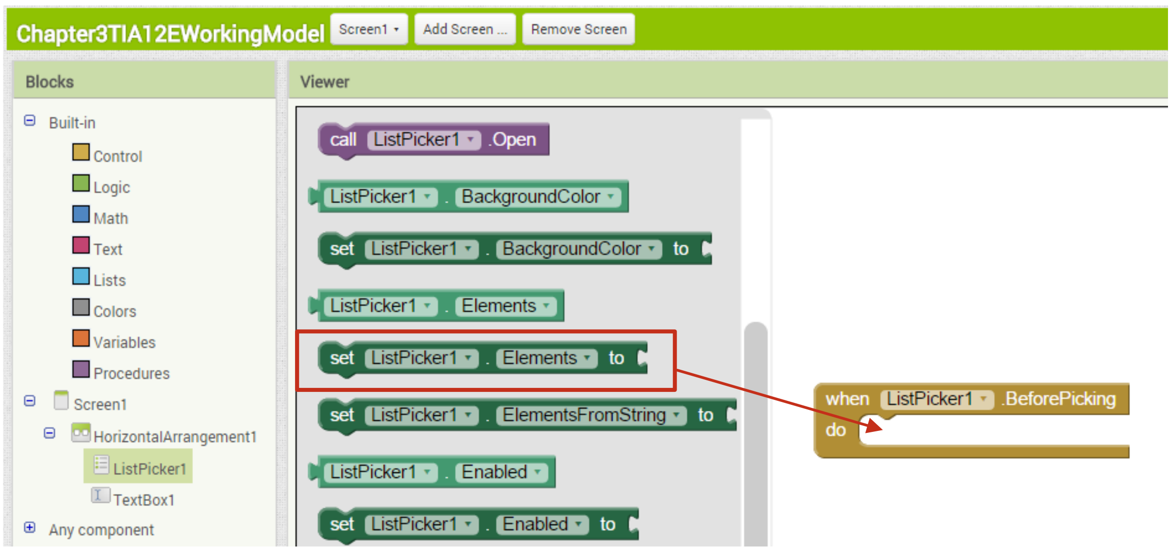
```
when ListPicker1 .AfterPicking
do  set TextBox1 . Text to  ListPicker1 . Selection
```

5. Switch to **Blocks** view.
6. In the **Blocks palette**, select the **ListPicker1** component to open its drawer. Click the event handler when ListPicker1 .BeforePicking block to add it to the **Viewer** window.


Chapter3TIA12EWorkingModel
Screen1
Add Screen ...
Remove Screen
Designer
Blocks
Blocks
Built-in
Control
Logic
Math
Text
Lists
Colors
Variables
Procedures
Screen1
HorizontalArrangement1
ListPicker1
TextBox1
Any component
Rename
Delete
Media
Upload File ...
Viewer
when ListPicker1 .AfterPicking do
when ListPicker1 .BeforePicking do
when ListPicker1 .GotFocus do
when ListPicker1 .LostFocus do
when ListPicker1 .TouchDown do
when ListPicker1 .TouchUp do
call ListPicker1 .Open
ListPicker1 . BackgroundColor
set ListPicker1 . BackgroundColor to
ListPicker1 . Elements
set ListPicker1 . Elements to
set ListPicker1 . ElementsFromString to
Select this block
Drawer with ListPicker blocks
Trashcan

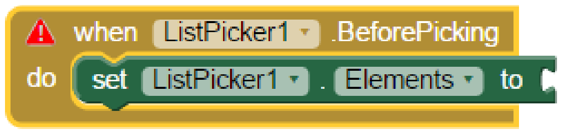
Tip: What happens if you drag the wrong block out onto the viewer? This is easily fixed. Just drag the block you don't need into the trashcan in the lower right-hand corner of the viewer and throw it away!

7. In the **Blocks palette**, select **ListPicker1**, then click and drag the set ListPicker1 . Elements to block from the drawer to insert it into the when ListPicker1 .BeforePicking block. They will click (with an audible sound) together like puzzle pieces.



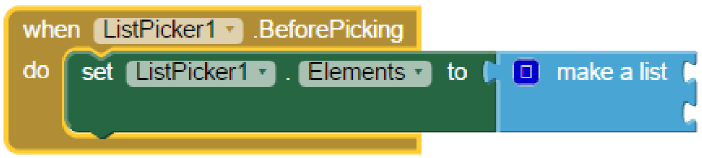
```
call ListPicker1 .Open
ListPicker1 . BackgroundColor
set ListPicker1 . BackgroundColor to
ListPicker1 . Elements
set ListPicker1 . Elements to
set ListPicker1 . ElementsFromString to
ListPicker1 . Enabled
set ListPicker1 . Enabled to

when ListPicker1 .BeforePicking
do
```

The block should now look like this:



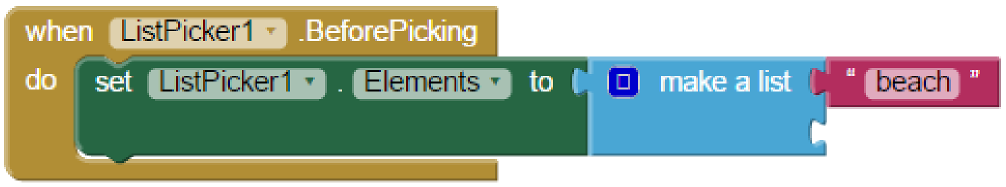
```
when ListPicker1 .BeforePicking
do  set ListPicker1 . Elements to
```

8. In the **Blocks palette**, under **Built-in**, select **Lists**. From the **drawer**, click and drag the make a list block and plug it into the open socket on the right side of the set ListPicker1 . Elements to block. The block should now look like this:



```
when ListPicker1 .BeforePicking
do  set ListPicker1 . Elements to  make a list
```

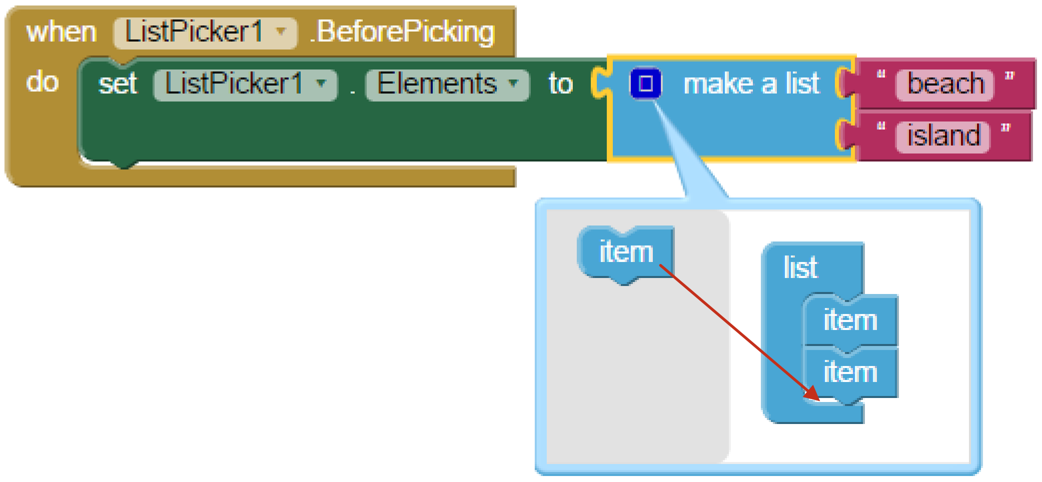
9. In the **Blocks palette**, under **Built-in**, select **Text.** From the drawer, click and drag out a " " block and connect it to one of the item sockets in the make a list block. Now click directly on the space in the " " block. You can now enter text into this block. Type in ***beach*** (the first happy place choice), then click outside the block. Your block should now look like this:


when ListPicker1 .BeforePicking
do set ListPicker1 . Elements to
make a list
" beach "

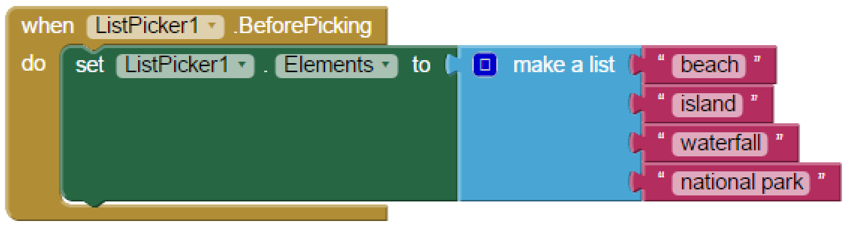
10. Drag out another “ ” block and place it in the open item socket on the make a list block. Type *island* in this block.

<span>Alert:</span> After you plug in two text blocks, there are no more open sockets left. However, you want to add two more choices (*waterfall* and *national park*) to your list of choices. You need to add additional sockets to the block!

11. To create more sockets, click the dark blue button on the make a list block. From the box that displays, drag the **item icon** on the left to the **list icon** on the right. This will add another choice to list. Do this twice to make two more sockets available.


when ListPicker1 .BeforePicking
do set ListPicker1 . Elements to
make a list
" beach "
" island "
item
list
item
item

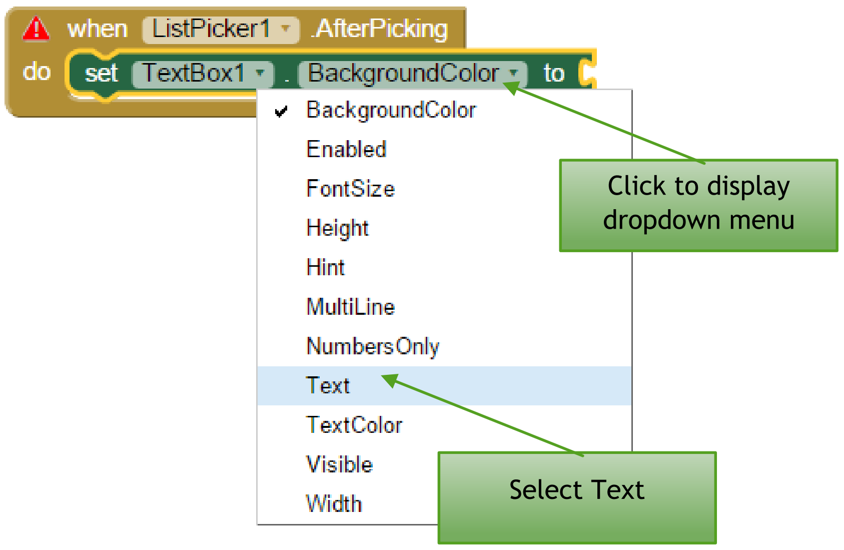
12. Drag out two more “ ” blocks and add them to the new sockets on the make a list block. Insert ***waterfall*** into one “ ” block and ***national park*** into the other. Your block should now look like this:



```
when ListPicker1 .BeforePicking
do  set ListPicker1 . Elements to  make a list  “ beach ”
                                                “ island ”
                                                “ waterfall ”
                                                “ national park ”
```

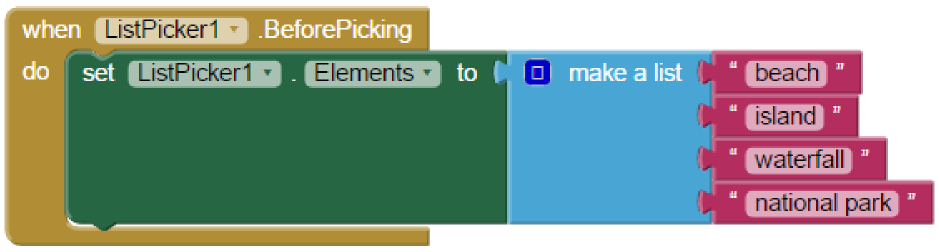
Note: This block will allow the user to make a selection of their “Happy Place” from one of the four choices you have provided. Now you want the selection the user makes to appear in your text box.

13. In the **Blocks palette**, select **ListPicker1**. From the drawer, select the when ListPicker1 .AfterPicking block and drag it into the **Viewer**.
14. In the **Blocks palette**, select **TextBox1**. From the drawer, select set TextBox1 . BackgroundColor to block and insert it into the when ListPicker1 .AfterPicking block.
15. Click the BackgroundColor option and from the dropdown list, select Text. This changes it to a set TextBox1 . Text to block.

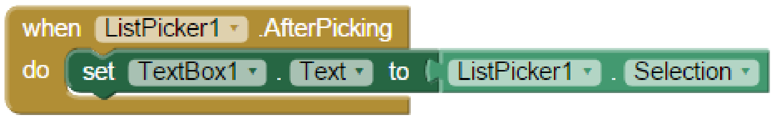


```
when ListPicker1 .AfterPicking
do  set TextBox1 . BackgroundColor to
```

16. In the **Blocks palette**, click **ListPicker1**. From the drawer, select the ListPicker1 . Selection block and insert it into the open socket on the set TextBox1 . Text to block. Your blocks should now look like these:



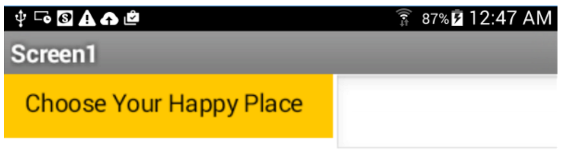
```
when ListPicker1 .BeforePicking
do  set ListPicker1 . Elements to  make a list  " beach "
                                                " island "
                                                " waterfall "
                                                " national park "
```

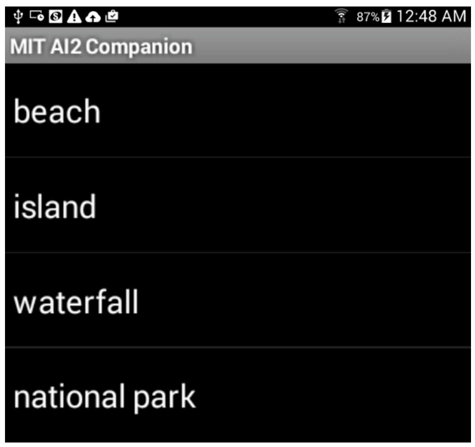


```
when ListPicker1 .AfterPicking
do  set TextBox1 . Text to  ListPicker1 . Selection
```

Note: The first block sets up our list choices for the user. The second block displays what the user chooses in the text box. Screenshots from an Android device are shown below.

Test out the app so far on your Android device. Click the Choose Your Happy Place **button (left image below) and you should see a list of four choices (right image below). Select one and ensure the result is displayed in the text box.**

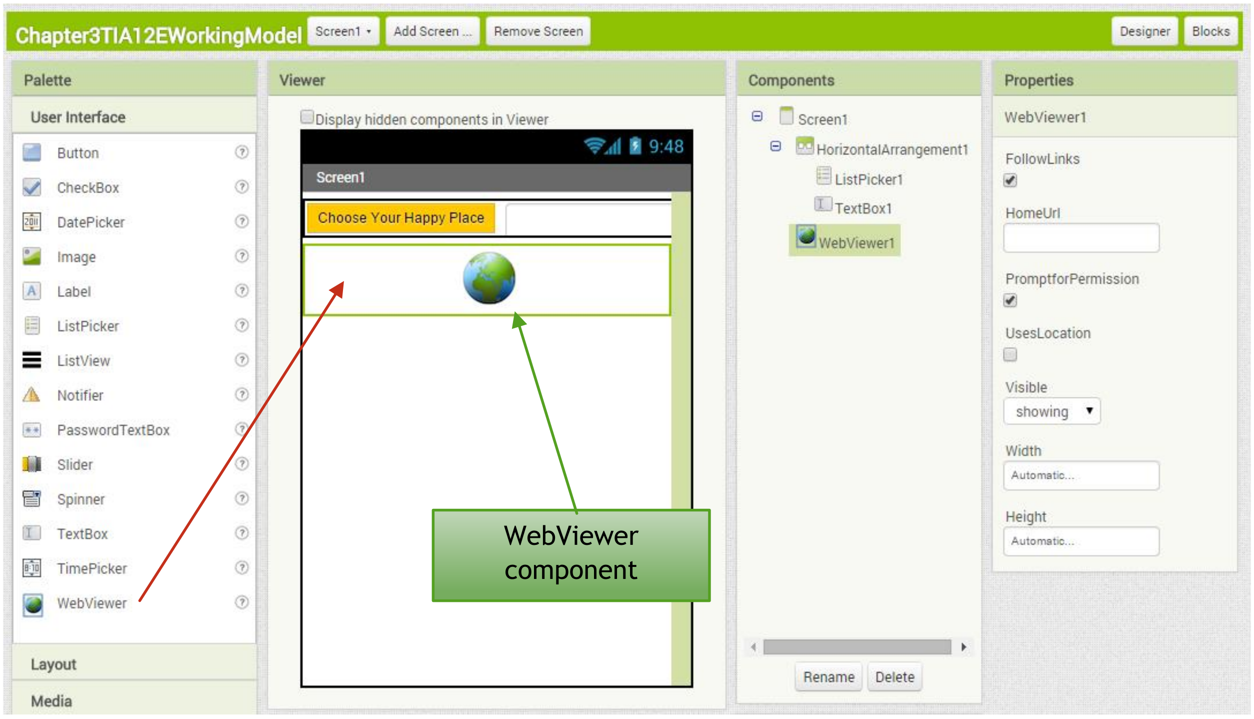




Part 2 – Searching for Images on the Web Using Web Viewer

When our users select their happy place, you want them to see pictures of that type of place. To do this, you can use the WebViewer component of App Inventor. WebViewer is not a full-featured web browser, but it does provide a quick means for displaying web pages and even filling in web forms.

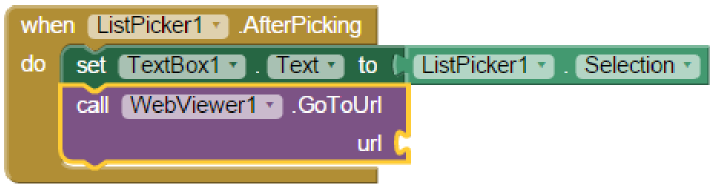
1. Click the **Designer button** to return to the Designer view.
2. In the **Palette**, under **User Interface**, click and drag the WebViewer component onto the **Viewer** underneath the ListPicker.



3. Click the **Blocks button** to switch back to the **Blocks** view.

Note: You want the WebViewer to display the user's happy place choice after they pick a place from the list. So you can utilize the event handler when ListPicker1 .AfterPicking **block that you already have to access the WebViewer. Event handler blocks are mustard colored and they respond to events or actions that occur.**

4. In the **Blocks palette**, select **WebViewer1**. From the drawer, click and drag the call WebViewer1 .GoToUrl block and insert it into the when ListPicker1 .AfterPicking block after the set TextBox1 . Text to block as shown below:



```
when ListPicker1 .AfterPicking
do  set TextBox1 . Text to  ListPicker1 . Selection
    call WebViewer1 .GoToUrl
                         url
```

Understanding Communications with another Computer

When you need to retrieve information from a web site, your computer needs to “talk” (communicate) with another computer. For computers to easily communicate, protocols (set methods of communication) have been developed. When both computers are using the same protocol, they can understand requests and provide information easily…just as two people who are speaking the same language would do.

You are going to search for images on Flickr.com. Your Android device needs to communicate with the computer that is hosting the website www.flickr.com. The HTTPS (HyperText Transport Protocol Secure) protocol is commonly used for communicating with secure web servers like the one hosting Flickr.com.

When you open up a browser and navigate to www.flickr.com, there is a search box at the top of the screen. If you were to enter “island” in the search box, the computer hosting Flickr.com runs a program that allows searching for specific images related to key words. Essentially, your program will be asking the Flickr web service for information, similar to asking a question (“show me images of islands”), and receiving a response. Asking a computer program to provide you with specific information is known as a **query**. The response to your query will be a URL that points to images that result from your query…in this example pictures of islands. Any web page where you go and make a choice (such as searching for a product on Amazon.com) will create a similar type of URL that directs you to the information you requested.

In order to display images relating to each place in your app, you need to determine the search URL required by Flickr.com. Then you can put that search URL into WebViewer so the app can do the appropriate search.

Open the Flickr.com website in a browser and type ***island*** into the search box. Examine the URL that Flickr uses to search for island (as shown in the top of your browser):

https://www.flickr.com/search/?q=island

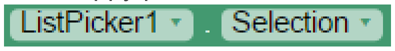
Change the search term to beach and note the URL. The end of the URL changes to reflect the search term. Therefore, for WebViewer to work properly, you need to recreate this URL substituting your user’s search term at the end after the equal sign.

In computer programming, a ***string*** is a sequence of characters. Strings are often used to represent literal text (including symbols, number and spaces). The URL above is a string. Computer programmers often manipulate strings to suit their needs. One common procedure is combining two strings by using a join command. This takes the contents of one string and appends it to the end of another string. This is exactly how you will generate the URL you need for your search.

Let’s examine the first part of the URL, which will stay the same for any search you want to do on images at Flickr: **https://www.flickr.com/search/?q=**

The first part of the URL is composed of three parts:

a) Protocol – The standard communications method used by your computer to request information from another computer (in this case the protocol is https).
b) Computer – The computer from which information is being requested. It is identified by its web site name (flickr) and domain (.com).
c) Web service – The web service provided on the web site you are querying that is designed to provide the information you are requesting (in this case a search for images).

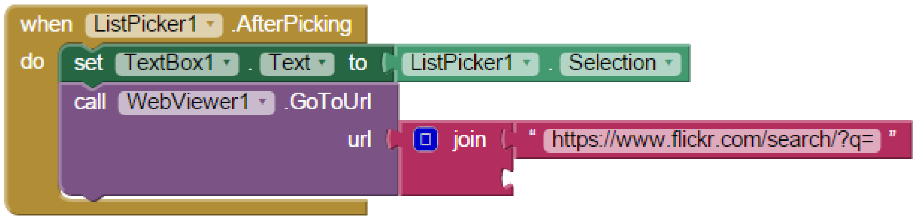
What will vary is the search term as that is the “happy place” the user selects from our list. So you will use a join command to add the result from the 

block to the first part of our URL.

5. In the **Blocks palette**, under **Built-in**, select **Text**. From the drawer, click and drag out a join block and insert it in the open socket on the call WebViewer1 .GoToUrl block.

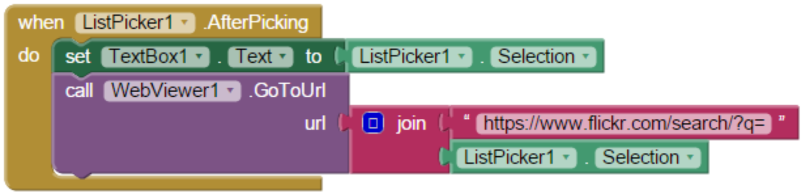
Note: The join block allows us to combine two (or more) strings into one.

6. In the **Blocks palette**, under **Built-in**, select **Text**. From the drawer, drag out a “ ” block and insert it into the first socket on the join block. Click on the “ ” block and insert the following text in the block: **https://www.flickr.com/search/?q=**



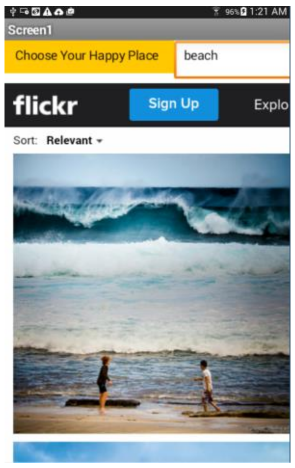
```
when ListPicker1 .AfterPicking
do  set TextBox1 . Text to ListPicker1 . Selection
    call WebViewer1 .GoToUrl
                         url  join  " https://www.flickr.com/search/?q= "
```

7. In the **Blocks palette**, click **ListPicker1**. From the drawer, select the ListPicker1 . Selection block and insert it into the open socket on the join block. This is telling the app to join the selection from the ListPicker to the end of the URL. Your block should now look like this:



```
when ListPicker1 .AfterPicking
do  set TextBox1 . Text to ListPicker1 . Selection
    call WebViewer1 .GoToUrl
                         url  join  " https://www.flickr.com/search/?q= "
                                    ListPicker1 . Selection
```

Now test your app and ensure that the WebViewer is displaying the appropriate pictures based on the “happy place” selected. Here is a screenshot from an Android device running this app when the happy place selected was beach.

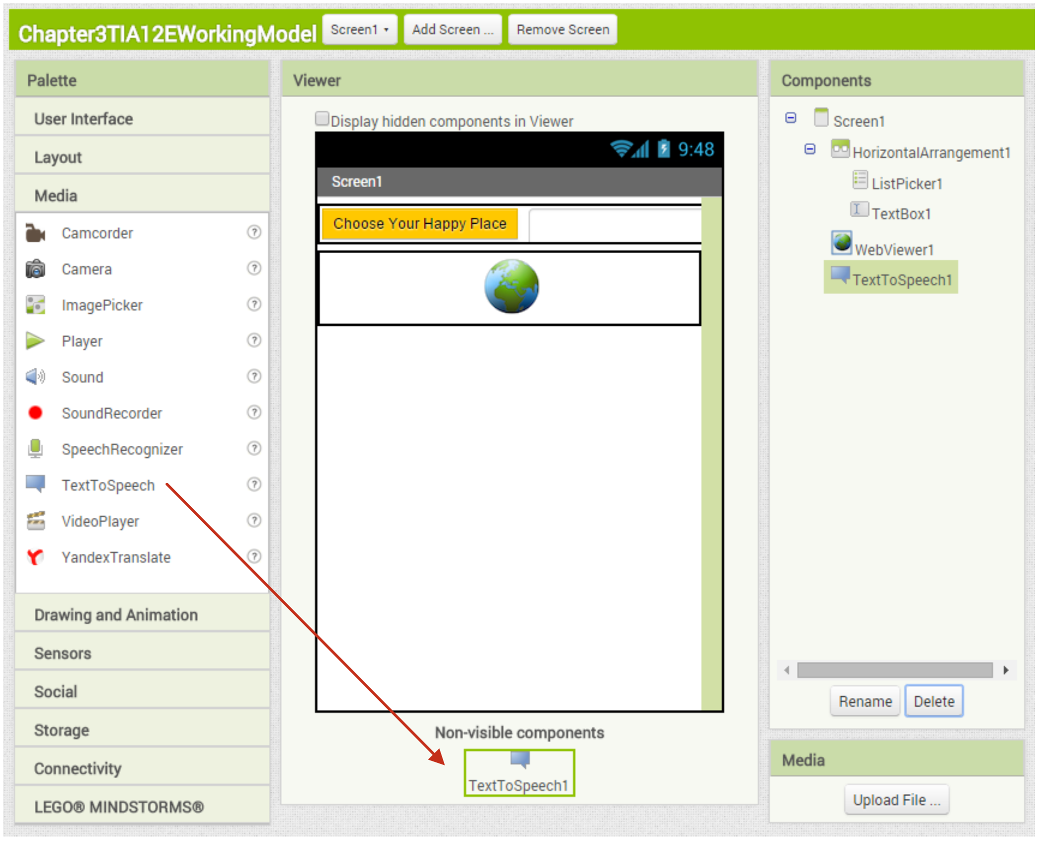


Part 3 – Using Text to Speech

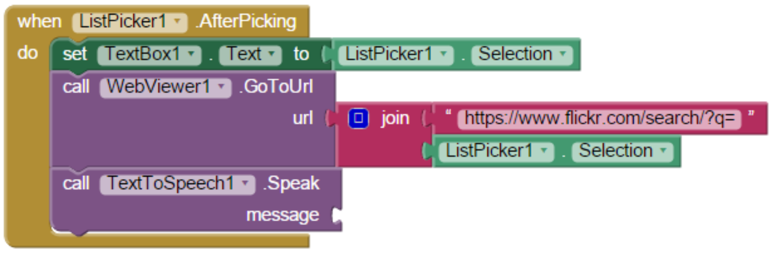
Aside from visual feedback, users often appreciate auditory feedback when they take an action in an app. In this section, you will modify your app to have your Android device say aloud the "happy place" to which you are going.

1. Click the **Designer button** to return to the **Designer** interface.
2. In the **Palette**, under **Media**, select the **TextToSpeech** component and drag it out onto the Viewer window.

Alert: Notice since this is a non-visible component (the user can't see it on the screen), it is displayed at the bottom of the Viewer window in the Non-visible components section.



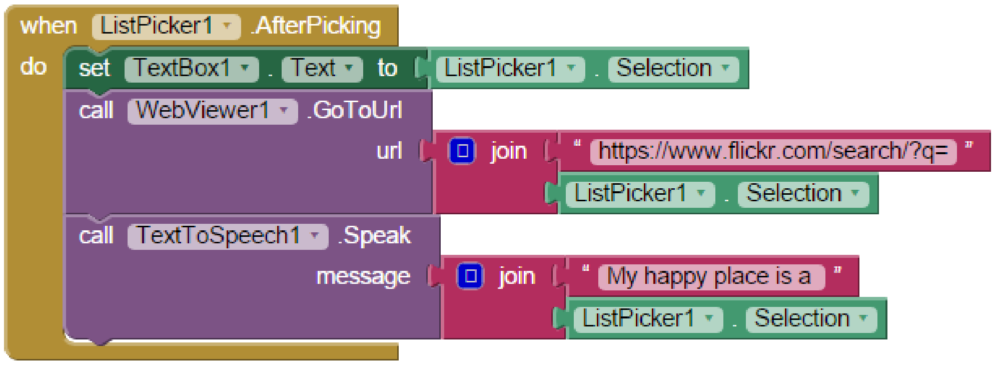
3. Click the **Blocks button** to switch to the **Blocks** view.
4. In the **Blocks palette**, click **TextToSpeech1**. From the drawer, select the call TextToSpeech1 .Speak block and insert it into the when ListPicker1 .AfterPicking block below the call WebViewer1 .GoToUrl block.



```
when ListPicker1 .AfterPicking
do  set TextBox1 . Text to  ListPicker1 . Selection
    call WebViewer1 .GoToUrl
                        url  join  " https://www.flickr.com/search/?q= "
                                   ListPicker1 . Selection
    call TextToSpeech1 .Speak
                    message
```

Note: Since the TextToSpeech component converts text into spoken words, you can again use strings…this time to create the phrase you want spoken aloud.

5. In the **Blocks palette**, under **Built-in**, select **Text**. From the drawer, click and drag out a join block and insert it in the open socket on the call TextToSpeech1 .Speak block.
6. Reopen the **Text drawer**. Click and drag out a " " block and insert it into the first socket on the join block. Click on the " " block and insert the following text in the block: **My happy place is a**
7. In the **Blocks palette**, select **ListPicker1**. From the drawer, click and drag the ListPicker1 . Selection block and insert it into the open socket on the join block. Your block should now look like this:



```
when ListPicker1 .AfterPicking
do  set TextBox1 . Text to ListPicker1 . Selection
    call WebViewer1 .GoToUrl
                       url  join "https://www.flickr.com/search/?q="
                                 ListPicker1 . Selection
    call TextToSpeech1 .Speak
                   message  join "My happy place is a"
                                 ListPicker1 . Selection
```

Test your app (make sure your volume is turned up) and your Android device should now speak the name of the happy place selected. Now that you've worked through the project once, go back and modify the components to better suit your own preferences, or try one of the ***Extensions to This Project*** below.

Extensions to This Project

1. Modify the app so that the user can type in their own happy place instead of picking from a list.
2. Modify the app so that the user can speak the name of their happy place instead of typing it in or picking from a list. You will need to use the **SpeechRecognizer** component to accomplish this.
3. Modify the app to search for only videos from Flickr, not images. **Hint**: try exploring the Advanced Search options on the Flickr website to see what kind of URL it builds when the search is restricted to videos.

Resources

- User Guide for App Inventor 2
- Guide to Understanding Blocks
- MIT App Inventor Support Forum
- Beginner Tutorials (with videos)

MIT App Inventor is a blocks-based programming tool that allows everyone, even novices, to start programming and build fully functional apps for Android devices. Google's Mark Friedman and MIT Professor Hal Abelson co-led the development of App Inventor while Hal was on sabbatical at Google. App Inventor runs as a Web service administered by staff at MIT's Center for Mobile Learning - a collaboration of MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) and the MIT Media Lab. MIT App Inventor supports a worldwide community of nearly 3 million users representing 195 countries worldwide. App Inventor is an open-source tool that seeks to make both programming and app creation accessible to a wide range of audiences. App Inventor is the property of the Massachusetts Institute of Technology (MIT) and the work licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. For more information on App Inventor, go to MIT App Inventor About Us page.