Authoring Playground Books with Bluefruit for iOS

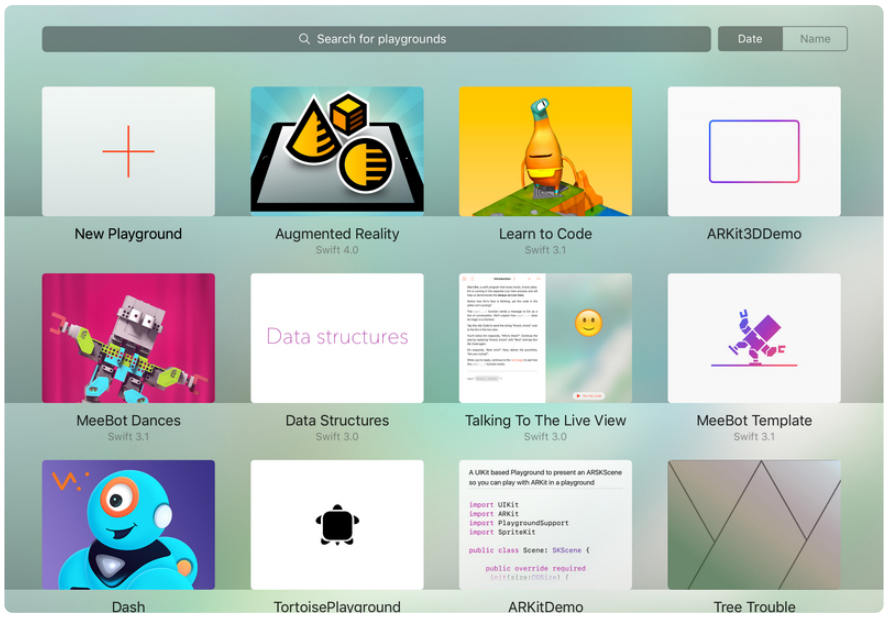
Created by Trevor Beaton

https://learn.adafruit.com/create-a-swift-playgroundbook-with-bluetooth-le

Last updated on 2024-06-03 02:14:50 PM EDT

Table of Contents

Overview



Ever thought about creating a Swift playground book (https://adafru.it/BtG)? If so, this is the learn guide for you!

In the process of creating a Swift Playground book for use with Adafruit's Bluefruit devices, I noticed that there's very little public information available on the subject. Hopefully this guide will help share what I've learned with the world.

In this learn guide we will:

• Give you a playground book template to get you started on project.

• Show you how to get started authoring your own playground book with bluetooth capabilities.

• Explain to you how a Playground book is structured.

What you should know before beginning:

• You should have a basic understanding of the **Swift Programming Language** & **iOS**.

• You should be familiar with the **CoreBluetooth Framework**.

If you're not familiar with CoreBluetooth, check out the Create Bluetooth LE App for iOS (https://adafru.it/CgZ) learn guide that shows you the fundamentals of the CoreBluetooth framework and transferring data.

To create a Playground book you'll need:

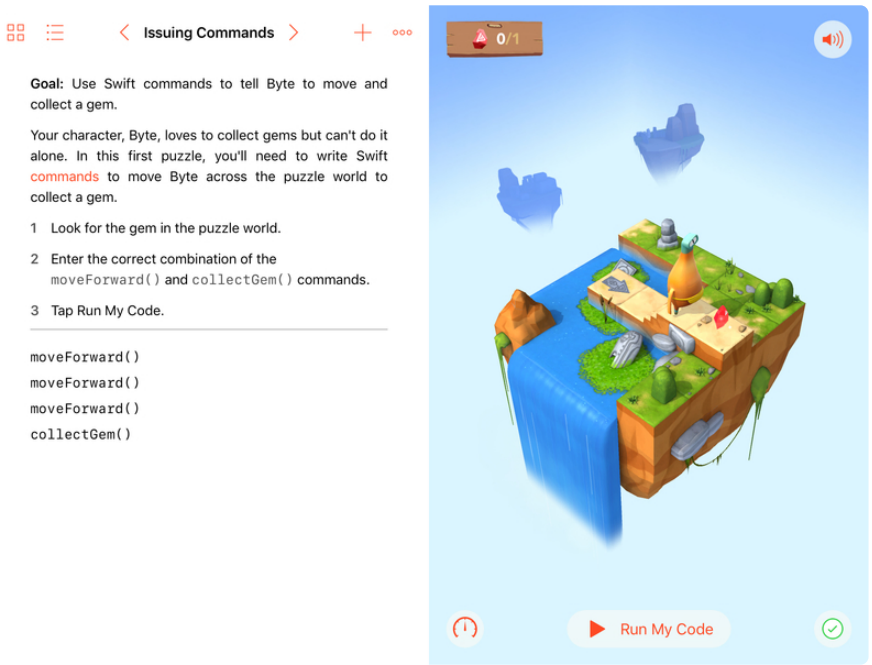
- A Mac computer
- Xcode version 9.0 (or newer)
- Swift Playgrounds App
- An iPad running iOS 11 (or newer)

If all the requirements are met, let's get started!

Currently, Swift Playgrounds is only available for the iPad.

What is a Playground Book?

Apple's Swift Playgrounds (https://adafru.it/Btl) is an app available for iPad that makes learning the Swift programming language fun and exciting with code through a collection of digital books called "Playground Books" that are available for download.



Playground books are split into **chapters**, which are further divided into **pages**.

Don't confuse **Swift Playgrounds** with **Xcode Playgrounds**. Though they sound similar, they're very different things :)

Bluefruit Playground Template & Book

Downloading and opening a Playground Book

I've created an empty playground book with an Always-On Live view and a debug log setup (You can delete the debug log if you wish). Instead of creating a playground book from scratch, I'd recommend downloading the template playground book that we provide.

You can download the blank playground book template here:

Download Bluefruit Swift Playground Book Template

https://adafru.it/AI2

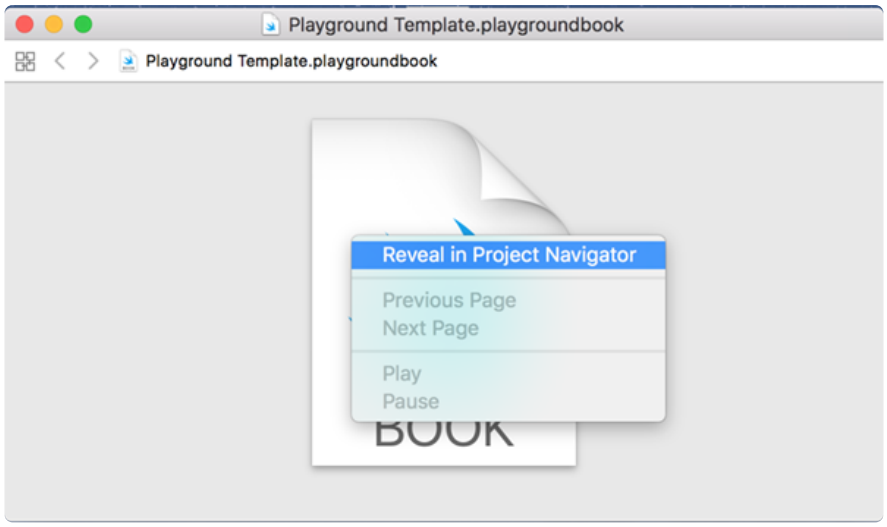
As a reference, I've also created a Swift Playground book for use with Adafruit's **Bluetooth LE** Devices. Download it here:

Download Bluefruit Playground Book

https://adafru.it/AI3

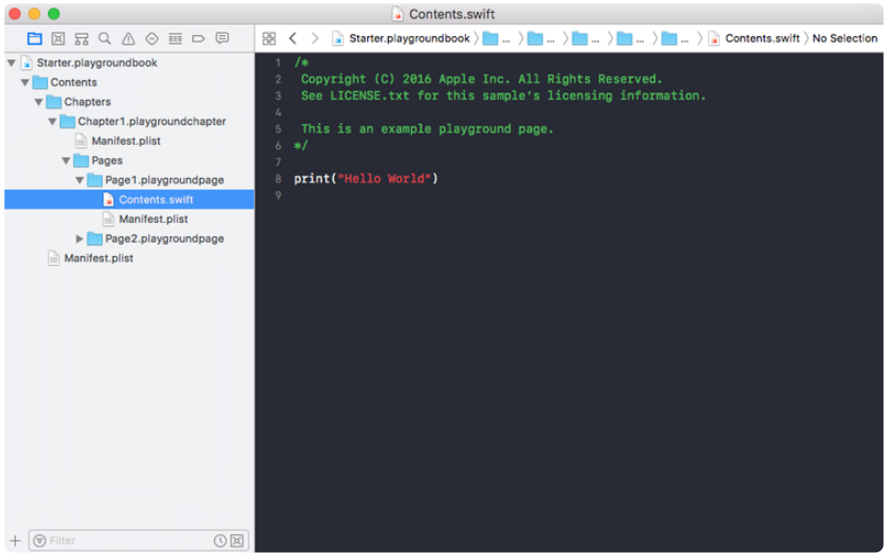
To open your playground book:

• Locate the Playground Template.playgroundbook you downloaded and open the file.



• Once the file is open, right-click the playground book icon and select "**Reveal in Project Navigator**".

• Now, you should see the File Hierarchy on the left side of the project window.



• In the file hierarchy, open **Contents -> Chapters -> Chapter 1 -> Pages -> Page 1 -> Contents.swift**

This is where you'll be doing some of your programming.

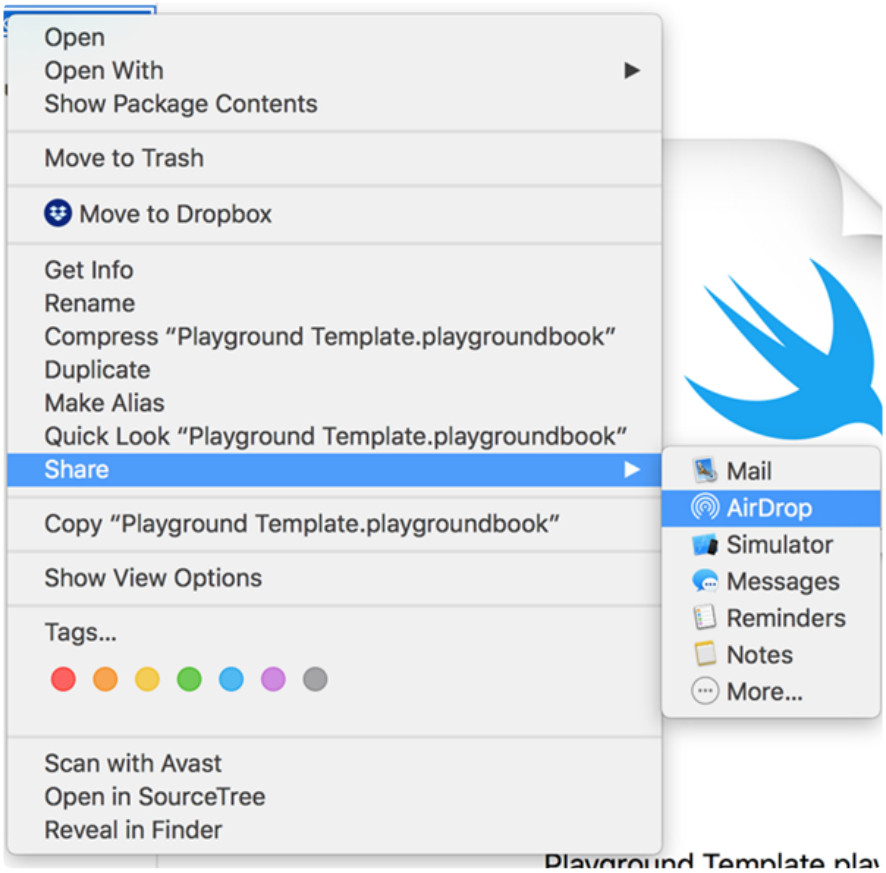
Before starting, be aware that there is no code-completion when opening a selected file in Xcode. There will be a bit of trial and error, but we'll move through that.

Now to test our playground book, we'll need to send it to the iPad. The easiest way to send the playground book in my opinion would be sending the Playground book to the iPad using AirDrop.

Sending a Playground Book

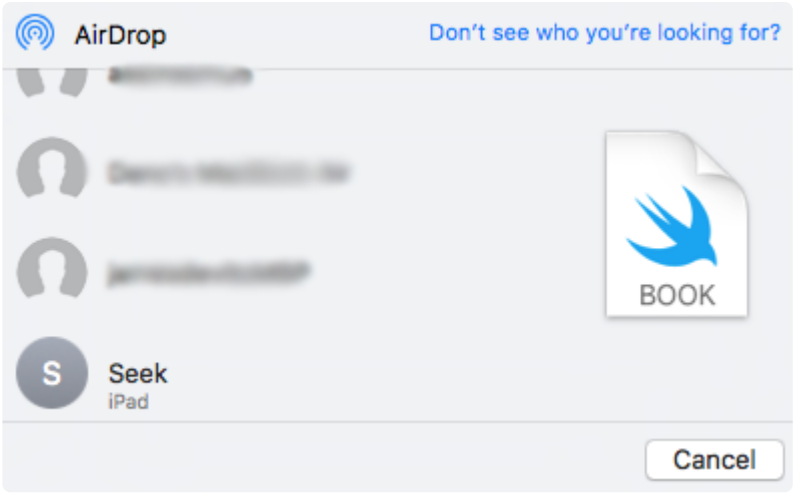
Sending your playground book to the iPad using AirDrop

• Right-click the playground book in **Finder**, then scroll down to Share then select AirDrop.

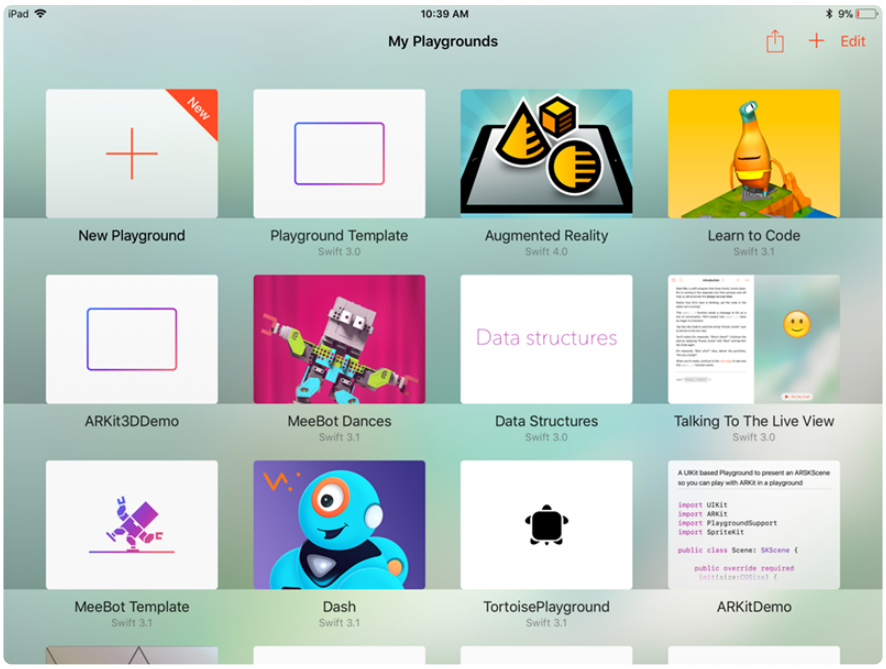


A pop-up window will show up. Select your iPad in the pop-up window. You may then see a popover window on the iPad asking you to accept the file transfer. Press OK.

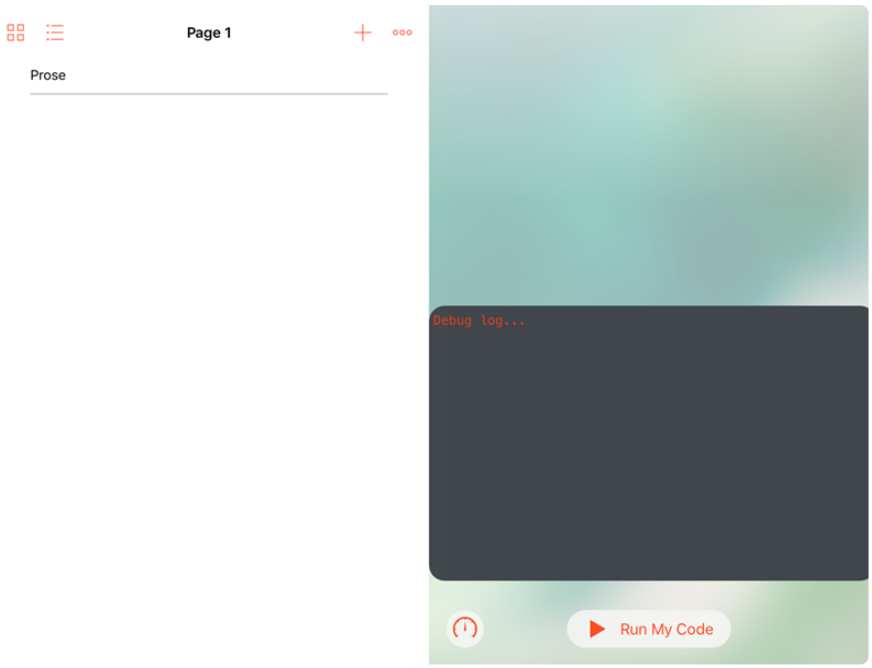
Don't forget to enable the Bluetooth in your iPad.



On the iPad, in the Swift Playgrounds app, your new playground book should be the first in the collection of playground books shown. It will be titled **Playground Template**.



Open up your playground book to see it's contents. Of course this book is empty, there aren't any contents.



If all goes well, you should have your playground book template running on your iPad! With Playground books you can be as creative as you wanna be. You can take advantage of real iOS frameworks like UIKit or CoreMotion to bring your ideas to life.

In the next page, we'll go more in-depth on the structure of the playground book.

Playground Book Structure

Playground Book Package

Playground books are document based app with a file and folder specific structure. The playground book file structure is similar to a regular book's structure.

To put it simply: the Playground book holds chapters, and those chapters hold pages and that's where you'll be writing your code to create your Playground book.

For each level in this book structure, there is a specific extension to accompany it:

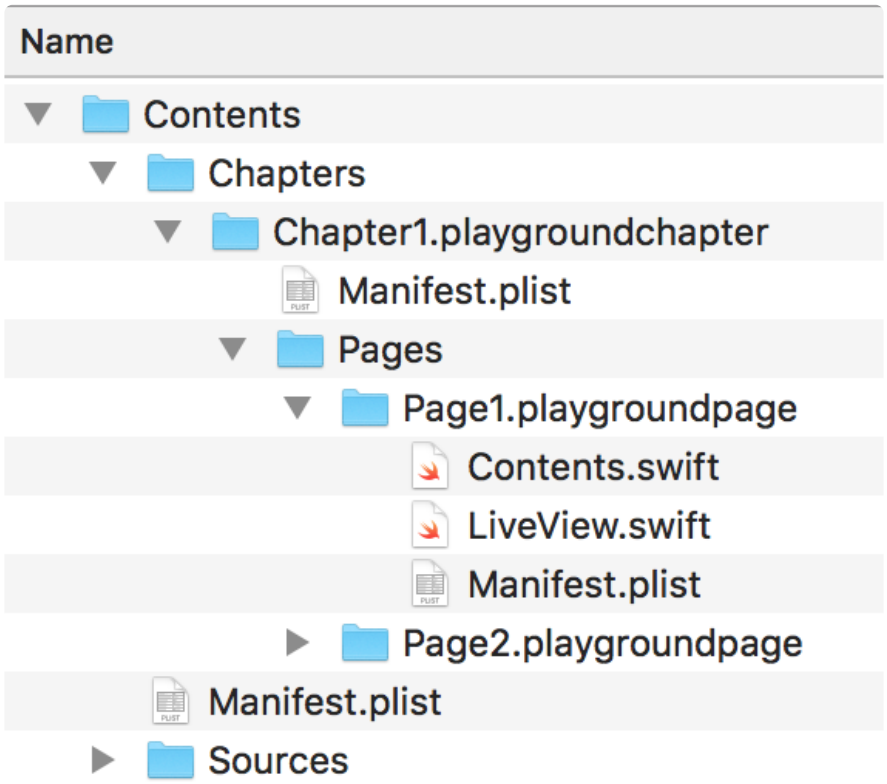
- The **book** folder has a **.playgroundbook** extension. It is the top-level folder for the playgroundbook.

- The **chapters** folder has a **.playgroundchapter** extension.

- The **pages** folder has a **.playgroundpage** extension.

And for each book level, you'll need a **manifest.plist** file.

Below you'll see the File Hierarchy. This illustrates the file structure of how files and folders for the playground book package should be formatted.



Source Folders

The **Sources** folder is where you'll put your global Swift files that you'd want to compile and make available to every page. You don't have to import anything to make this work. Anything that's declared as a public source will be ready as soon as you open your playground page.

Chapter Folders

The **Chapter** folder is what you probably expected. It's the folder that holds all of the chapters in the playground book which also contain one or more playground pages. Each page within the folder contains the content you'll interact with when the playground book app is used.

Playground Book Manifest

Manifest files are property lists, dictionaries of keys and values that Swift Playgrounds uses to determine how pages with documents are supposed to behave. The manifest.plist provides information such as the location of icons being used in a document, the documents development targets, and list of chapters in the playground book or just the name of the page .

The **manifest.plist** file also stores:

- Metadata such as book version and book target platform
- Order of chapters
- User-viewable name for the book

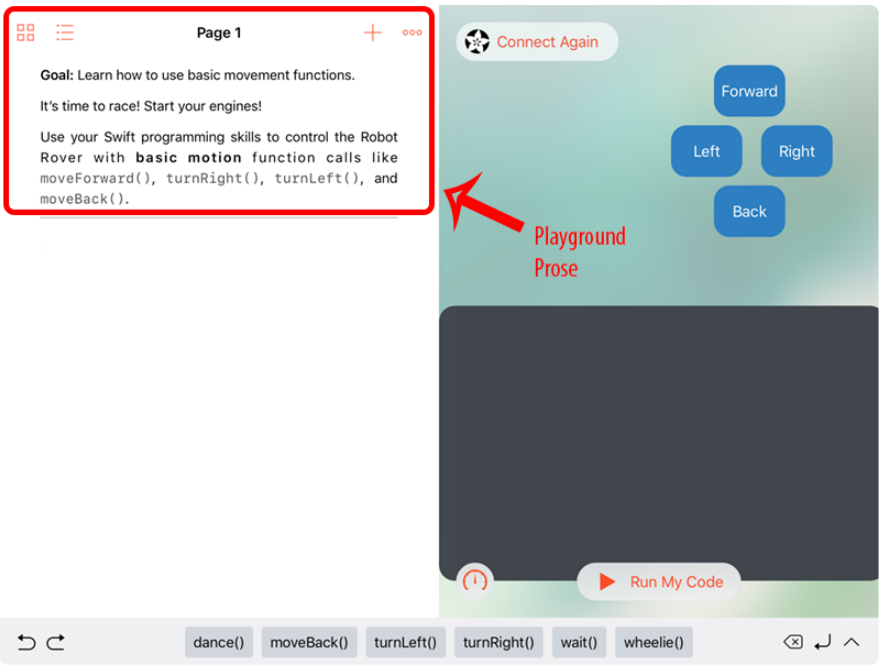
As you can see in the image above, a **manifest** file is located in the **Contents** folder inside the top-level playground book folder and also at every book level.

There are three major files in a Playground page folder. There's the **manifest.plist**, **LiveView.swift** file, and the **Contents.swift** file. Next, we're going to take a look at the Contents.swift file.

Playground Markup and Source Code

Contents.swift is the source file where your users interact with the playground source code.

You can also use it to add **playground prose** that's shown to your users using special markup comments within that swift file. The **Playground Markup Prose** describes what the user is to do - providing objectives, goals, and instructions.



When you open the Swift Playground Book for Bluefruit (https://adafru.it/BtJ), you'll notice that the **Playground prose** is separate from the Playground source code. To create a prose we'll add a **Block Comment Marker** in the **Contents.swift** file.

Block Comment Marker

Using a **Block comment marker** helps you to enclose multiple lines of text containing markup delimiters. Here's what that looks like:

```
/*:

"Your Content"

*/
```

Here's an example of a Block comment marker in the Contents.swift file of the Swift Playground book for Bluetooth:

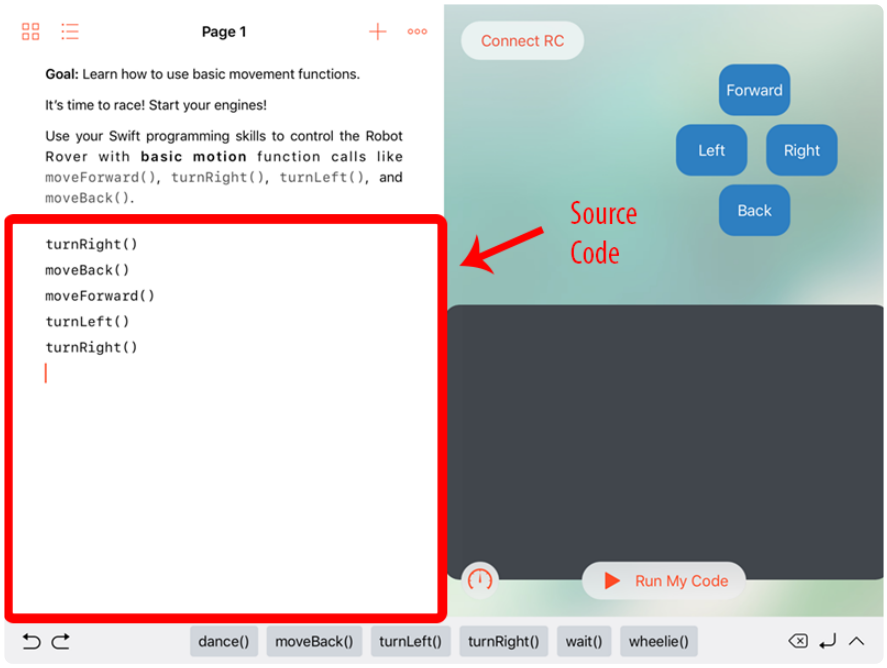
```
/*:
**Goal:** Learn how to use basic movement functions.

It's time to race! Start your engines!

Use your Swift programming skills to control the Robot Rover with **basic motion**
function calls like `moveForward()`, `turnRight()`, `turnLeft()`, and `moveBack()`.

*/
```

Once you've added the block comment marker to your playground book, you'll notice lines between the opening and closing comments that you've added. The Block comment maker does not interfere with the process of run your code.

Now let's look at the Playground Source Code.

Playground Source Code

The Playground Source Code is where your users interact with your playground book adding their code. This is where your users enter their code. Once they've added their code, they'll hit the "**Run My Code**" button to execute what they've written.



Hidden Code

You can hide code that doesn't relate to the content of your playground page to do things like importing frameworks or code completions ... basically anything that you don't want the user to see or manipulate.

Hidden code is still executed when the playground is run and all of the public symbols are accessible on the playground page.

Hidden code can be placed between `//#-hidden-code` and `//#-end-hidden-code`. Here's an example of that in the Contents.swift file.

```
/*:
 **Goal:** Learn how to use basic movement functions.

 It's time to race! Start your engines!

 Use your Swift programming skills to control the Robot Rover with **basic motion**
function calls like `moveForward()`, `turnRight()`, `turnLeft()`, and  `moveBack()`.
```

```
 */

//#-code-completion(everything, hide)
//#-code-completion(identifier, show, moveForward(), turnLeft(), moveBack(),
turnRight(), wait(), wheelie(), dance())
//#-hidden-code
import Foundation
import PlaygroundSupport

setup()

//#-end-hidden-code

//#-editable-code

moveForward()

//#-end-editable-code

//#-hidden-code

//#-end-hidden-code
```

Editable Areas

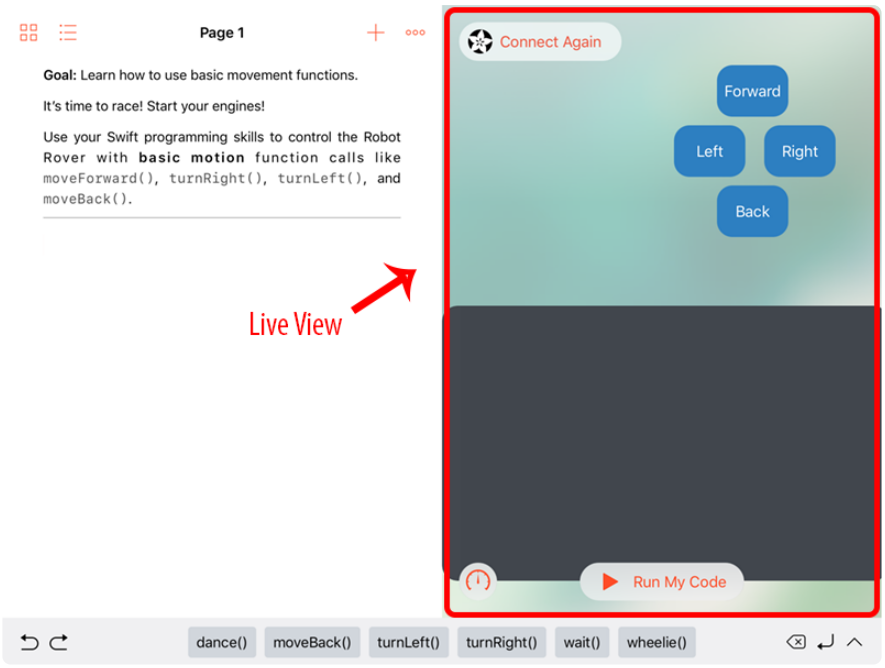
Editable areas work the same way the Hidden Code does. You'll need to add inline editable areas within your playground book so that you can receive code entered by the user. It's very simple to add an editable area, here's how you add one:

```
//#-editable-code

"Add your code here"

//#-end-editable-code
```

Runtime Errors

Swift Playgrounds will display runtime errors that occur in the Playground book. Swift Playgrounds is able to determine which line of code caused the execution to crash and then highlights the line with the text that's provided.

Playground Live View



Your Live View manages the other half of your playground book's user interface as well as the interactions between that interface and the data. It doesn't take much to create the live view.

Your playground template contains a **LiveView.swift** file. You could write all of your live view code inside the LiveView.swift file, but then you'll need to copy all of your code to another playground page live view if you wanted to make multiple pages that share the same live view behavior.

So with these few lines of code, I can share the same View Controller in the Sources Folder.

Here's what it looks like under the hood:

```
import PlaygroundSupport

let viewController: ViewController = ViewController(1)
PlaygroundPage.current.liveView = viewController
```

First, we import the PlaygroundSupport framework:

`import PlaygroundSupport`

Then, I mirror a view controller that's in the Sources folder. In the Live View we copy a View controller that is in the Sources folder to a locale constant variable:

```
let viewController: ViewController = ViewController(1)
```

Then when the playground page looks for the current page live view, we give the page an instance of View Controller.swift that's located in the Sources folder:

```
PlaygroundPage.current.liveView = viewController
```

That's it!

This Live View is "Always-On" it runs in a different process as the code editor on the left that runs when code in the editor is compiled.

Playground Page Manifest

In the playground page, **manifest.plist** contains the following key-value pairs specifying the attributes for a page.

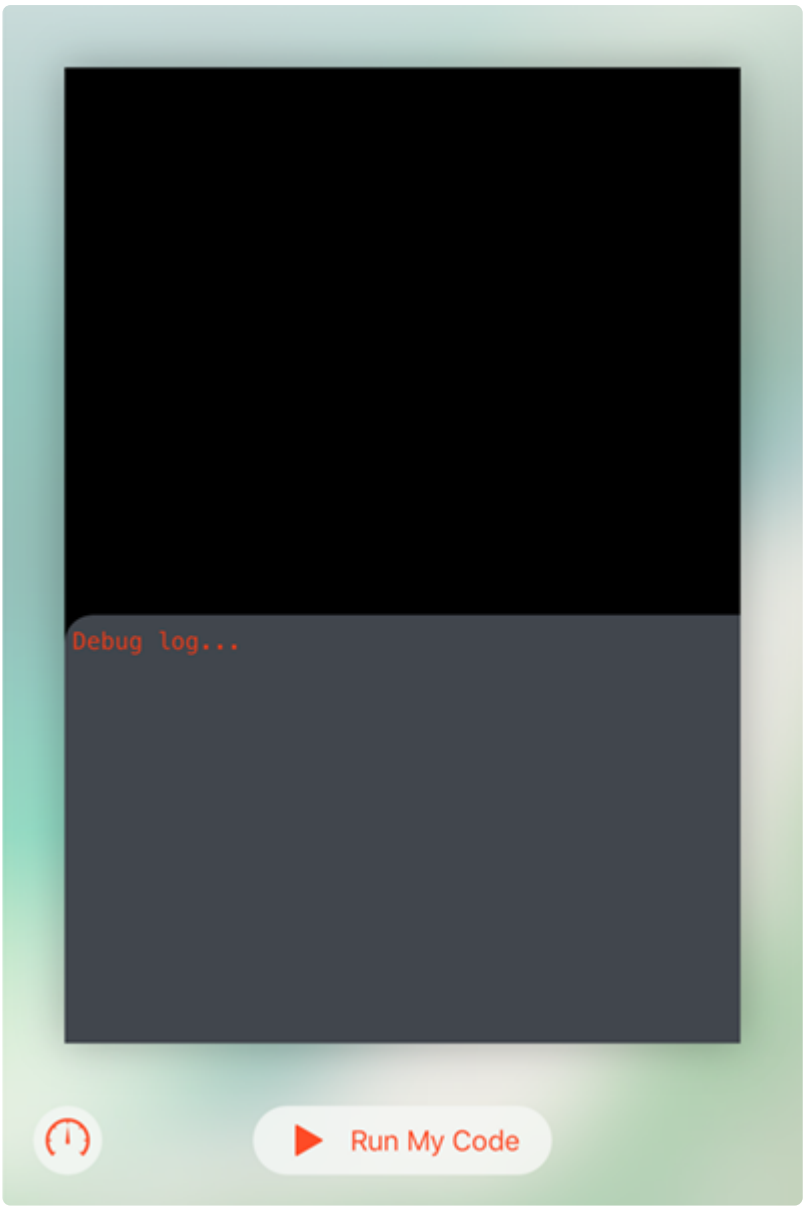
Key	Type	Value
▼ Root	Dictionary	(4 items)
Name	String	Page 1
LiveViewMode	String	VisibleByDefault
LiveViewEdgeToEdge	Boolean	YES
PlaygroundLoggingMode	String	Off

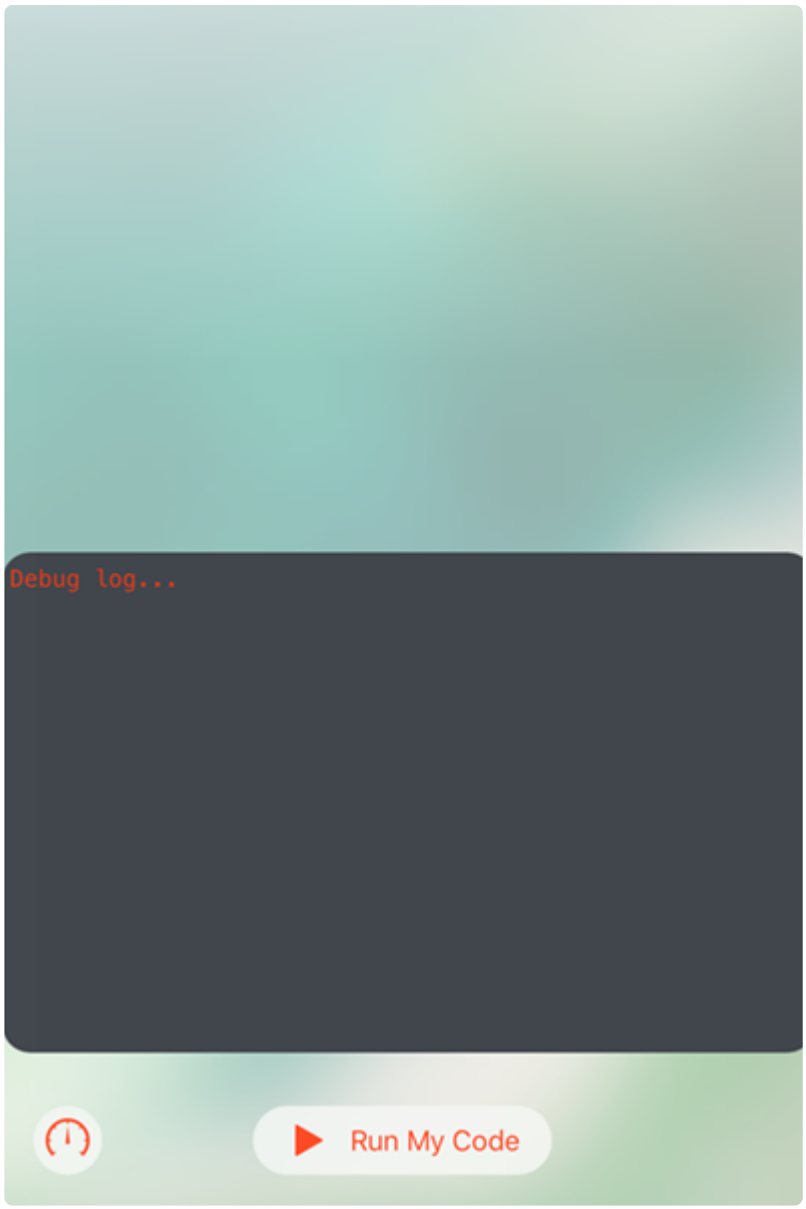
Name Key

The display name of the playground page displays at the top of the user interface ...

LiveViewEdgetoEdge Key

The LiveViewEdgetoEdge controls the size of the live view with a boolean value. Setting the key to NO reduces the size of the live view, while setting the key to YES expands the view.


Debug log...
Run My Code



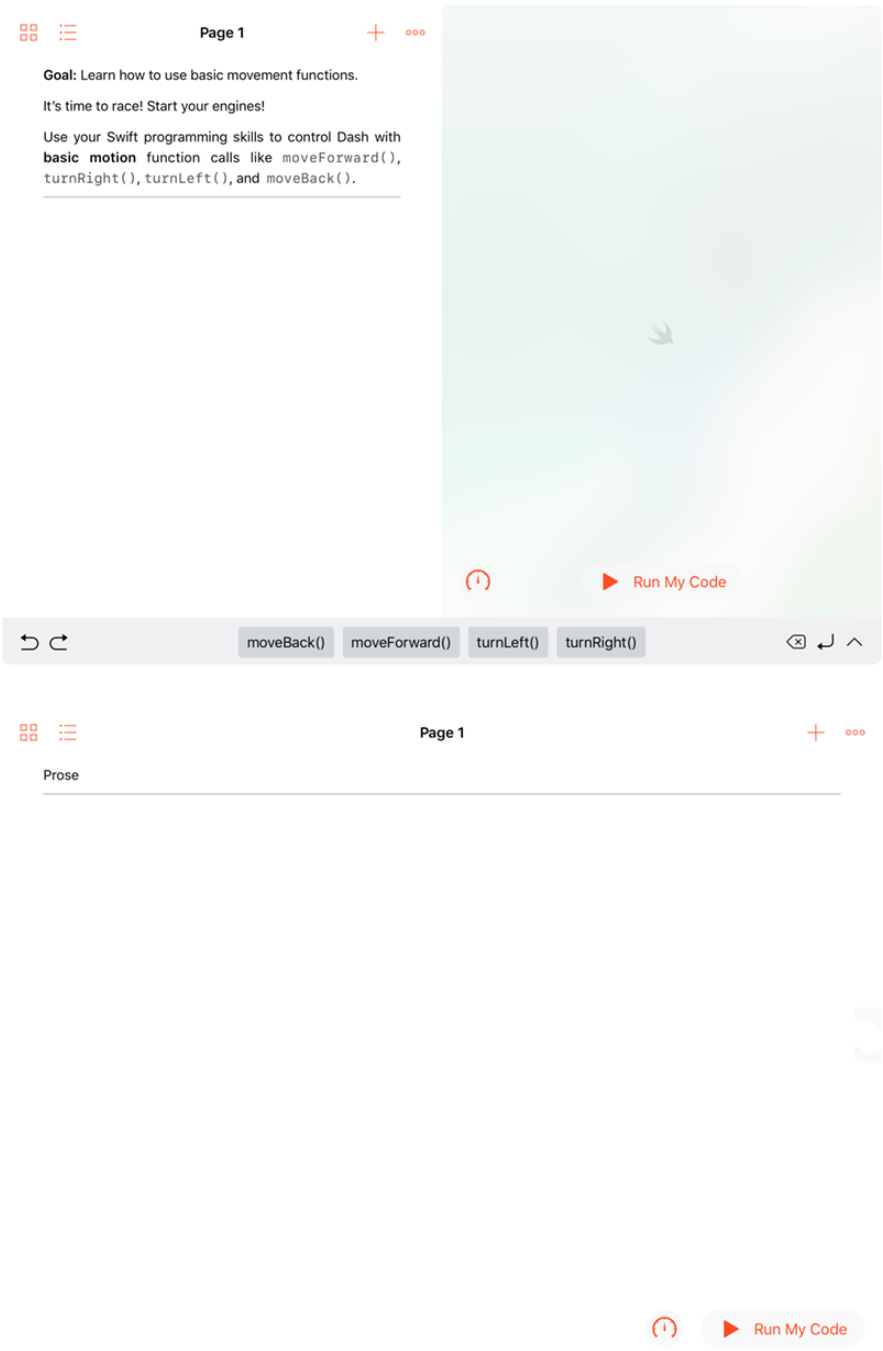
LiveViewMode Key

The LiveViewMode is used to control the display of the live view area while the live view isn't running.

The only values that can be used are:

• **VisibleByDefault -** Shows the live view when the playground opens.

• **HiddenByDefault -** Hides the live view until the playground is run.



Poster Reference Key

The base name of an image file that is shown centered and unscaled in the live view before the live view runs. Your image can be located in any **Resource** folder of your playground book.

PlaygroundBluetooth Connection View

Playground Bluetooth Connection View provides an interface for displaying connection status of Bluetooth devices. Before we tackle the PlaygroundBluetooth API, we need to setup this Connection View in the view controller so that users can choose discovered peripherals to connect to.

PlaygroundBluetoothConnectionViewDelegate

The **PlaygroundBluetoothConnectionView** protocol defines the methods that a delegate of **PlaygroundBluetoothCentralManager** objects must adopt. The optional

methods of the protocol allow the delegate to monitor the discovery, connectivity, and retrieval of peripheral devices.

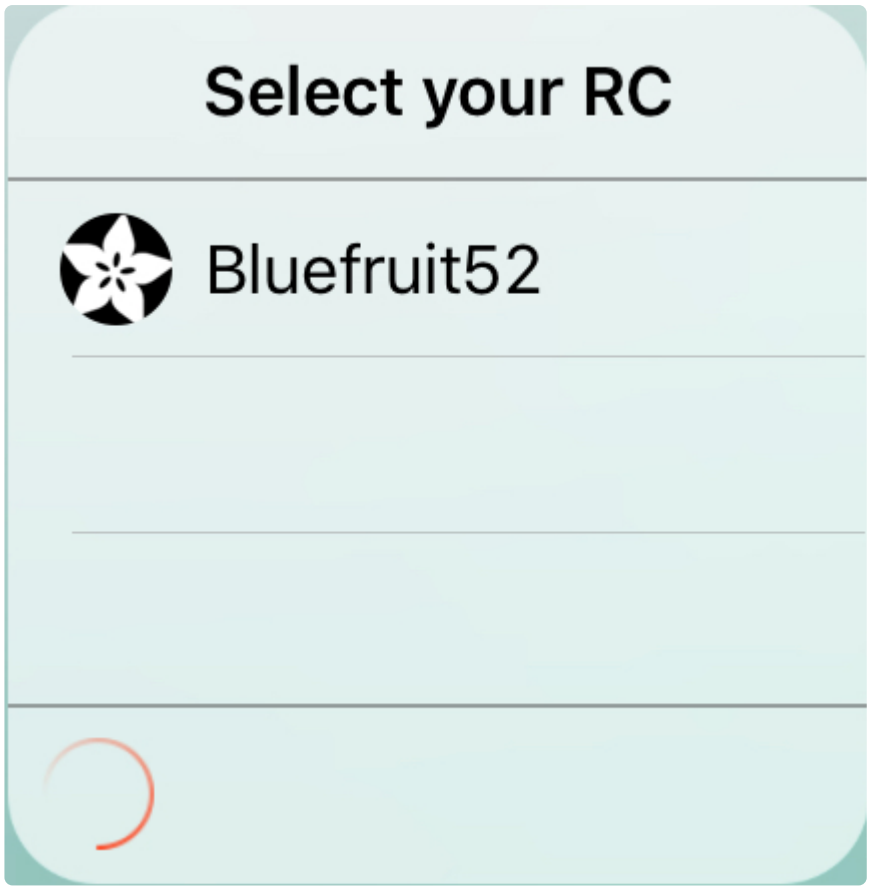
Apple does not go into much detail about PlaygroundBluetooth Connection View, but these are the methods used to display a table view that shows you the connectable devices discovered during a peripheral scan:

```
connectionView(_: titleFor:)

connectionView(_:itemForPeripheral:withAdvertisementData:rssi:)

connectionView(_: shouldDisplayDiscovered:withAdvertisementData:rssi)

connectionView(_: firmwareUpdateInstructionFor:)
```

Connection View: TitleFor method

In the Swift Playgrounds for Bluefruit playground books live view, you'll notice a button with the label "**Connect RC**". This was added using the `connectionView(_: titleFor: )` method. This method provides a localized title for the given state of the connection view.

For example, if the connection view state is "**noConnection**", the UI label will display "**Connect RC**" which will give the user the option to search for peripherals in the area. If the connection view state is "**selectingPeripherals**" the UI will expand into a table view that will populate the rows with peripheral devices found with the **UUID** we are looking for. On top of the table view, we see the "**Select your RC**" label that was localized in the **titleFor** method:



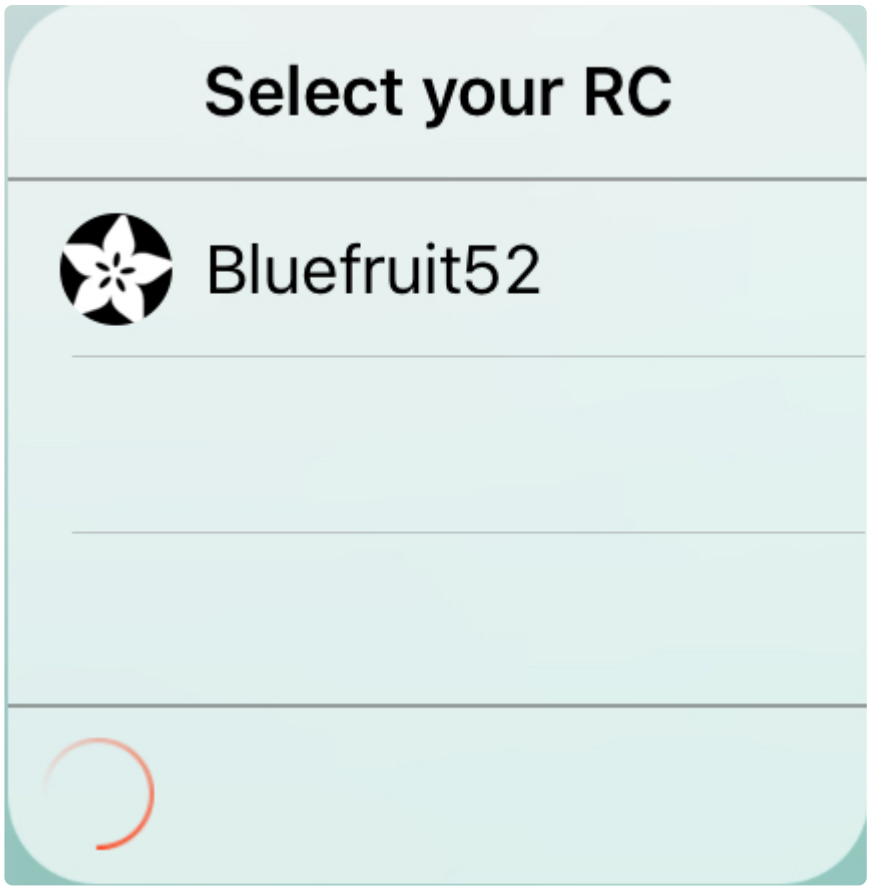
Connection View: itemForPeripheral method

This method displays UI elements when a peripheral appears in the table view. Here, you'll need to add a peripheral name placeholder for any unidentified peripheral.

Then, you'll need to give each peripheral device an icon image for the peripheral to be displayed in the table view. I created an image of the Adafruit logo in photoshop with the dimensions 83x83. The image was placed in the **Private Resources** folder then referenced in the **itemForPeripheral** method:

```
let icon = UIImage(imageLiteralResourceName:"Images/adafruit_logo_small copy.png")
let issueIcon = icon
```

Now when a peripheral is found, it will have a name displayed and image attached to it:



Connection View: shouldDisplayDiscovered method

This method filters out peripheral items. There's not much information available for this method, but it is optional.

Connection View: firmwareUpdateInstructionsFor method

This method provides firmware update instructions. This is also optional.

Adding Bluetooth to your Playground Book

Swift playgrounds provides full access to the **PlaygroundBluetooth** framework, which allows you to integrate bluetooth devices into your Playground Book. There are two main parts of the **PlaygroundBluetooth** framework.

• First is the **PlaygroundBluetoothConnectionView**, which allows you to customize a user interface that allows users to interact with devices.

• Second would be the **PlaygroundBluetoothCentralManager**, which is responsible for interacting with and managing devices.

The **PlaygroundBluetoothCentralManagerDelegate** has methods of interacting with devices, such as connecting to and disconnecting from devices.

PlaygroundBluetoothCentralManagerDelegate

These are the methods you'll need for interface connectivity:

```
centralManagerStateDidChange(_: )

centralManager(_: didDiscover:withAdvertisementData: )

centralManager(_: willConnectTo: )

centralManager(_: didConnectTo: )

centralManager(_: didFailToConnect: error: )

centralManager(_: didDisconnectFrom: error: )
```

PlaygroundBluetooth is similar to CoreBluetooth's CBCentralManager, so if you've implemented that before in a Bluetooth LE, then you reuse code here. Or you can take a look at the Create a Bluetooth LE app for iOS (https://adafru.it/Ch3) learn guide.