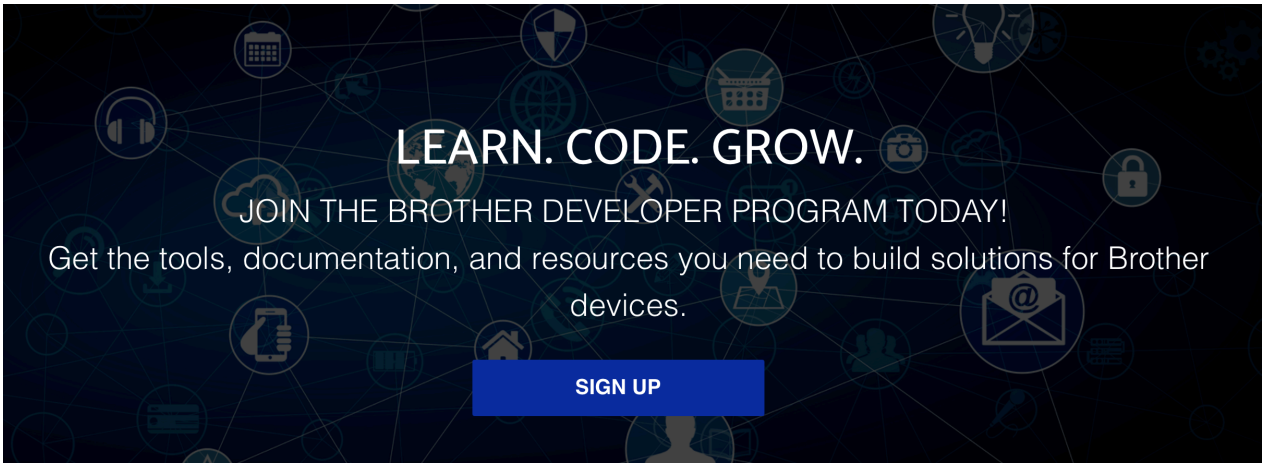
Brother Printer SDK for WiFi

Project Setup and Tutorial

DOWNLOAD THE BROTHER PRINTER SDK

Create a developer account at:

https://developerprogram.brother-usa.com

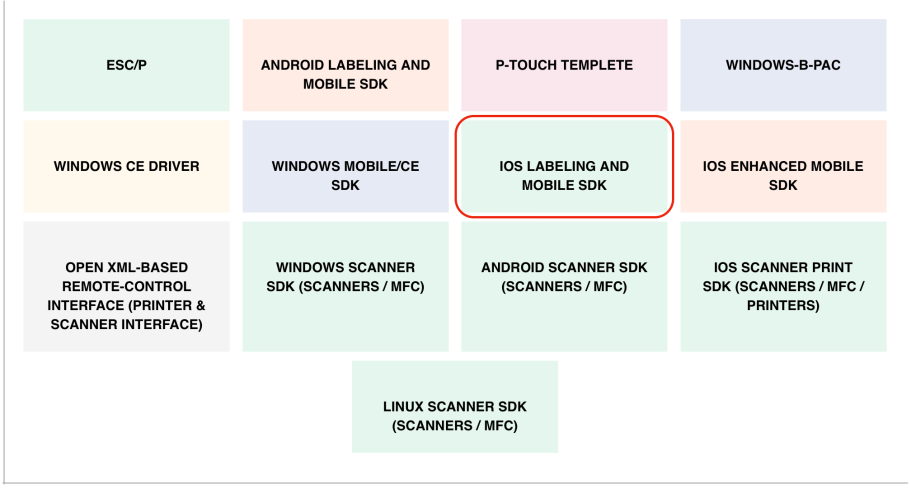


Navigate to:

https://developerprogram.brother-usa.com/sdk-download

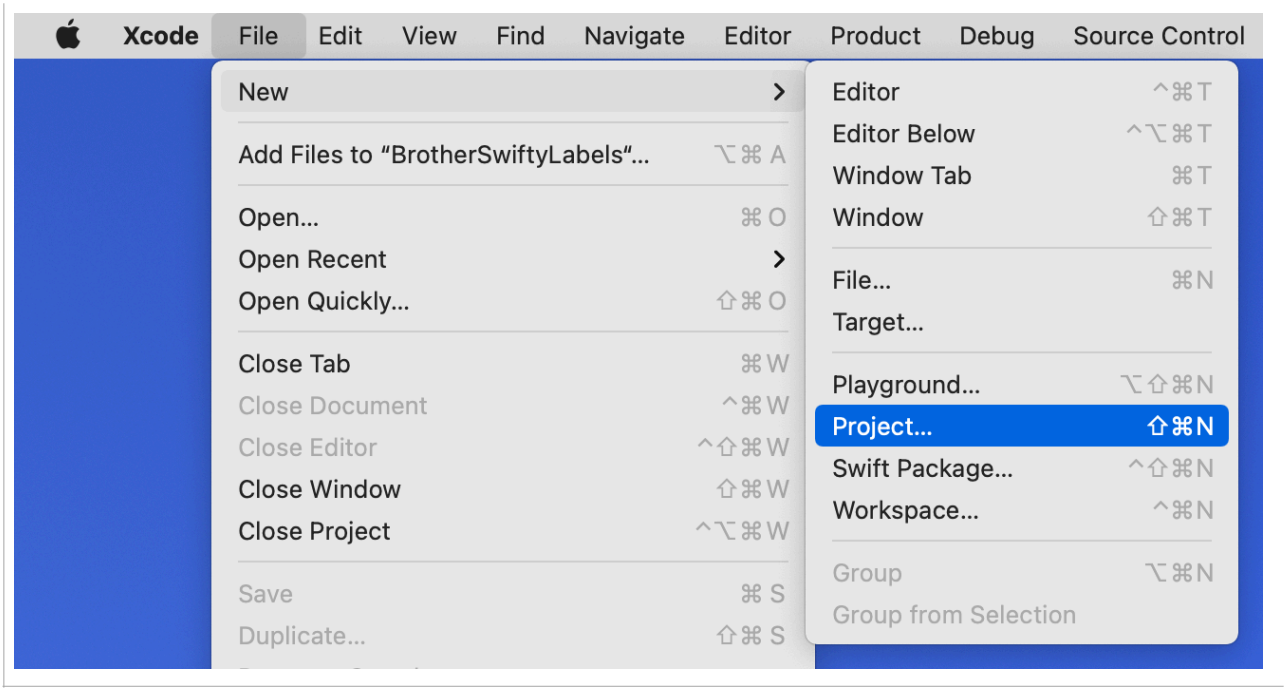
Click on the **IOS LABELING AND MOBILE SDK** link and download version 4.3.1

Unpack the download and store the files in your development area

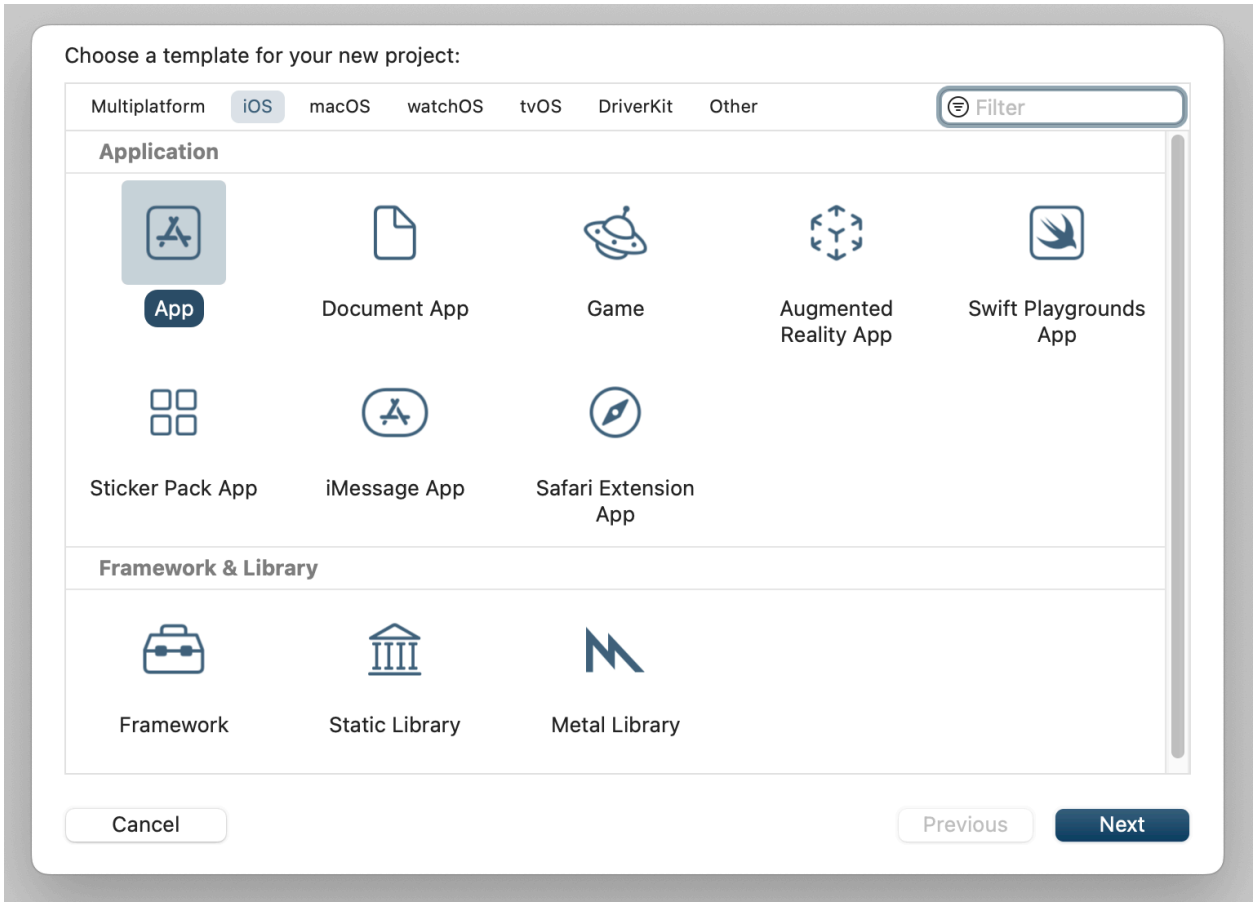


BUILD AN XCODE PROJECT

Start Xcode and create a new project from the **File** menu



Select the **App** template from the **iOS** tab and press **Next**



Give your project a name

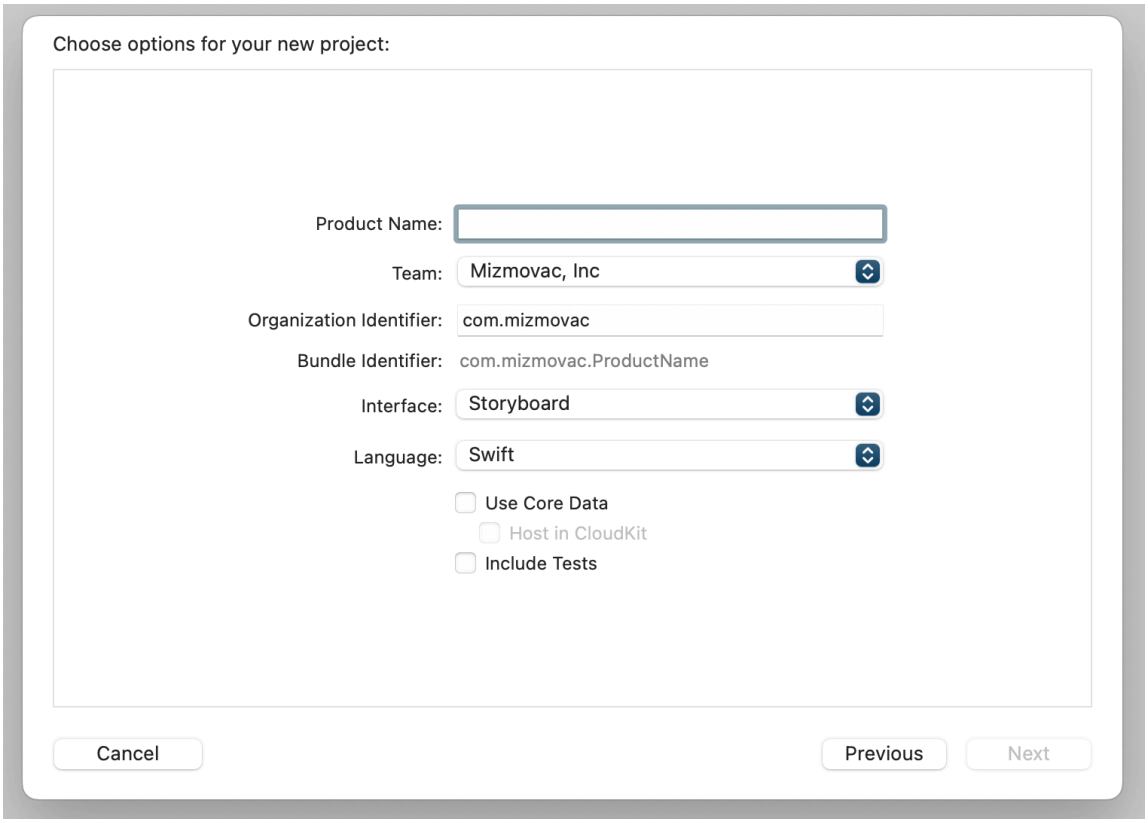
Select your Apple Developer ID from the **Team** dropdown

Select **Storyboard** from the **Interface** dropdown

Set **Swift** as your **Language** option

Make sure that **Use Core Data** and **Include Tests** are not checked

Press **Next**



Select or create a folder for your project files

Uncheck the **Create Git repository…** option (when it is time to create your Hackathon projects, you should always use Git to save and backup your work)

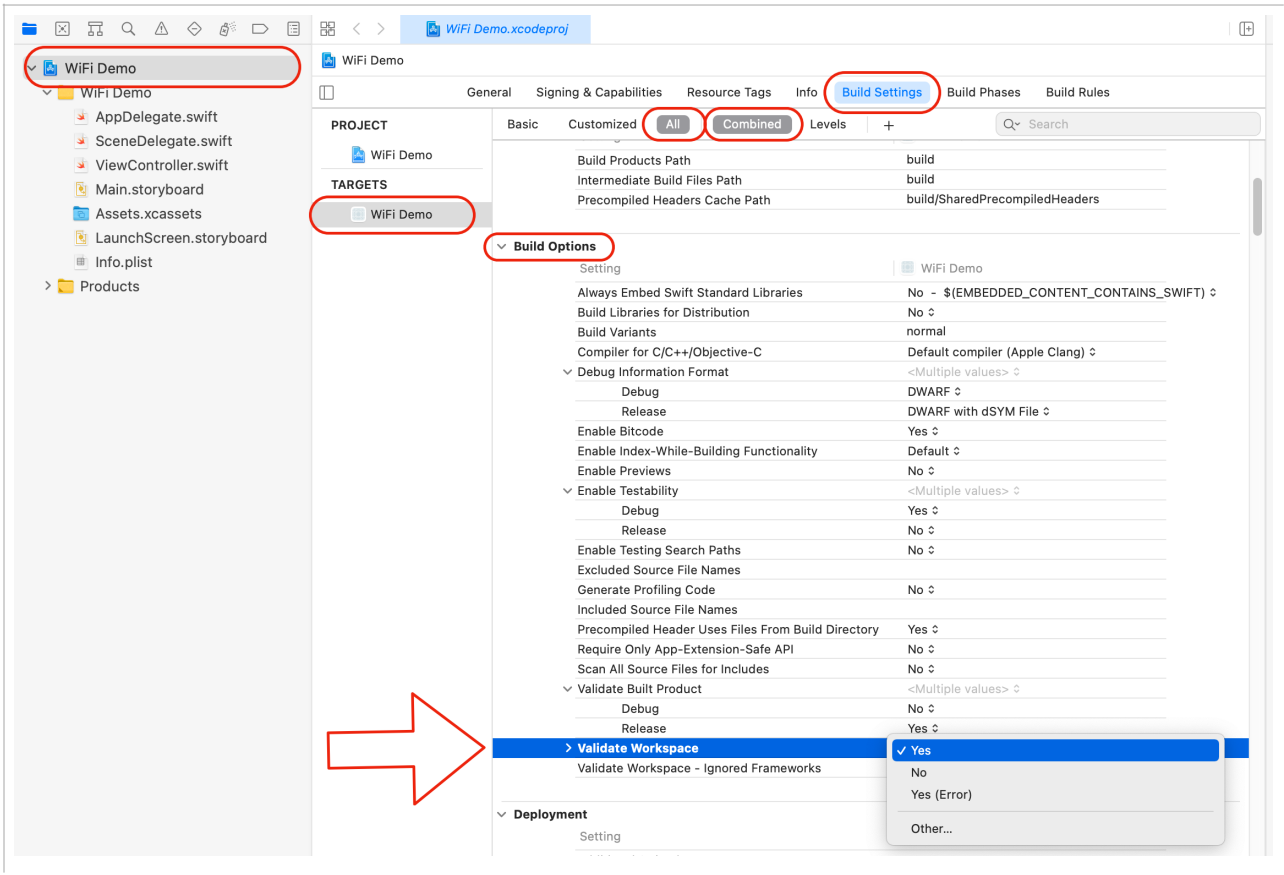
Press **Create** to save your project files

Immediately after your project files are created, you will be taken to the project option tabs

Refer to the circled screen elements, as illustrated below, and select them so that you navigate to the **Build Settings** tab

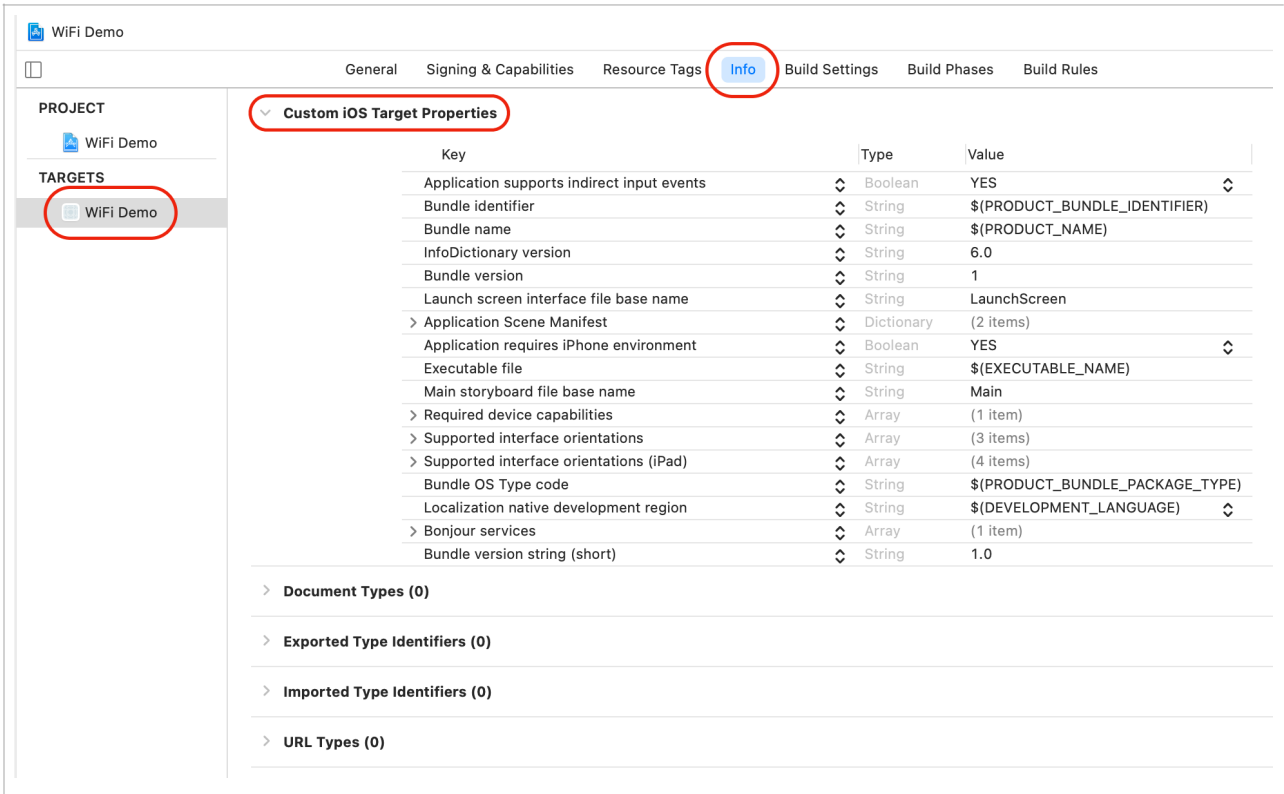
Inside of the **Build Options** category, look for the **Validate Workspace** option and select **Yes** (you will have to scroll down a bit to find **Build Options**)

Click away from the row and double check that the option is set to **Yes**



Your app will need to use Bonjour services to find Brother printers in your network

Click on the **Info** tab and expand the **Custom iOS Target Properties** category, if necessary



You will add additional key/value properties to this list

With your mouse pointer, hover over the first row in the list to expose ⊕ and ⊖ buttons, as shown in the first step

Click the ⊕ button to display a list of property keys. Search for **Bonjour services** and select it by pressing **Return**.

Expand the **Bonjour services** array list view by clicking the **>** button at the left, as seen in step 4

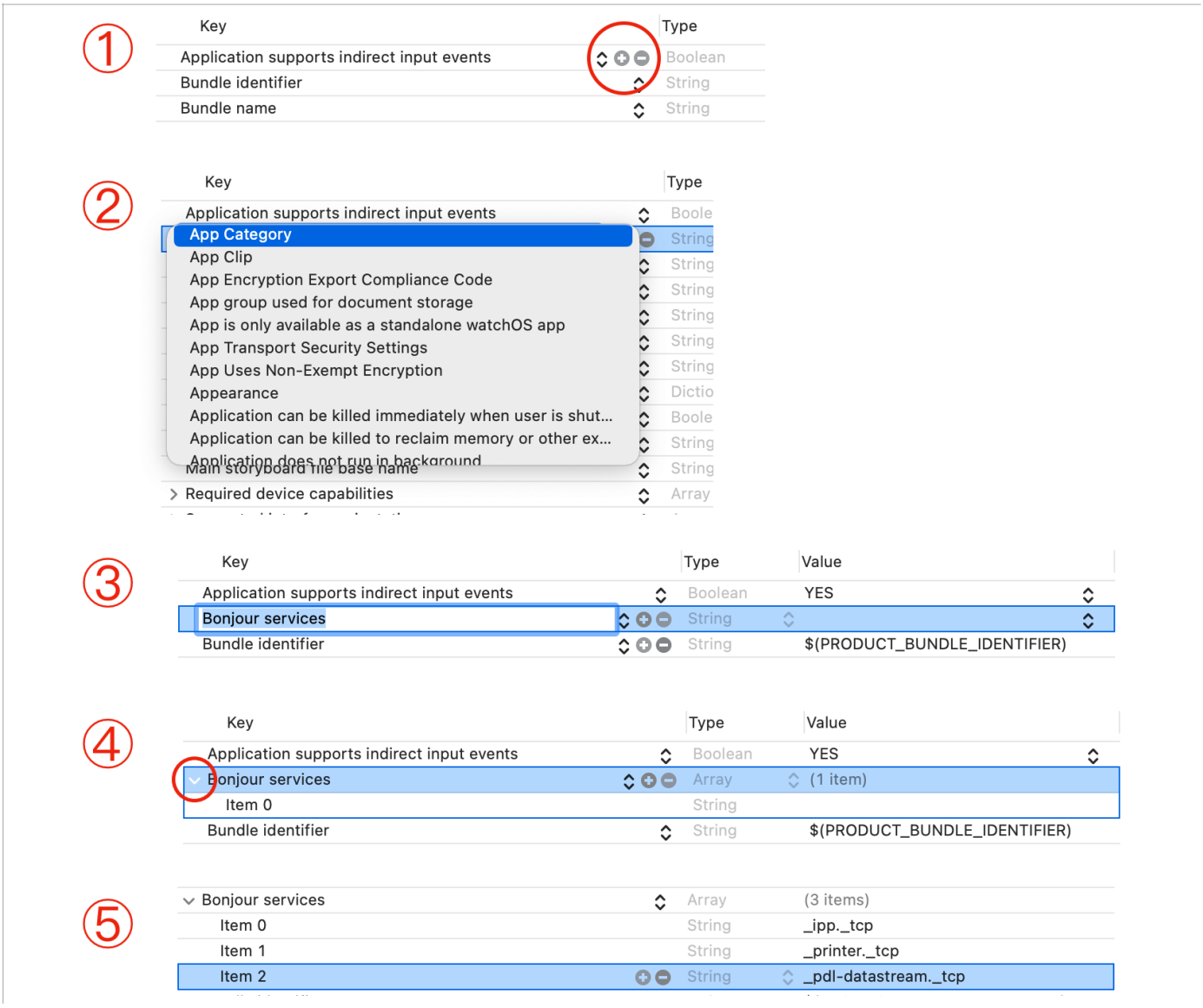
Click on the **Value** field in the far right of the row and type **_ipp._tcp**

Add a new item to the list by clicking on the ⊕ button in the **Item 0** row

Type **_printer._tcp** into the **Value** field for **Item 1**

Add another item row and type **_pdl-datastream._tcp** into the **Value** field for **Item 2**

These are the names of the services that Bonjour will use when searching for Brother printers



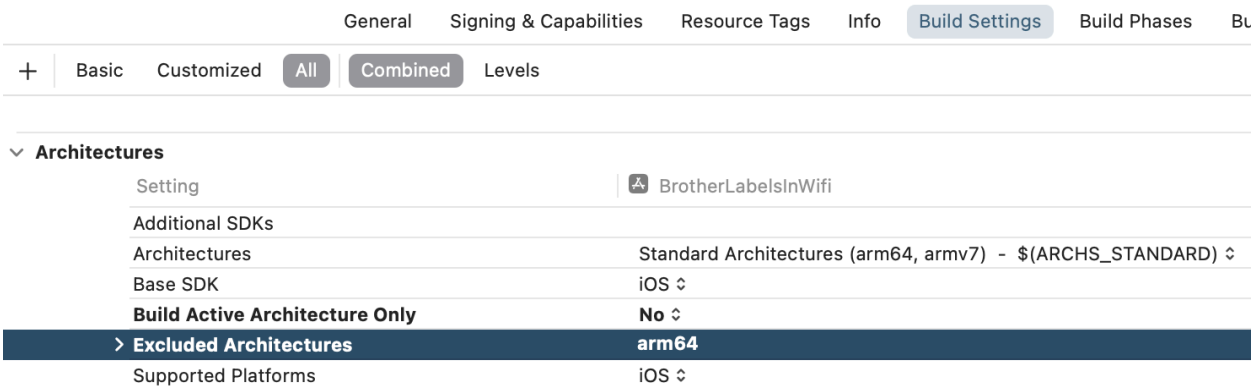
If you are using an M1 Mac for development, you will need to force Xcode to exclude arm64 architecture in the simulator runtime code. This step is not necessary if you are using an Intel Mac, but making the changes detailed below will not affect the simulator. When you install an updated version of the SDK, verify if you no longer need to exclude ARM architecture.

Click on the **Build Settings** tab and expose the **Architectures** category, if necessary

Make sure that **Build Active Architecture Only** is set to **No**

In the field next to **Excluded Architectures**, enter **arm64**

Verify that the **Supported Platforms** option is set to **iOS**



Your user will have to grant permission to your app for local network access. iOS allows you to present a custom message explaining the need for access. In this section you will directly modify the **info.plist** file and supply that message.

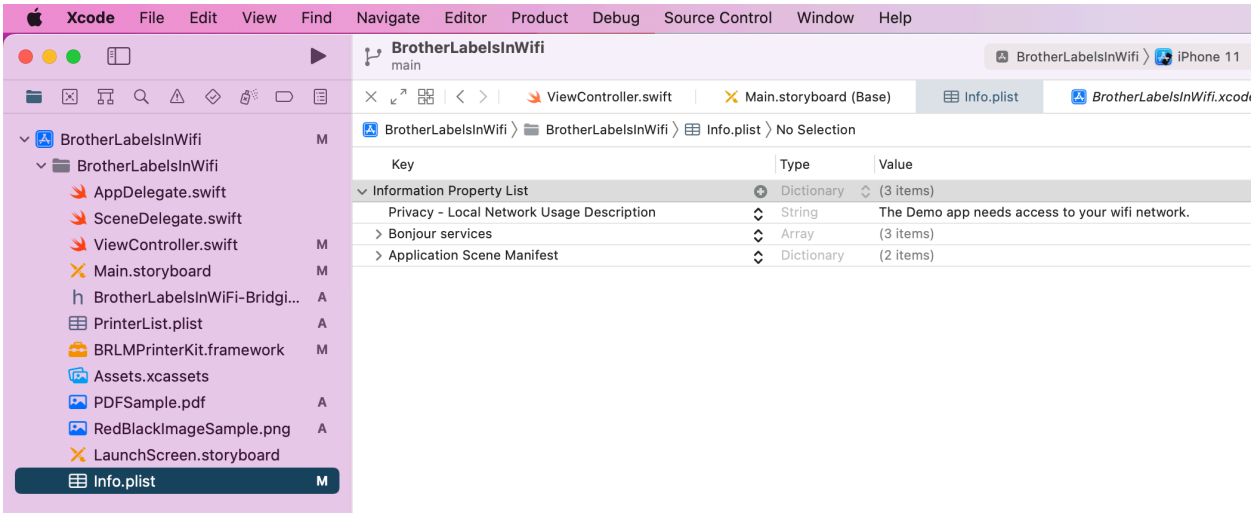
The file contains XML used to define properties used by your app. In the steps above we edited similar properties through the project property browser. You can preform the step below inside of the **Info** tab in the **Custom iOS Target Properties** section, but at the time this document was written the property did not seem to get persisted into the **info.plist** file, as required. You may wish to try this yourself or just directly edit the **info.plist** file.

In the **Project Navigator** select the **info.plist** file

Click on the ⊕ in the **Information Property List** column and select **Privacy - Local Network Usage Description** and press Return

Click in the **Value** field and enter a brief statement that asks for permission to use the network. This will appear in a dialog box when the app makes an initial attempt to use the attached WiFi network.

Press return to commit the message



INSTALL THE BROTHER SDK AND RESOURCE FILES

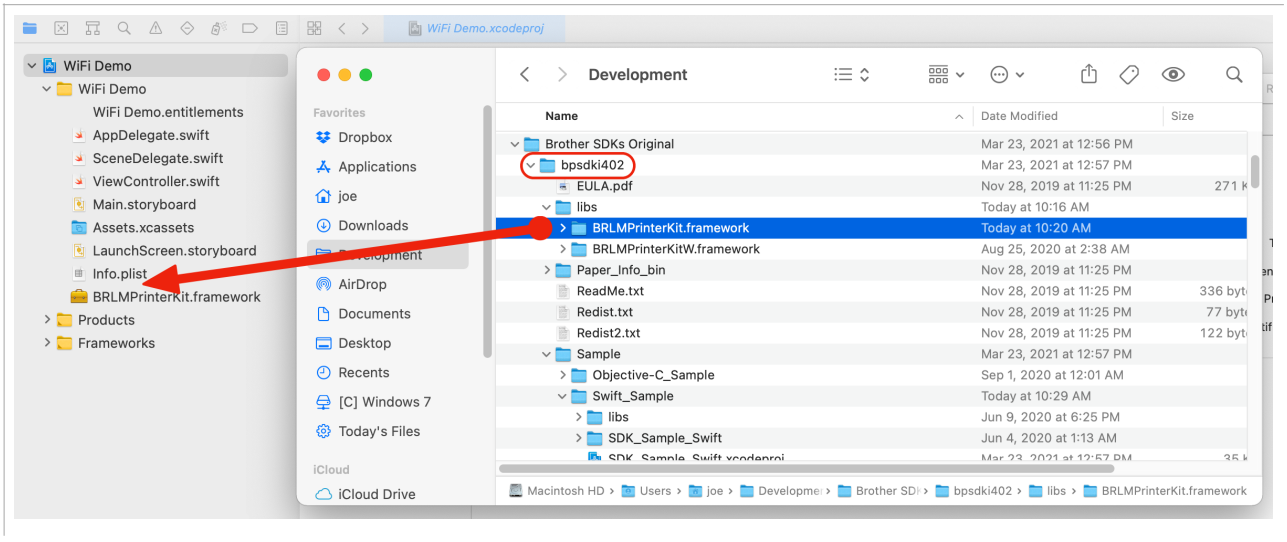
You will next add a copy of the Brother Printer SDK into your project.

Open a **Finder** window and navigate to your copy of the Brother Printer SDK (the folder will probably be labeled “bpsdki431”)

Drill down to the **libs/Net** folder and find “BRLMPrinterKit.framework”

Drag the folder out of the **Finder** window into the project navigation pane in **Xcode**

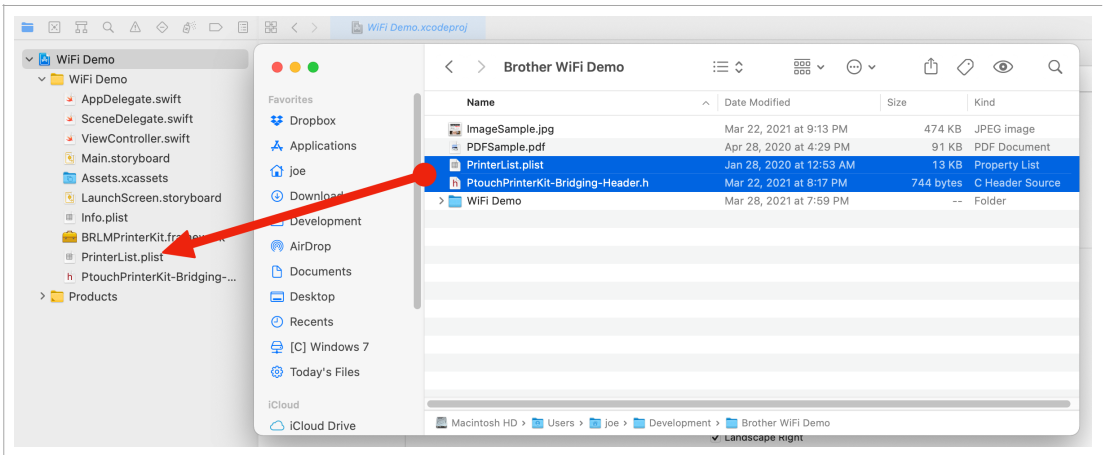
Verify that you see the framework in the navigator



In the **Finder** window, open the **Project Resources** folder

Find the files **PtouchPrinterKit-Bridging-Header.h. PrinterList.plist, PDFSample.pdf** and **RedBlackImageSample.png** that were included with this tutorial document

Drag them out of the **Finder** window into the project navigation pane in **Xcode**



In the project **Build Settings** tab, scroll down and find the **Swift Complier - General** section, as shown in step 1

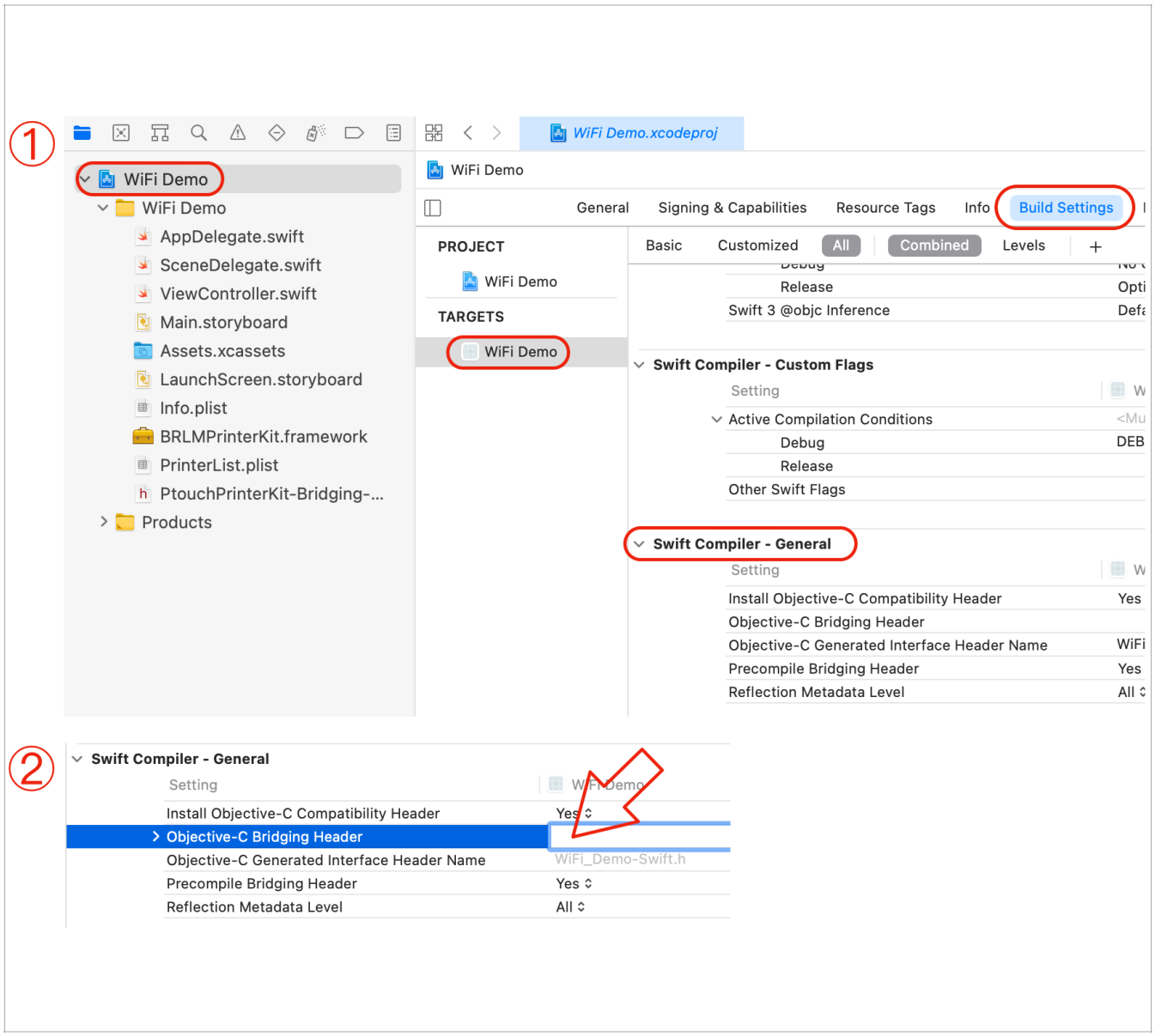
Click in the empty field next to **Objective-C Bridging Header** as shown in step 2 (you may have to click more than once to select the field for editing)

Type (or copy) this file path into the selected field:

./WiFi Demo/PtouchPrinterKit-Bridging-Header.h

Be careful to note leading "**.**" in the file path. This is necessary to create a relative file path to the bridging header.

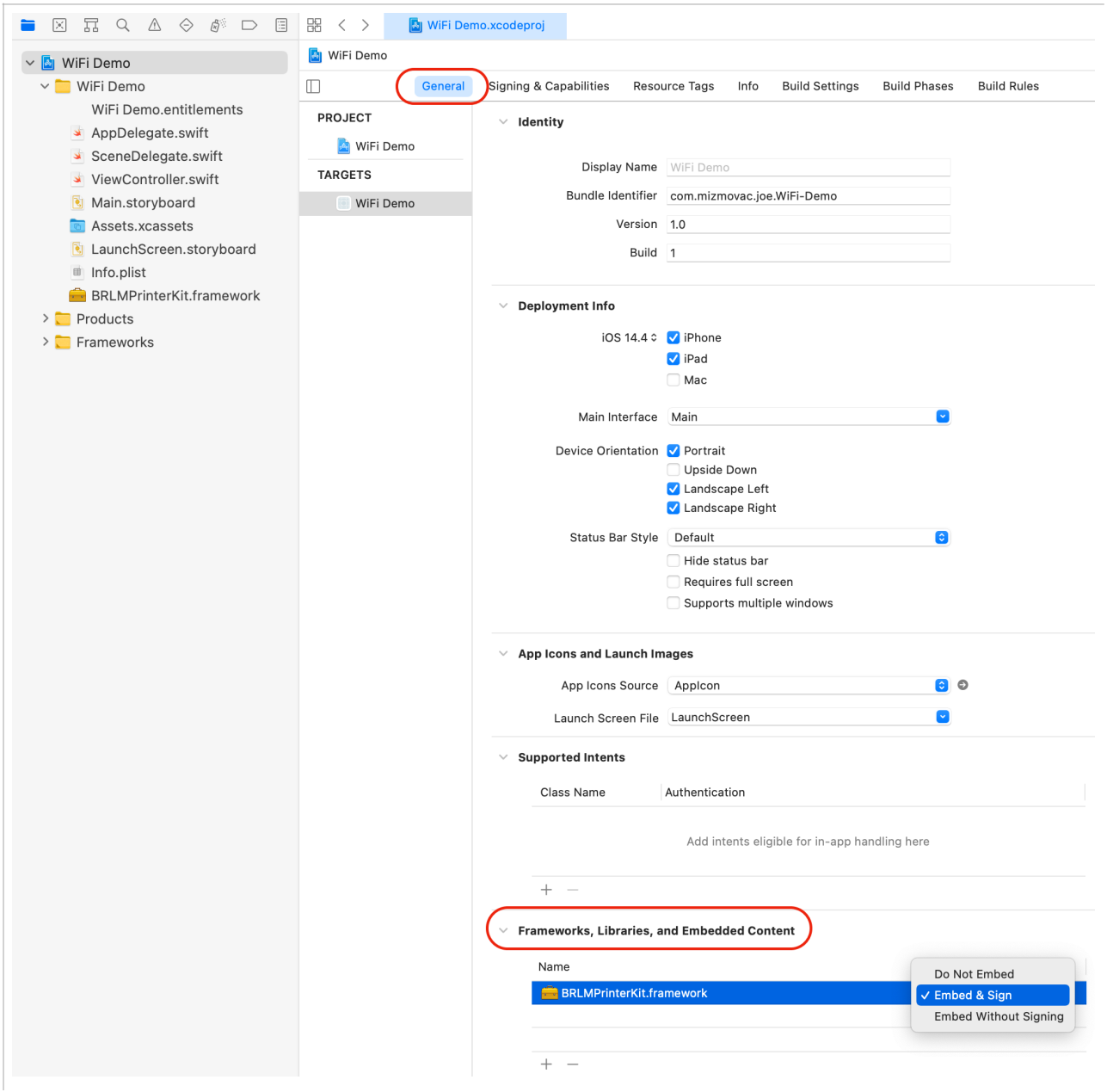
If you named your project other than "WiFi Demo", be sure to substitute your project name in the path



Click on the **General** tab and locate the **Frameworks, Libraries, and Embedded Content** section (you may have to scroll down in the window)

Find the **BRLMPrinterKit** framework and select the **Embed & Sign** option

Click away from the dropdown and confirm that **Embed & Sign** is selected

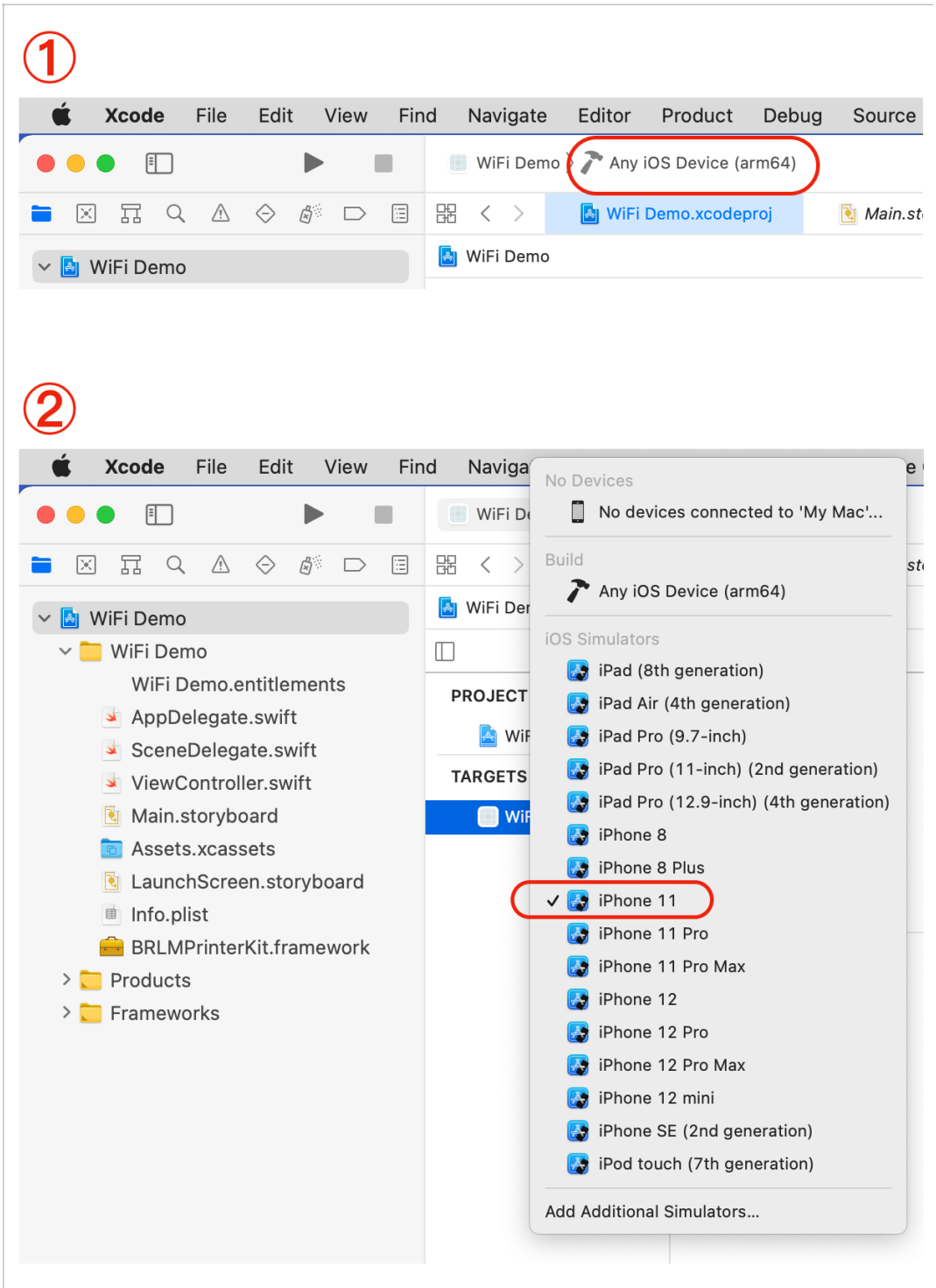


VERIFY THAT THE PROJECT WILL COMPILE

Make sure that your project is correctly built by compiling and running the simulator.

Locate the **Active Scheme** dropdown control, as shown in step 1

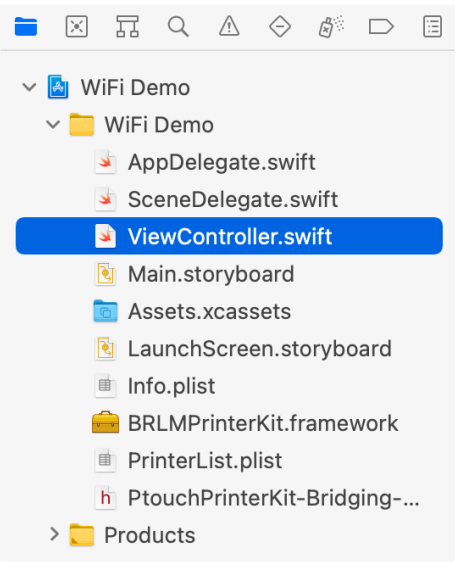
Click and select **iPhone 11**



In the project navigator, click on **ViewController.swift** to display the file in the editor

Just below the **class** name, type the indicated line of code into the file (be sure to type the variable name exactly—it will be used by other methods later)

①



②

```
8   import UIKit
9
10  class ViewController: UIViewController {
11
12      var selectedDeviceInfo: BRPtouchDeviceInfo?
13
14      override func viewDidLoad() {
15          super.viewDidLoad()
16          // Do any additional setup after loading the view.
17      }
18
19
20  }
21
```

Click the **Build and Run** button as shown in step 1

If the project builds successfully, you should see a simulation of an iPhone 11, but without any user interface controls

Click the **Stop** button as shown in step 3

Now that you have confirmation that your project is correctly configured, you can begin to build a functional app.

BUILD A USER INTERFACE

Begin by selecting the **Main Storyboard** as shown in step 1

Xcode will display a view of an iPhone where you can drop user interface components

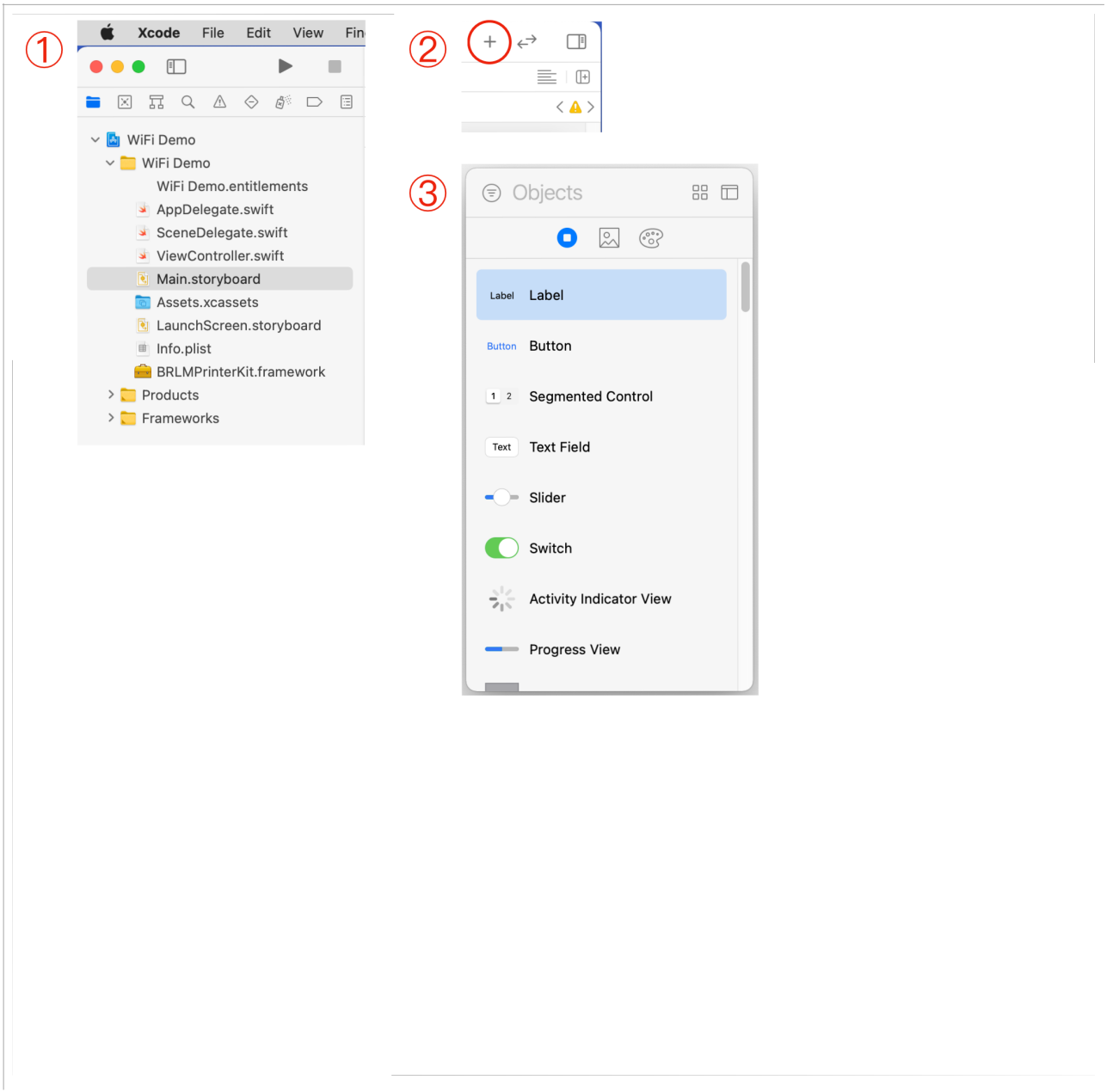
Click the **Library** button, located in the far upper right of the window, as shown in step 2

A selection of user interface controls will be displayed (if you do not see a list similar to step 3, try clicking the circle button just below the **Objects** field

You will drag and drop user controls from this list into the iPhone on the screen

If the list disappears after placing a user control, just press the **Library** button again

Layout user controls similar to the illustration on the right



Notice the order and type of user controls:

Label

Table View

Label

Label

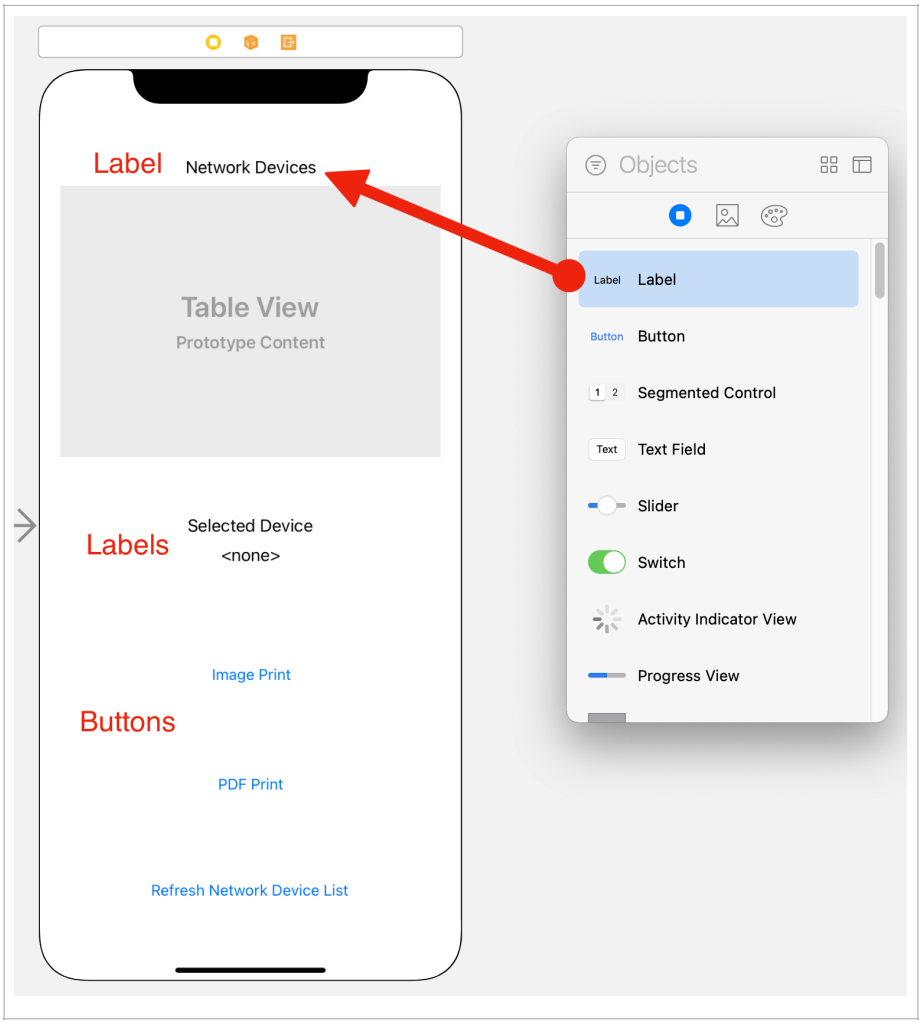
Button

Button

Button

Customize your labels and buttons by typing into the field that surrounds the control (if you don't see the field, try clicking on the control)

Also, consider stretching the controls so that they occupy most of the screen width



You may have noticed that stretched labels are left justified

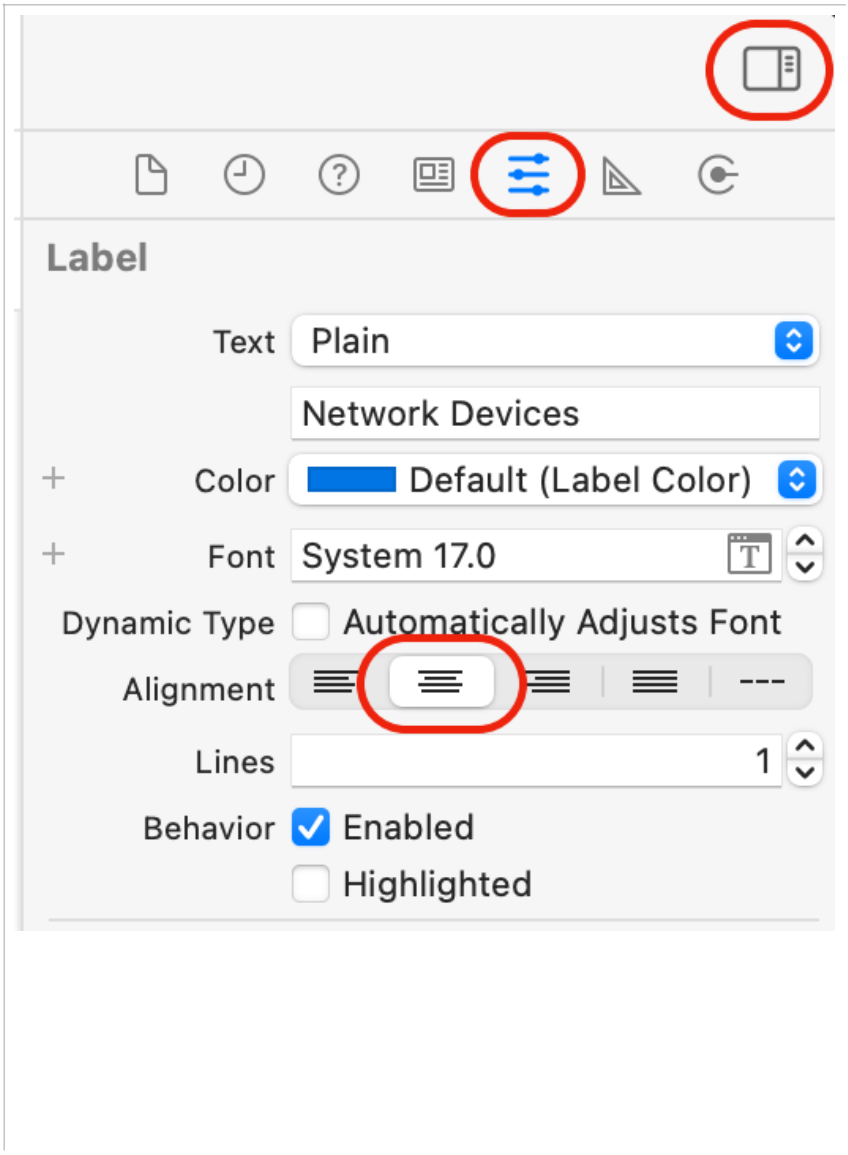
If the **Attributes Inspector** is hidden, click on a label in the layout view

If you still do not see the inspector, locate the **Hide or Show Inspectors** button at the upper left-most corner of the Xcode window and click it to reveal the inspector

Select the third icon from the left (it looks like three stacked slider controls) to display the **Attributes Inspector** view

Click on the **Center Alignment** tab as shown in the example and verify if your selected label has centered itself

At this point in building your app, don't spend too much time perfecting the user interface. You can always come back to the **Storyboard** later and modify the view.



LOCATING BROTHER DEVICES

Create properties to store devices discovered on your network. Refer to the **ViewController.swift** file to see the follow code in context.

Copy and paste the code on the right into the **ViewControl.swift** file, just below the the **class ViewController: UIViewController** statement

wiFiDevicesInfo is an array that will contain all of the Brother printers in your network

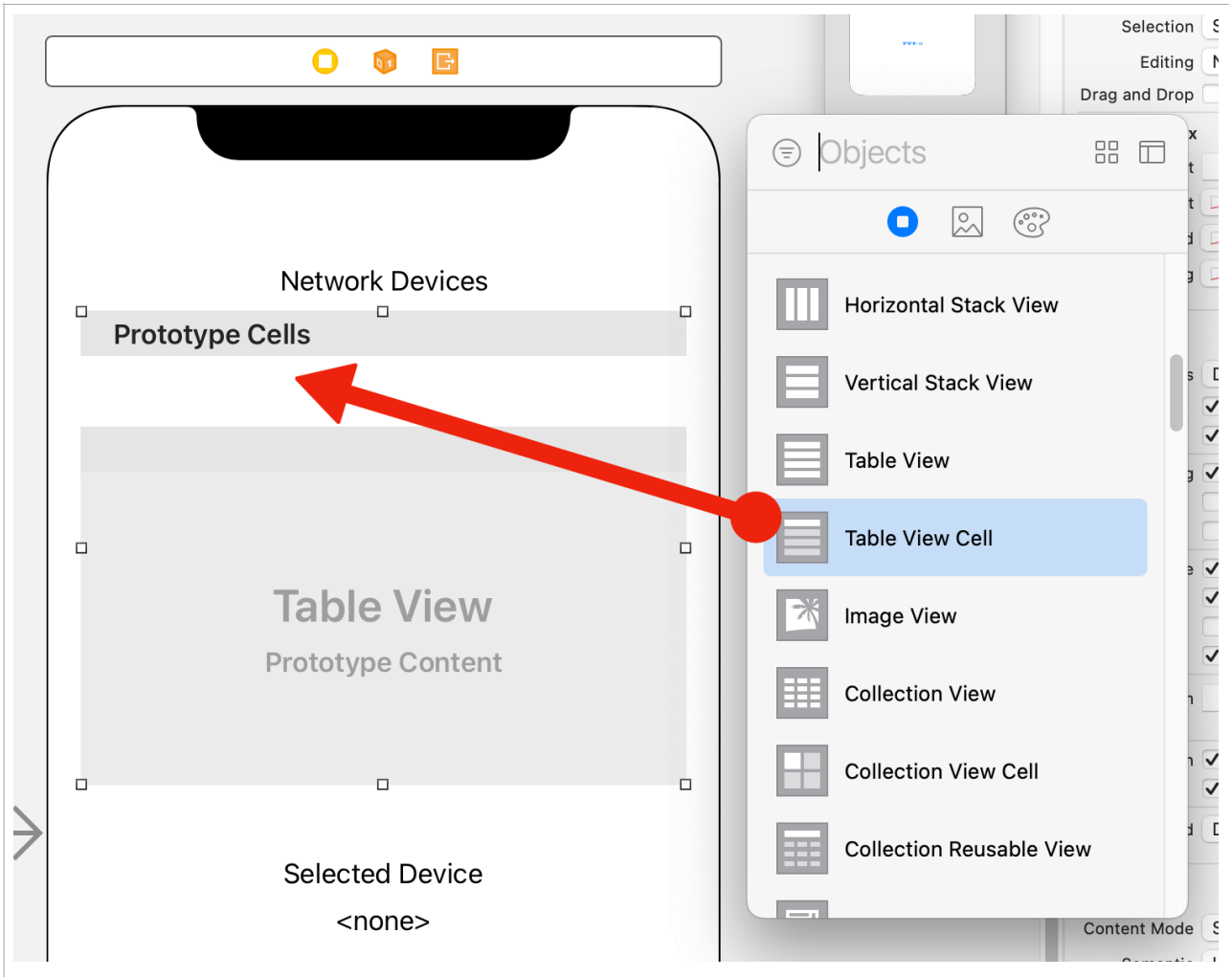
selectedDeviceInfo will be used later to specify which printer you will send printer commands to

networkManager is a class provided by Brother to manage printer communications

```
    var wiFiDevicesInfo: [BRPtouchDeviceInfo]?
    var selectedDeviceInfo: BRPtouchDeviceInfo?
    var networkManager: BRPtouchNetworkManager?
```

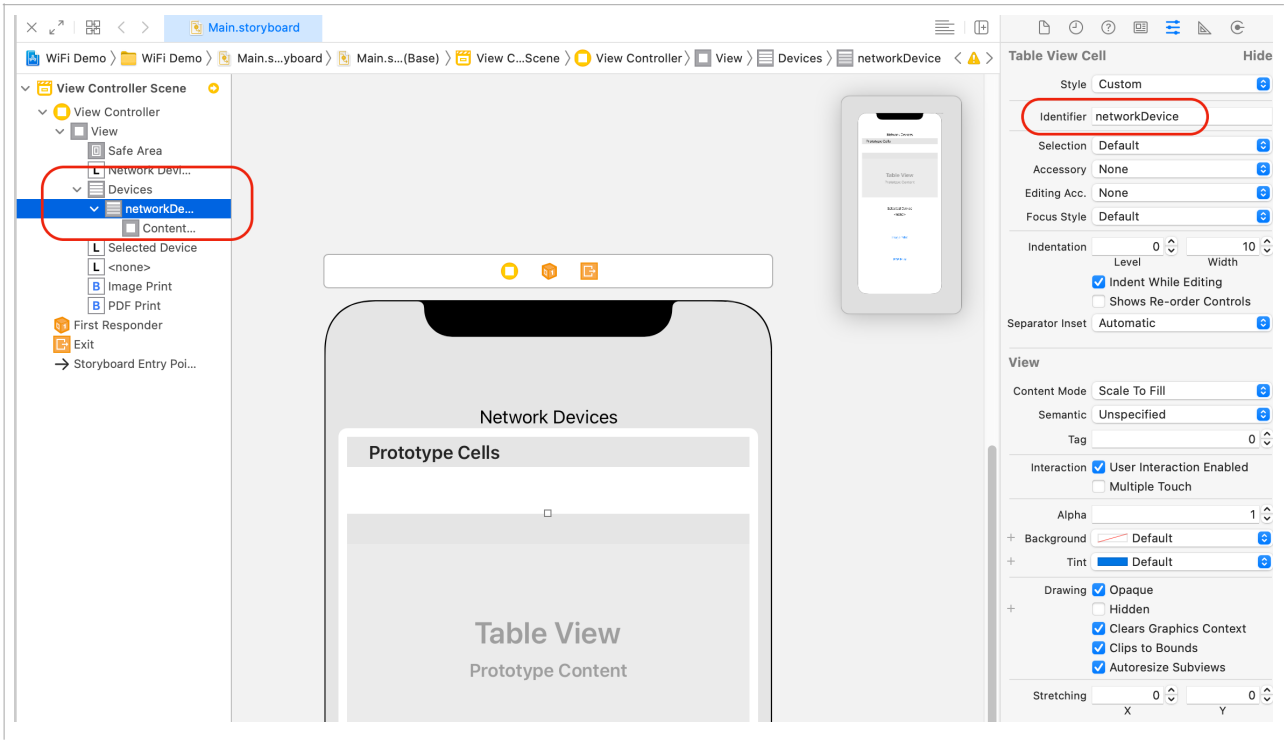
Add a prototype cell to your TableView by dragging a **Table View Cell** from the **Library** into the on-screen **TableView**

Prototype cells are containers for your data into the TableView



Name the cell **networkDevice** as shown on the right

Be careful to select the correct TableView element in the hierarchy and note the camelcase formatting of the name



Establish a link between your **ViewController** and the TableView inside your **Storyboard** by dragging an “outlet” to your code

First, select the **ViewController.swift** file from the project navigator to display it in the editor

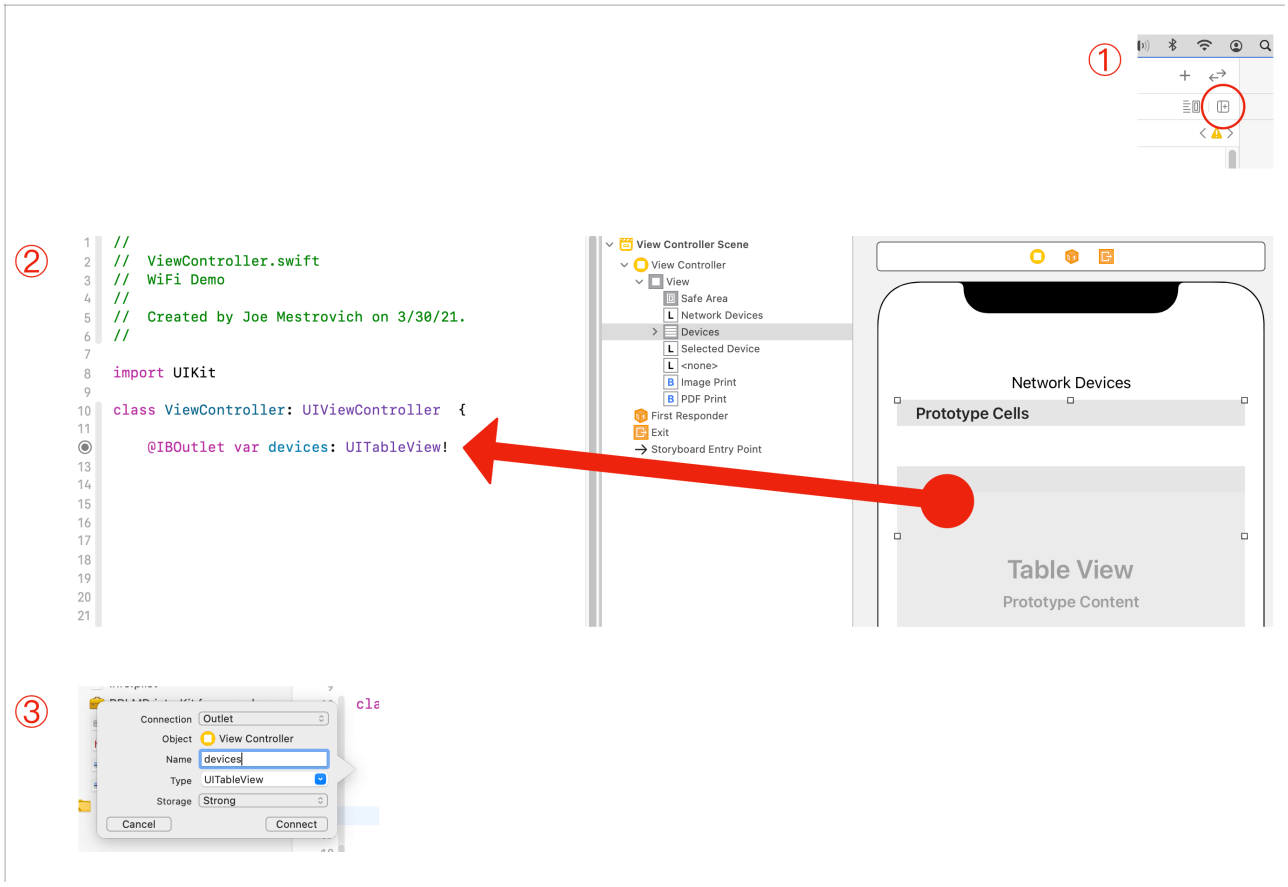
Click on the **Add Editor…** button located in the upper right of the editor window (see step 1)

Click the **Main.storyboard** member in the navigator to show a split-screen of your code and storyboard

Hold down the Control key and drag from inside of the TableView to just below the **Class ViewContoller** statement

When you complete the drag operation, a small dialog window will be displayed—type **devices** into the **Name** field and press **Connect**

Notice the filled in circle to the left of the **@IBOutlet** statement that indicates a successful link



At the very bottom of the **ViewController** file, copy and paste this extension (be sure to place this below the last closing bracket "**}**" in the file)

The **didFinishSearch** function will gather all the Brother printers in your network and store them into the **wiFiDevicesInfo** array

The **allBrotherPrinters** function will provide a list of compatible Brother printers for the **BRPtouchNetworkManager** to search for

```
extension ViewController: BRPtouchNetworkDelegate {

        func didFinishSearch(_ sender: Any!) {
                guard let foundDevices = networkManager?.getPrinterNetInfo() else { return }

                wiFiDevicesInfo = foundDevices as? [BRPtouchDeviceInfo] ?? []

                devices.reloadData()
        }

        func allBrotherPrinters() -> [String] {
                guard let printNamesURL = Bundle.main.url(forResource: "PrinterList", withExtension: "plist")
                        else { fatalError("PrinterList.plist missing in bundle") }

                let printersDict = NSDictionary.init(contentsOf: printNamesURL)!
                let printersArray = printersDict.allKeys as! [String]

                return printersArray
        }

}
```

Copy and paste these **UITableView** and **UITableViewDelegate** extensions into the **ViewController** file, again at the bottom

This code will supply your **devices** TableView with data from the **wiFiDevicesInfo** array

```
extension ViewController: UITableViewDataSource, UITableViewDelegate {

         func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
                 guard let deviceCount = wiFiDevicesInfo?.count else { return 0 }

                 return deviceCount
         }

         func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
                 guard
                         let devices = wiFiDevicesInfo,
                         let cell = tableView.dequeueReusableCell(withIdentifier: "networkDevice")
                 else { return UITableViewCell() }

                 cell.textLabel?.text = devices[indexPath.row].strModelName

                 return cell
         }

}
```

Modify the **viewDidLoad** method in the **ViewController** so that it is similar to the right (you can copy and paste)

Notice that this code initializes and configures **networkManager** to search for printers

Add the function **fillDeviceList(manager:)** below the **viewDidLoad** function. This function calls the **startSearch** method in the **BRPtouchNetworkManager** class, which will search your network for Brother devices for two seconds.

Try running the project in the simulator and notice that after a brief delay Brother compatible printers will be displayed

Later in this document you will program the **Refresh Network Device List** button to find network devices on demand

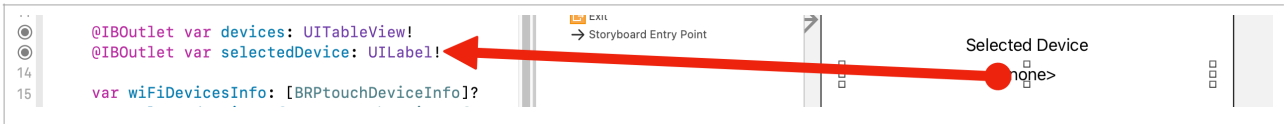
```
override func viewDidLoad() {
      super.viewDidLoad()

      let manager = BRPtouchNetworkManager()
      manager.delegate = self
      manager.isEnableIPv6Search = false
      manager.setPrinterNames(allBrotherPrinters())

      self.networkManager = manager
      fillDeviceList(manager: networkManager!)

      devices.delegate = self
      devices.dataSource = self
}

func fillDeviceList(manager: BRPtouchNetworkManager) {
      manager.startSearch(2)
}
```

SELECTING A PRINTER

Before sending out a print job, you must target a printer

Drag a outlet from the label below the “Selected Device” to the ViewController code and place it just below the devices outlet—name it selectedDevice

This linked label will display additional information about the printer you have selected



Add the **tableView(_:didSelectRowAt)** method to the **ViewController** extension (you can copy and paste)

This will set the **selectedDeviceInfo** with data from the printer you clicked on and set the **text** property on the **selectedDevice** label

Press the **Build and Run** button to test this in the simulator. Notice that the label below **Selected Device** will display the Brother printer model name and serial number for the printer you tapped on.

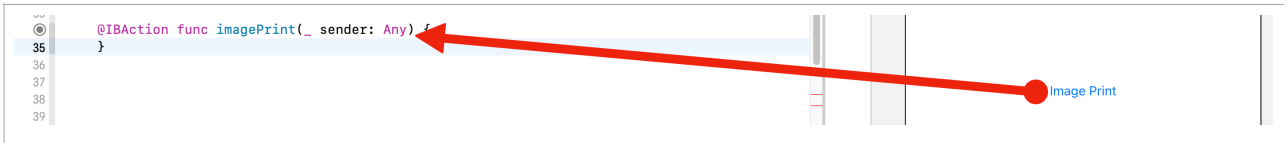
```
extension ViewController: UITableViewDataSource, UITableViewDelegate {

        func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
                guard let deviceCount = wiFiDevicesInfo?.count else { return 0 }

                return deviceCount
        }

        func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
                guard
                        let devices = wiFiDevicesInfo,
                        let cell = tableView.dequeueReusableCell(withIdentifier: "networkDevice")
                else { return UITableViewCell() }

                cell.textLabel?.text = devices[indexPath.row].strModelName

                return cell
        }

        func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
                guard let deviceInfo = wiFiDevicesInfo?[indexPath.row] else { return }

                selectedDeviceInfo = deviceInfo

                selectedDevice.text = deviceInfo.strModelName +
                        ", Serial #" + deviceInfo.strSerialNumber
        }

}
```

PRINTING

Link the **Image Print** button from the storyboard to the **ViewContoller**

Hold down the control key and drag the **Image Print** button link just below the **viewDidLoad** method

Name the action event **imagePrint**



Copy and paste the code below into the **imagePrint** method

This code opens communication with your selected printer, gets a file path to **RedBlackImageSample.png**, configures the print job, and invokes the **printImage** method

This sample code is configured for the **QL-1110NWB** with 62mm label paper. If your printer is different, change the enum by deleting the existing value and typing the model number of your printer (enums require a leading "." to work). **Xcode** will automatically compete the value for you as you type.

Run the app in the simulator and verify that it will print an image.

```
@IBAction func imagePrint(_ sender: Any) {
        guard let selectedDeviceInfo = selectedDeviceInfo else { return }

        imagePrint(selectedPrinter: selectedDeviceInfo)
}

func imagePrint(selectedPrinter: BRPtouchDeviceInfo) {
        let channel = BRLMChannel(wifiIPAddress: selectedPrinter.strIPAddress)
        let openChannelResult = BRLMPrinterDriverGenerator.open(channel)

        guard openChannelResult.error.code == BRLMOpenChannelErrorCode.noError,
                let printerDriver = openChannelResult.driver else {
                        print("Channel Error: \(openChannelResult.error.code.rawValue)")
                        return
        }

        defer {
                printerDriver.closeChannel()
        }

        let imageURL = Bundle.main.url(forResource: "RedBlackImageSample", withExtension: "png")
        let printerSettings = BRLMQLPrintSettings(defaultPrintSettingsWith: .QL_1110NWB)
        printerSettings?.halftone = .errorDiffusion
        printerSettings?.labelSize = .rollW62
        printerSettings?.autoCut = true

        let error = printerDriver.printImage(with: imageURL!, settings: printerSettings!)
        print("Image Sample print - result code: \(error.code.rawValue)")
}
```

Link the **PDF Print** button from the storyboard to the **ViewContoller**

Hold down the control key and drag the **PDF Print** button link just below the **viewDidLoad** method

Name the action event **pdfPrint**

Copy and paste the code on the right into the imagePrint method

This code is nearly identical to the **imagePrint** method

If necessary, change the printer model and paper enums

Run the app in the simulator and verify that it will print a PDF

```
@IBAction func pdfPrint(_ sender: Any) {
	guard let selectedDeviceInfo = selectedDeviceInfo else { return }

	pdfPrint(selectedPrinter: selectedDeviceInfo)
}

func pdfPrint(selectedPrinter: BRPtouchDeviceInfo) {
	let channel = BRLMChannel(wifiIPAddress: selectedPrinter.strIPAddress)
	let openChannelResult = BRLMPrinterDriverGenerator.open(channel)

	guard openChannelResult.error.code == BRLMOpenChannelErrorCode.noError,
		let printerDriver = openChannelResult.driver else {
			print("Channel Error: \(openChannelResult.error.code.rawValue)")
			return
	}

	defer {
		printerDriver.closeChannel()
	}

	let pdfURL = Bundle.main.url(forResource: "PDFSample", withExtension: "pdf")
	let printerSettings = BRLMQLPrintSettings(defaultPrintSettingsWith: .QL_820NWB)
	printerSettings?.labelSize = .rollW62
	printerSettings?.autoCut = true

	let error = printerDriver.printPDF(with: pdfURL!, settings: printerSettings!)
	print("PDFSample print - result code: \(error.code.rawValue)")
}
```

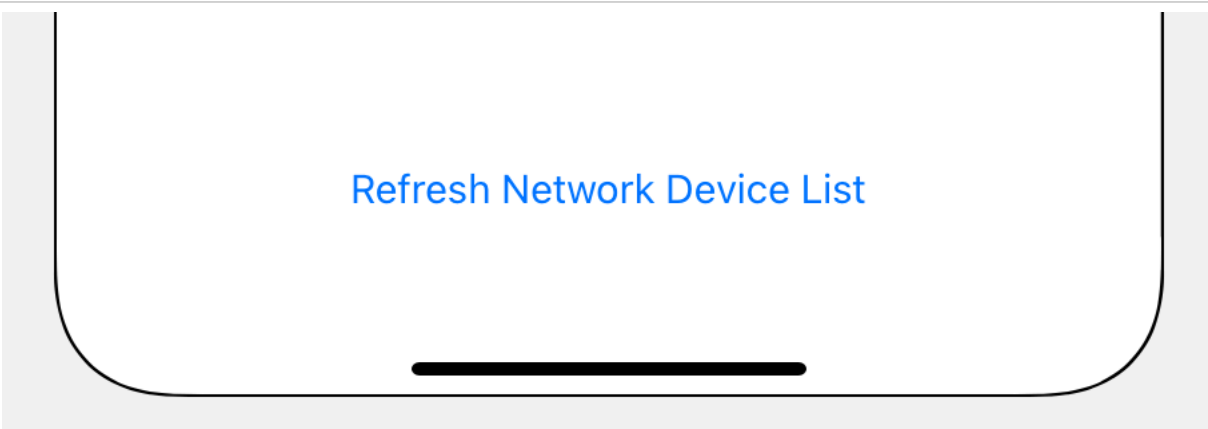
REFRESHING THE NETWORK PRINTER LIST

Use your new skills to program the **Refresh Network Device List** button.

Begin by linking the button to your project code, just like you did with the **Image Print** and **PDF Print** buttons

Inside of the IBAction function, make a call to the **fillDeviceList** function (hint: you have already done this once in your code)

The next page details the one line of code necessary to make this button work. Try building this on your own before peeking!



Below is a working version of the **Refresh Network Device List** button. The button action is named “findPrinters”, but you may have used a different name in the prior page.

```
@IBAction func findPrinters(_ sender: Any) {
    fillDeviceList(manager: networkManager!)
}
```

Congratulations! You now have a foundation for creating great Brother printer apps!