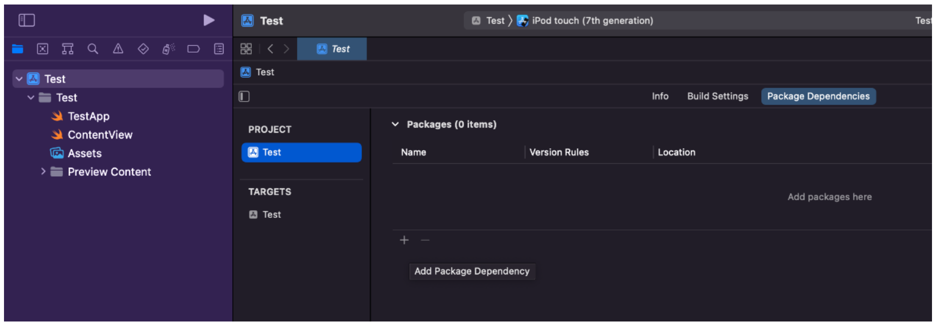
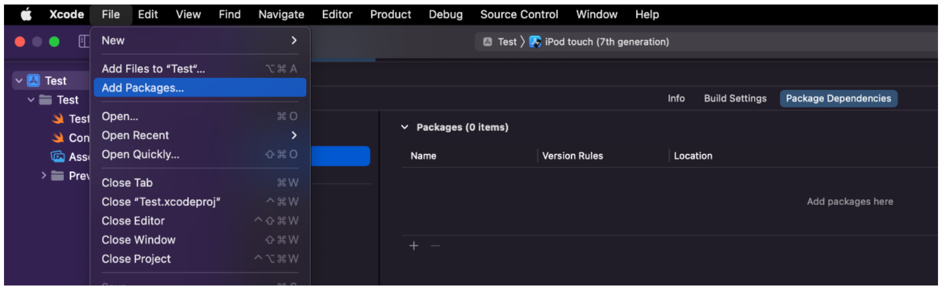
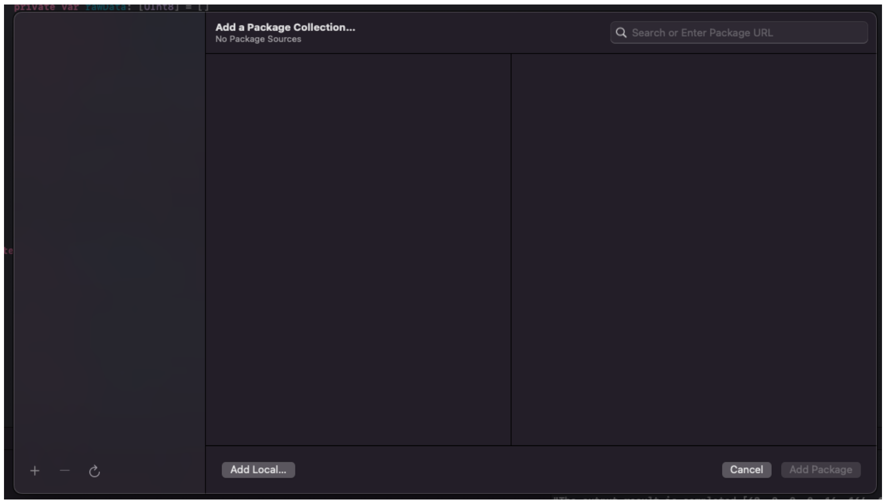
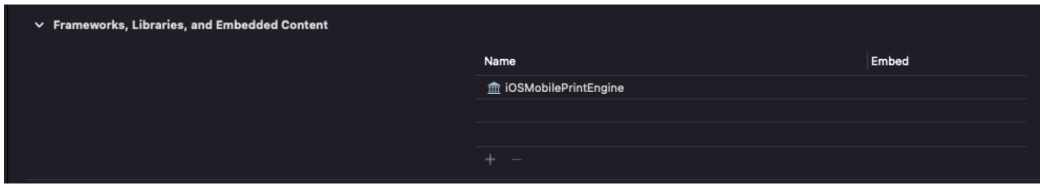
iOS SDK Setup and Test App Instructions - Version 1.0

1. Create a new XCode Application
2. Select the project file:
   a. 

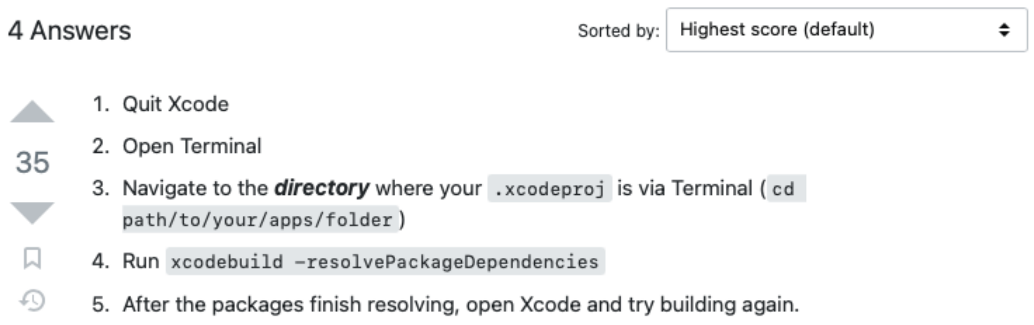
3. File >> Add Packages **OR** select the plus button in the "Packages" window
   a. 

4. Select the "Add Local" button and add the SDK folder that you were given.
   a. 

5. Once the package is added, click the project header (the “app” symbol) in the file tree in XCode and select your app under “Targets”. In the Frameworks section, you may now add the SDK as a library.



6. You may now try to use “import iOSMobilePrintEngine” in a swift file to see if you can access the library. If you get build errors from other package dependencies you may follow this.



4 Answers

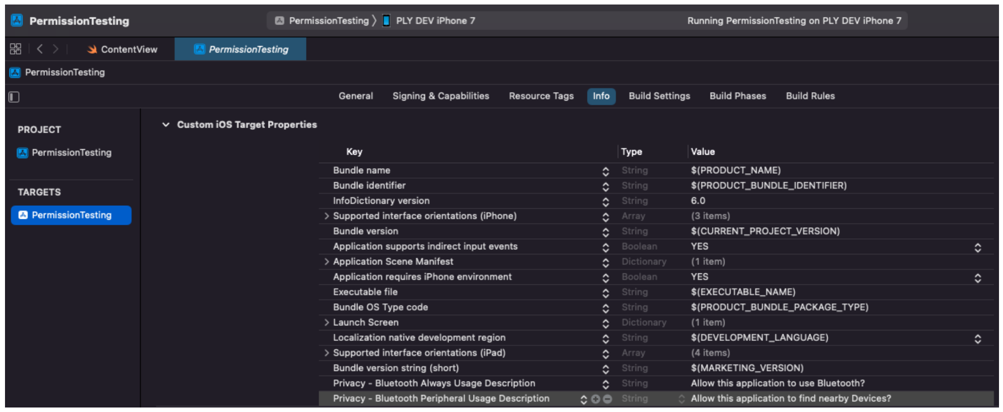
Sorted by: Highest score (default)

35

1. Quit Xcode
2. Open Terminal
3. Navigate to the ***directory*** where your `.xcodeproj` is via Terminal (`cd path/to/your/apps/folder`)
4. Run `xcodebuild -resolvePackageDependencies`
5. After the packages finish resolving, open Xcode and try building again.

Adding Bluetooth Permissions

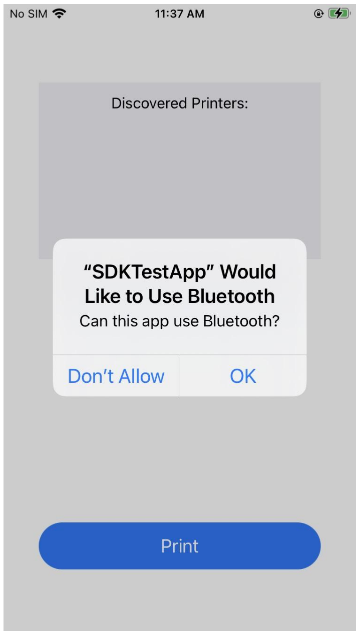
- In order to use the Bluetooth libraries within the SDK, you must let the user allow these permissions.
- Click the project header in the file tree in XCode and select your app under "Targets". Navigate to the "Info" tab and add them underneath "Custom iOS Target Properties".



Custom iOS Target Properties

Key	Type	Value
Bundle name	String	$(PRODUCT_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Supported interface orientations (iPhone)	Array	(3 items)
Bundle version	String	$(CURRENT_PROJECT_VERSION)
Application supports indirect input events	Boolean	YES
Application Scene Manifest	Dictionary	(1 item)
Application requires iPhone environment	Boolean	YES
Executable file	String	$(EXECUTABLE_NAME)
Bundle OS Type code	String	$(PRODUCT_BUNDLE_PACKAGE_TYPE)
Launch Screen	Dictionary	(1 item)
Localization native development region	String	$(DEVELOPMENT_LANGUAGE)
Supported interface orientations (iPad)	Array	(4 items)
Bundle version string (short)	String	$(MARKETING_VERSION)
Privacy - Bluetooth Always Usage Description	String	Allow this application to use Bluetooth?
Privacy - Bluetooth Peripheral Usage Description	String	Allow this application to find nearby Devices?

- The first time you install your app and use Bluetooth you will now receive this pop up asking for confirmation.



"SDKTestApp" Would Like to Use Bluetooth

Can this app use Bluetooth?

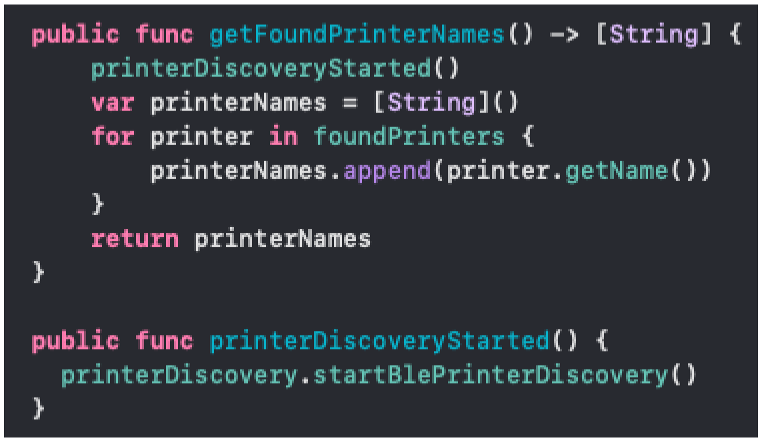
Implementation Tutorial

1. Create a new project and select "App" template to create it
2. We will only touch 2 files in this tutorial:
   a. ContentView - (anything related to a view on the screen)
   b. Model - (all the function calls we make to the SDK directly)
3. We must initialize our Model in the first line of our ContentView as such:
   a. @ObservedObject var model = Model()
4. Model will inherit the PrinterUpdateListener and PrinterDiscoveryListener interfaces and ObservableObject.
   a. public class Model: PrinterUpdateListener, PrinterDiscoveryListener, ObservableObject {
   b. WARNING: Add "import iOSMobilePrintEngine" to the top of this file if they are not showing up.
5. If you try to build, you should get a warning that allows you to include these interfaces methods. Add these.
6. We will initialize 2 crucial variables in Model.swift
   a. private var printerDiscovery: PrinterDiscovery = PrinterDiscoveryFactory.getPrinterDiscovery(listeners: [PrinterDiscoveryListeners](self))
      i. We will use this to discover and connect to a printer
   b. @Published private var foundPrinters = [DiscoveredPrinterInformation]()

i. This will be a list of printers that were found nearby. Since this is a “Published” variable, as soon as its contents are changed, it automatically notifies our UI because of the Model’s @ObservableObject tag

7. **Discovering Printers:**

   a. Make a function called getFoundPrinterNames() in Model.swift

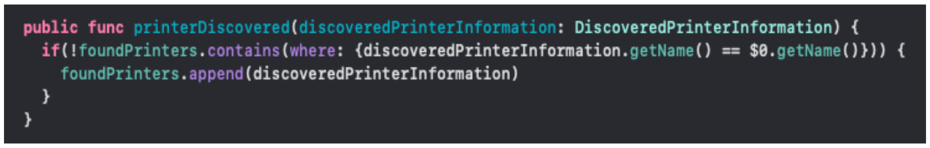


```
public func getFoundPrinterNames() -> [String] {
    printerDiscoveryStarted()
    var printerNames = [String]()
    for printer in foundPrinters {
        printerNames.append(printer.getName())
    }
    return printerNames
}

public func printerDiscoveryStarted() {
  printerDiscovery.startBlePrinterDiscovery()
}
```

   b. From ContentView.swift you will call: yourList=model.getFoundPrinterNames() to return a list of just the printer names that you can populate any sort of view with.

   c. When a printer is discovered, it will trigger the printerDiscovered() method in Model where you can filter out printers like:

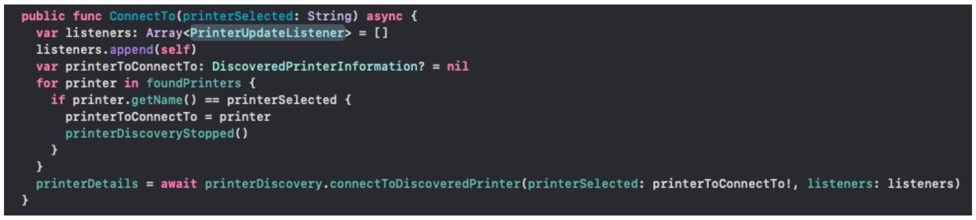


```
public func printerDiscovered(discoveredPrinterInformation: DiscoveredPrinterInformation) {
  if(!foundPrinters.contains(where: {discoveredPrinterInformation.getName() == $0.getName()})) {
    foundPrinters.append(discoveredPrinterInformation)
  }
}
```

8. **Connecting to Printers:**

   a. Now that you are displaying the discovered printers to the user, you can allow them to select one to connect to it. Make a function that takes the selected printer name and find its corresponding DiscoveredPrinterInformation.

   b. Then, we can use the printerDiscovery's connectToDiscoveredPrinter function to connect.



```swift
public func ConnectTo(printerSelected: String) async {
  var listeners: Array<PrinterUpdateListener> = []
  listeners.append(self)
  var printerToConnectTo: DiscoveredPrinterInformation? = nil
  for printer in foundPrinters {
    if printer.getName() == printerSelected {
      printerToConnectTo = printer
      printerDiscoveryStopped()
    }
  }
  printerDetails = await printerDiscovery.connectToDiscoveredPrinter(printerSelected: printerToConnectTo!, listeners: listeners)
}
```

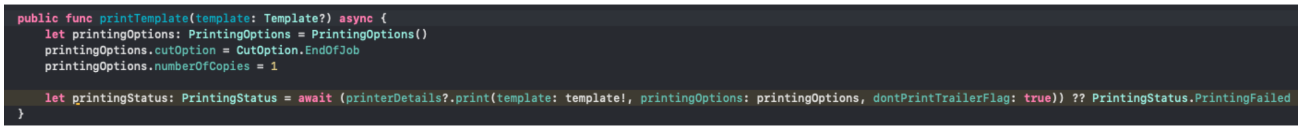
   c. This will return the printer's details if it is connected successfully. This allows you to see the printer's battery life, supply information, and connection status, amongst other useful information.

9. **Printing:**

   a. You need two objects that you will pass into the .print() function.

      i. The template object (look at step 10)

      ii. The specified PrintingOptions you will create such as:



```swift
public func printTemplate(template: Template?) async {
    let printingOptions: PrintingOptions = PrintingOptions()
    printingOptions.cutOption = CutOption.EndOfJob
    printingOptions.numberOfCopies = 1

    let printingStatus: PrintingStatus = await (printerDetails?.print(template: template!, printingOptions: printingOptions, dontPrintTrailerFlag: true)) ?? PrintingStatus.PrintingFailed
}
```

      iii. Pass in true to "dontPrintTrailerFlag" for now

      iv. You may want to add a spinning wheel or progress bar since the .print() is an await method and it takes time to print.

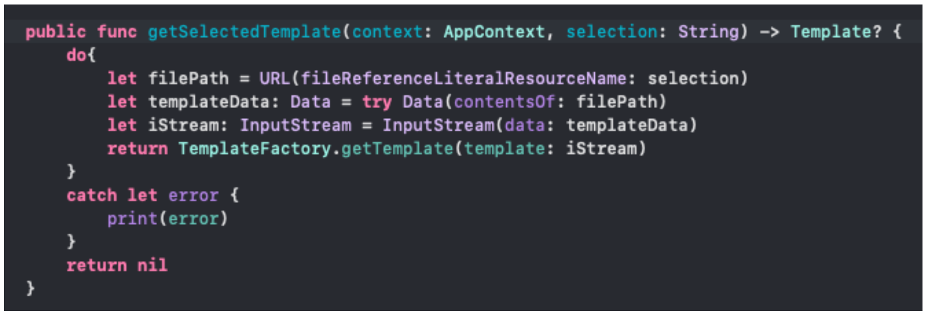
   b. DISCLAIMER: Only shapes and text print properly in this version.

10. **Previewing A Label:**

   a. You must have a local directory that your app can "point to" which holds a collection of .BWT files.

   b. Choose a way to grab the file names and display a list of clickable strings to the user such as a Picker object within a Menu.

   c. When the user clicks a template to preview from this list, create a function in Model.swift that takes that String:



```
public func getSelectedTemplate(context: AppContext, selection: String) -> Template? {
    do{
        let filePath = URL(fileReferenceLiteralResourceName: selection)
        let templateData: Data = try Data(contentsOf: filePath)
        let iStream: InputStream = InputStream(data: templateData)
        return TemplateFactory.getTemplate(template: iStream)
    }
    catch let error {
        print(error)
    }
    return nil
}
```

   d. This function will grab the file matching the String, will convert it to an InputStream, and will pass this into TemplateFactory.getTemplate(template: iStream)

   e. Finally, from ContentView.swift, you can call this function and use the returned template to get a rendered image from the SDK.

      i. let template: Template? = model.getSelectedTemplate(context: AppContext(), selection: selection)

      ii. let bitmap: CGImage? = try template?.getPreview(labelNumber: 1, dpi: 96, maxPixelWidthAndHeight: 288)

11. **Apply Data to Template Objects:**
    a. With the selected template object from the previous step, you can call template.getTemplateData() to get a list of the objects in the template.
    b. You can then call each object's respective methods to make changes such as "setName" or "setValue". More is coming soon!

<u>Software Requirements</u>

- Currently works on iOS 15 and up
- Xcode version 14.1
- Swift 5