iOS 17 App Development Essentials

Neil Smyth

Payload publishing

iOS 17 App Development Essentials

iOS 17 App Development Essentials

ISBN-13: 978-1-951442-80-4

Rev: 1.0

Chapter 17

17. Using Xcode in SwiftUI Mode

When creating a new project, Xcode now provides a choice of creating either a Storyboard or SwiftUI-based user interface for the project. When creating a SwiftUI project, Xcode appears and behaves significantly differently when designing an app project's user interface than the UIKit Storyboard mode.

When working in SwiftUI mode, most of your time as an app developer will be spent in the code editor and preview canvas, which will be explored in detail in this chapter.

17.1 Starting Xcode 15

As with all the examples in this book, the development of our example will take place within the Xcode 15 development environment. If you have not already installed this tool together with the latest iOS SDK, refer first to the *"Installing Xcode 15 and the iOS 17 SDK"* chapter of this book. Assuming the installation is complete, launch Xcode either by clicking on the icon on the dock (assuming you created one) or using the macOS Finder to locate Xcode in the Applications folder of your system.

When launched for the first time, and until you turn off the *Show this window when Xcode launches* toggle, the screen illustrated in Figure 17-1 will appear by default:

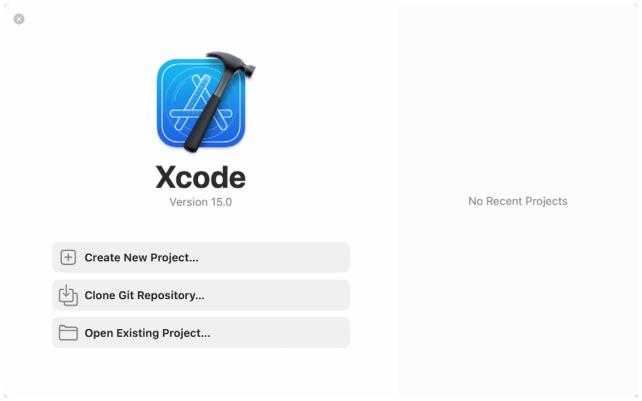


Figure 17-1

If you do not see this window, select the *Window -> Welcome to Xcode* menu option to display it. From within this window, click on *Create New Project.*

17.2 Creating a SwiftUI Project

When creating a new project, the project template screen includes options to select how the app project is to be implemented. Options are available to design an app for a specific Apple platform (such as iOS, watchOS, macOS, DriveKit, or tvOS) or to create a *multiplatform* project. Selecting a platform-specific option will also provide the choice of creating either a Storyboard (UIKit) or a SwiftUI-based project.

A multiplatform project allows an app to be designed for multiple Apple platforms with a minimum of platform-specific code. Even if you plan to initially only target iOS, the multiplatform option is still recommended since it provides the flexibility to make the app available on other platforms in the future without having to restructure the project.

Templates are also available for creating a basic app, a document-based app, or a game project. For this chapter, use the multiplatform app option:

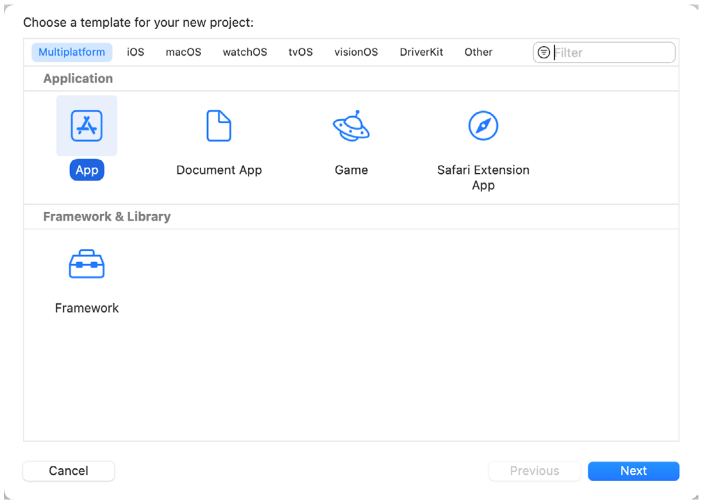
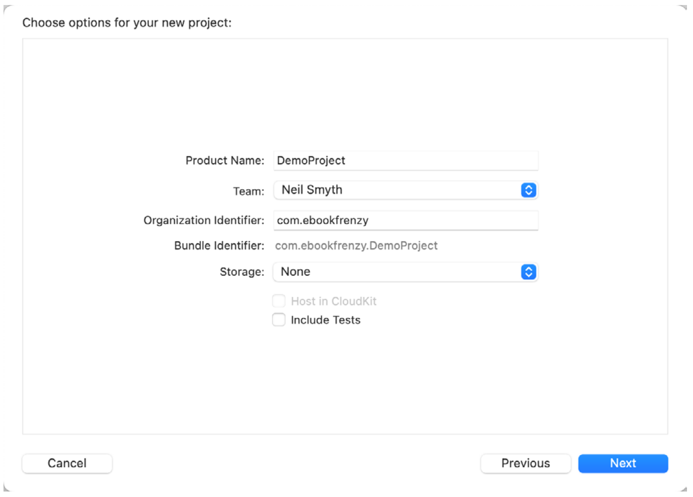


Figure 17-2

Clicking *Next* will display the project options screen where the project name needs to be entered (in this case, name the project *DemoProject*).

The Organization Identifier is typically the reversed URL of your company's website, for example, "com. mycompany". This will be used when creating provisioning profiles and certificates to enable the testing of advanced features of iOS on physical devices. It also serves to uniquely identify the app within the Apple App Store when the app is published:


Choose options for your new project:

Product Name:	DemoProject
Team:	Neil Smyth
Organization Identifier:	com.ebookfrenzy
Bundle Identifier:	com.ebookfrenzy.DemoProject
Storage:	None
	☐ Host in CloudKit
	☐ Include Tests

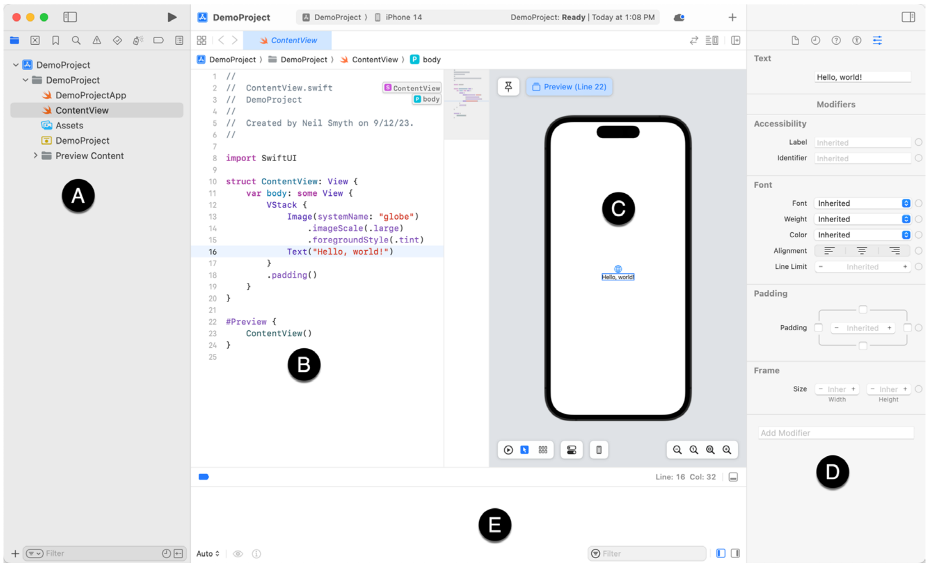
Figure 17-3

Click *Next* again and choose a location on your filesystem to place the project before clicking the *Create* button.

Once a new project has been created, the main Xcode panel will appear with the default layout for SwiftUI development displayed.

17.3 Xcode in SwiftUI Mode

Before beginning work on a SwiftUI user interface, it is worth taking some time to gain familiarity with how Xcode works in SwiftUI mode. A newly created multiplatform "app" project includes two SwiftUI View files named <*app name*>*App.swift* (in this case *DemoProjectApp.swift*) and *ContentView.swift*, which, when selected from the project navigation panel, will appear within Xcode, as shown in Figure 17-4 below:



```
//
//  ContentView.swift
//  DemoProject
//
//  Created by Neil Smyth on 9/12/23.
//

import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {
            Image(systemName: "globe")
                .imageScale(.large)
                .foregroundStyle(.tint)
            Text("Hello, world!")
        }
        .padding()
    }
}

#Preview {
    ContentView()
}
```

Figure 17-4

When the project loads for the first time, Xcode may default to macOS as the destination platform for building and testing your app. Before continuing, use the run destination chooser to select an iPhone device, as illustrated in Figure 17-5 below:

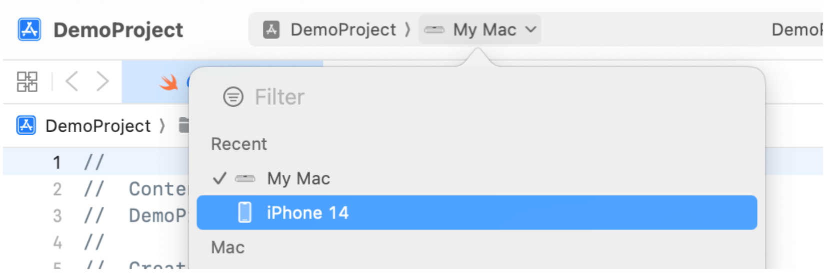


Figure 17-5

Located to the right of the project navigator (A) is the code editor (B). To the right is the preview canvas (C), where any changes made to the SwiftUI layout declaration will appear in real-time.

Selecting a view in the canvas will automatically select and highlight the corresponding entry in the code editor and vice versa. Attributes for the currently selected item will appear in the attributes inspector panel (D).

During debugging, the debug panel (E) will appear, containing debug output from both the iOS frameworks and any diagnostic print statements you have included in your code. If the console is not currently visible, display it by clicking on the button indicated by the arrow in Figure 17-6:

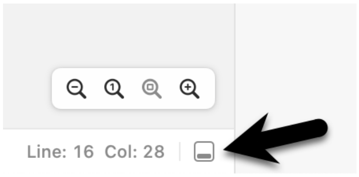
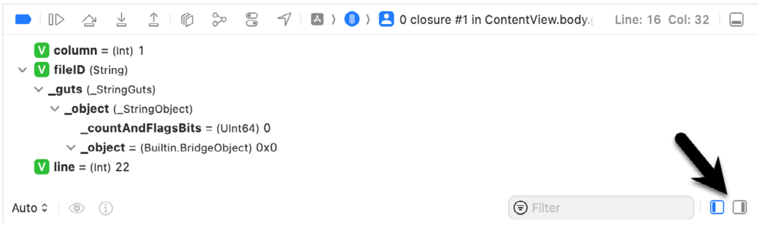


Figure 17-6

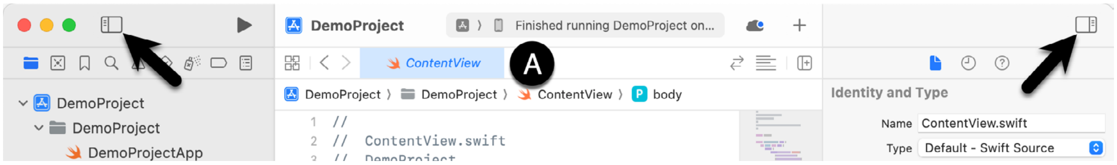
The debug panel can be configured to show a Variables view or a Console view. The variables view displays variables within the app at the point that the app crashes or reaches a debugging breakpoint. The console view, on the other hand, displays print output and messages from the running app. Figure 17-7 shows the variables view with an arrow indicating the buttons used to switch views:



```
V column = (Int) 1
V fileID (String)
  _guts (_StringGuts)
    _object (_StringObject)
      _countAndFlagsBits = (UInt64) 0
      _object = (Builtin.BridgeObject) 0x0
V line = (Int) 22
```

Figure 17-7

The button indicated in Figure 17-6 above may be used to hide the debug panel (E), while the two buttons highlighted in Figure 17-8 below hide and show the project navigator and inspector panels:


DemoProject
Finished running DemoProject on...
ContentView
A
DemoProject
DemoProject
DemoProjectApp
DemoProject 〉 DemoProject 〉 ContentView 〉 body
Identity and Type
Name ContentView.swift
Type Default - Swift Source

```
//
//  ContentView.swift
//  DemoProject
```

Figure 17-8

The tab bar (marked A above) displays a tab for each currently open file. When a tab is clicked, the corresponding file is loaded into the editor. Hovering the mouse pointer over a tab will display an "X" button within the tab which will close the file when clicked.

The area marked F in Figure 17-4 is called the *Minimap*. This map provides a miniaturized outline of the source code in the editor. Particularly useful when working with large source files, the minimap panel provides a quick way to move to different areas of the code. Hovering the mouse pointer of a line in the Minimap will display a label indicating the class, property, or function at that location, as illustrated in Figure 17-9:

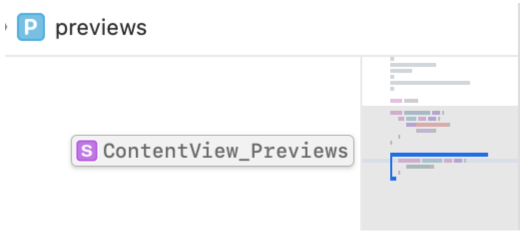


Figure 17-9

Clicking on the label or within the map will take you to that line in the code editor. Holding down the Command key while hovering will display all of the elements contained within the source file, as shown in Figure 17-10:

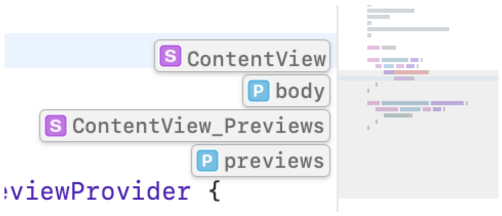


Figure 17-10

The Minimap can be displayed and hidden by toggling the *Editor -> Minimap* menu option.

17.4 The Preview Canvas

The preview canvas provides both a visual representation of the user interface design and a tool for adding and modifying views within the layout design. The canvas may also be used to perform live testing of the running app without launching an iOS simulator. Figure 17-11 illustrates a typical preview canvas for a newly created project:

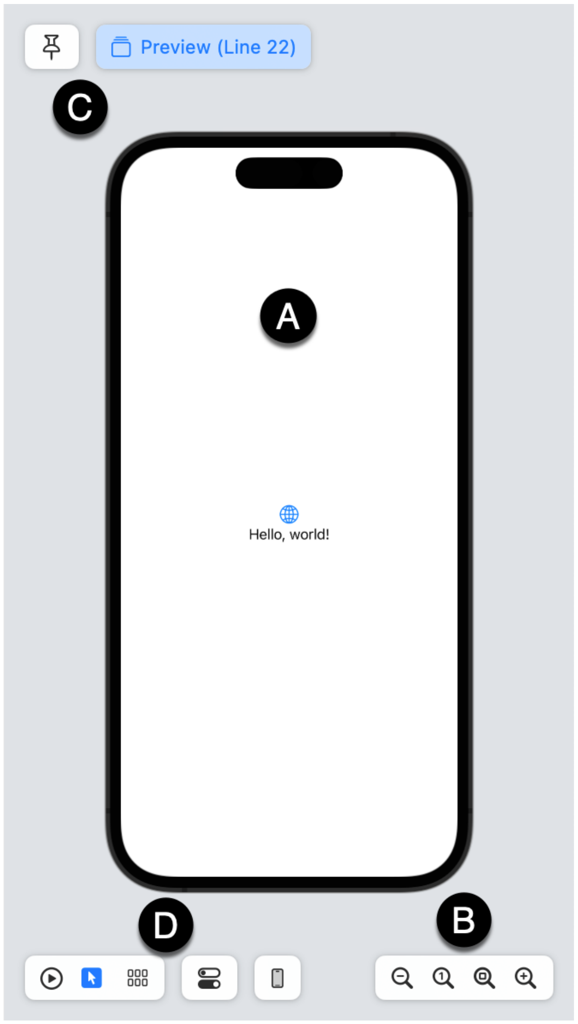


Figure 17-11

If the canvas is not visible, it can be displayed using the Xcode *Editor -> Canvas* menu option.

The main canvas area (A) represents the current view as it will appear when running on a physical device (also referred to as the *canvas device*). When changes are made to the code in the editor, those changes are reflected within the preview canvas. To avoid continually updating the canvas, and depending on the nature of the changes being made, the preview will occasionally pause live updates. When this happens, the Resume button will appear, which, when clicked, will once again begin updating the preview:

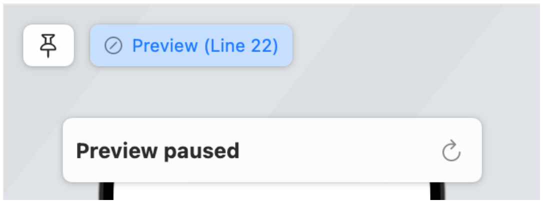


Figure 17-12

The size buttons (B) will zoom in and out of the canvas, zoom to 100%, and fit the available space.

17.5 Preview Pinning

Building an app in Xcode will likely consist of several SwiftUI View files in addition to the default *ContentView.swift* file. When a SwiftUI View file is selected from the project navigator, both the code editor and preview canvas will change to reflect the currently selected file. Sometimes, you may want the user interface layout for one SwiftUI file to appear in the preview canvas while editing the code in a different file. This can be particularly useful if the layout from one file is dependent on or embedded in another view. The pin button (labeled C in Figure 17-11 above) pins the current preview to the canvas so that it remains accessible on the canvas after navigating to a different view. Switch between pinned views by clicking on the view buttons along the top of the preview panel. In Figure 17-13, for example, buttons are provided to switch between two views:

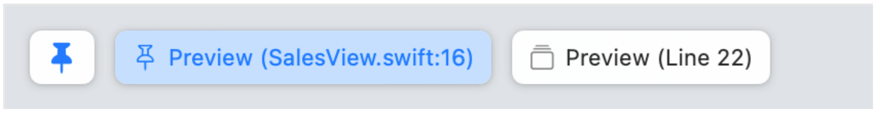


Figure 17-13

The #Preview macro can be used to apply more meaningful names to the views as follows:

```
#Preview("Content View") {
    ContentView()
}

#Preview("Sales View") {
    SalesView()
}
```

When the above changes are applied to the *ContentView.swift* and *SalesView.swift* files, the view buttons in the preview canvas will appear, as illustrated below:

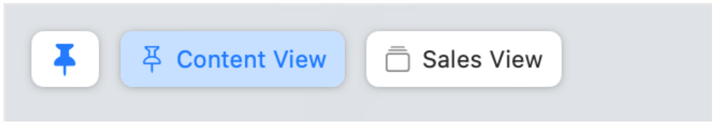


Figure 17-14

17.6 The Preview Toolbar

The preview toolbar (marked D in Figure 17-11 above and shown below) provides several options for changing the preview panel:

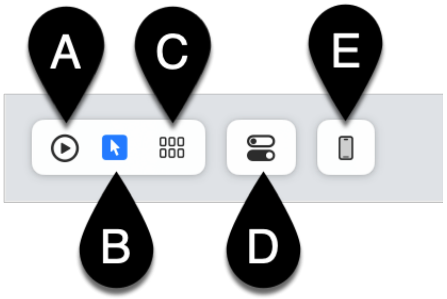


Figure 17-15

The preview panel can display the current view in either *live* or *selectable* mode. In live mode, the app runs interactively within the canvas and allows you to interact with it as if running on a device or simulator. Selectable mode allows you to select items in the canvas view to edit attributes and also to add and remove components. To switch between modes, use the buttons marked A (for live mode) and B (for selectable mode) as marked in the above figure.

The current version of the app may also be previewed on different device models by clicking on the Preview Device button (E). If you have a physical device connected to Xcode, selecting it from the menu will launch the app on that device. As with the preview canvas, the running app on the device will update dynamically as changes are made to the code in the editor.

The Variants button (C) allows the view to be previewed in a variety of dynamic text sizes, color schemes, and device orientations:

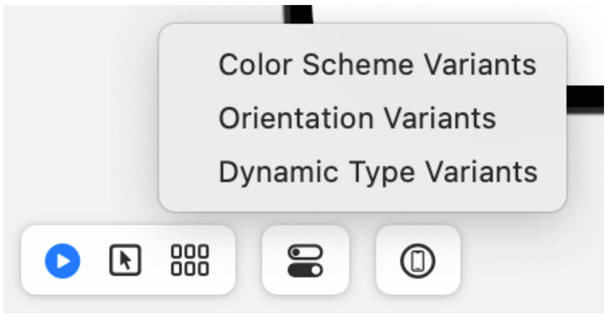


Figure 17-16

Figure 17-17, for example, shows the result of selecting the orientation variations option:

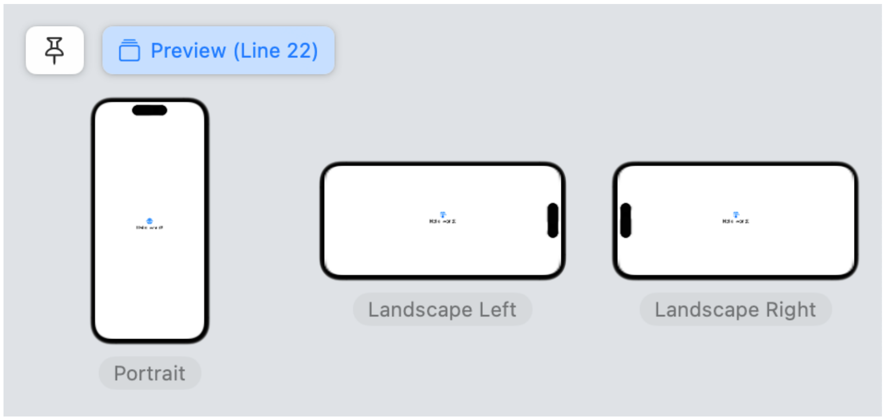


Figure 17-17

The Device Settings button (D) controls the color scheme (light or dark), orientation (portrait or landscape), and dynamic font size of the canvas device:

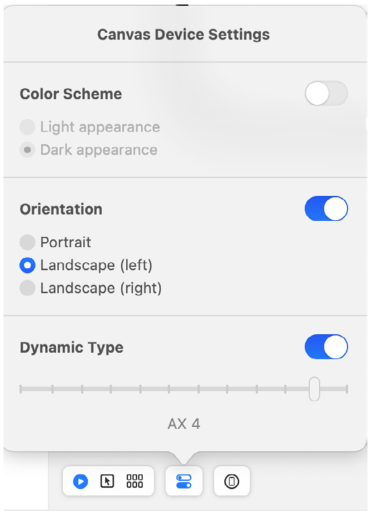


Figure 17-18

17.7 Modifying the Design

Working with SwiftUI primarily involves adding additional views, customizing those views using modifiers, adding logic, and interacting with state and other data instance bindings. All of these tasks can be performed exclusively by modifying the structure in the code editor. The font used to display the "Hello, world!" Text view, for example, can be changed by adding the appropriate modifier in the editor:

```
Text("Hello, world!")
    .font(.largeTitle)
```

An alternative to this is to change the SwiftUI views by dragging and dropping items from the Library panel. The Library panel is displayed by clicking on the toolbar button highlighted in Figure 17-19: