Article

Editing source files in Xcode

Edit source files in Xcode and add Quick Help comments to improve your project's maintainability.

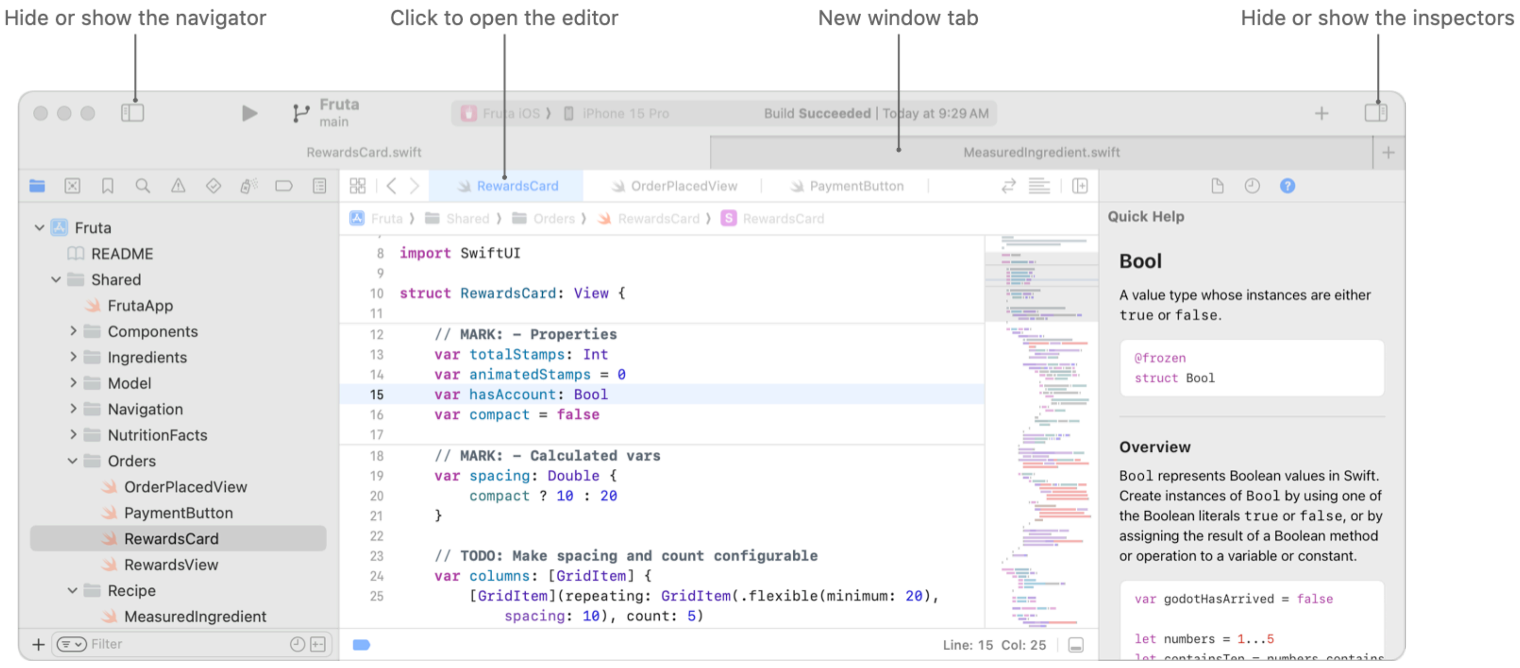
Overview

Set up your Xcode project window to efficiently edit multiple related files at the same time. While you edit, annotate your files so you can easily determine a file's content with the minimap or jump bar. Then add comments that Xcode can present as Quick Help to make your code easier to understand.

For information about how to add files and folders to your project, see Managing files and folders in your Xcode project.

Edit related files using tabbed windows

Edit related files together using tabbed windows in Xcode. To open a new window tab with the same configuration as your current window, press Command-T. To open existing files in tabs within a single window, select one or more files, and choose File > Open in New Window. Then choose Window > Merge All Windows.



```
import SwiftUI

struct RewardsCard: View {

    // MARK: - Properties
    var totalStamps: Int
    var animatedStamps = 0
    var hasAccount: Bool
    var compact = false

    // MARK: - Calculated vars
    var spacing: Double {
        compact ? 10 : 20
    }

    // TODO: Make spacing and count configurable
    var columns: [GridItem] {
        [GridItem](repeating: GridItem(.flexible(minimum: 20),
            spacing: 10), count: 5)
```

Bool

A value type whose instances are either `true` or `false`.

```
@frozen
struct Bool
```

Overview

`Bool` represents Boolean values in Swift. Create instances of `Bool` by using one of the Boolean literals `true` or `false`, or by assigning the result of a Boolean method or operation to a variable or constant.

```
var godotHasArrived = false

let numbers = 1...5
```

Next, configure each tab for how you plan to use it. To give each tab a name that describes how you use it, choose Window > Rename Window Tab.

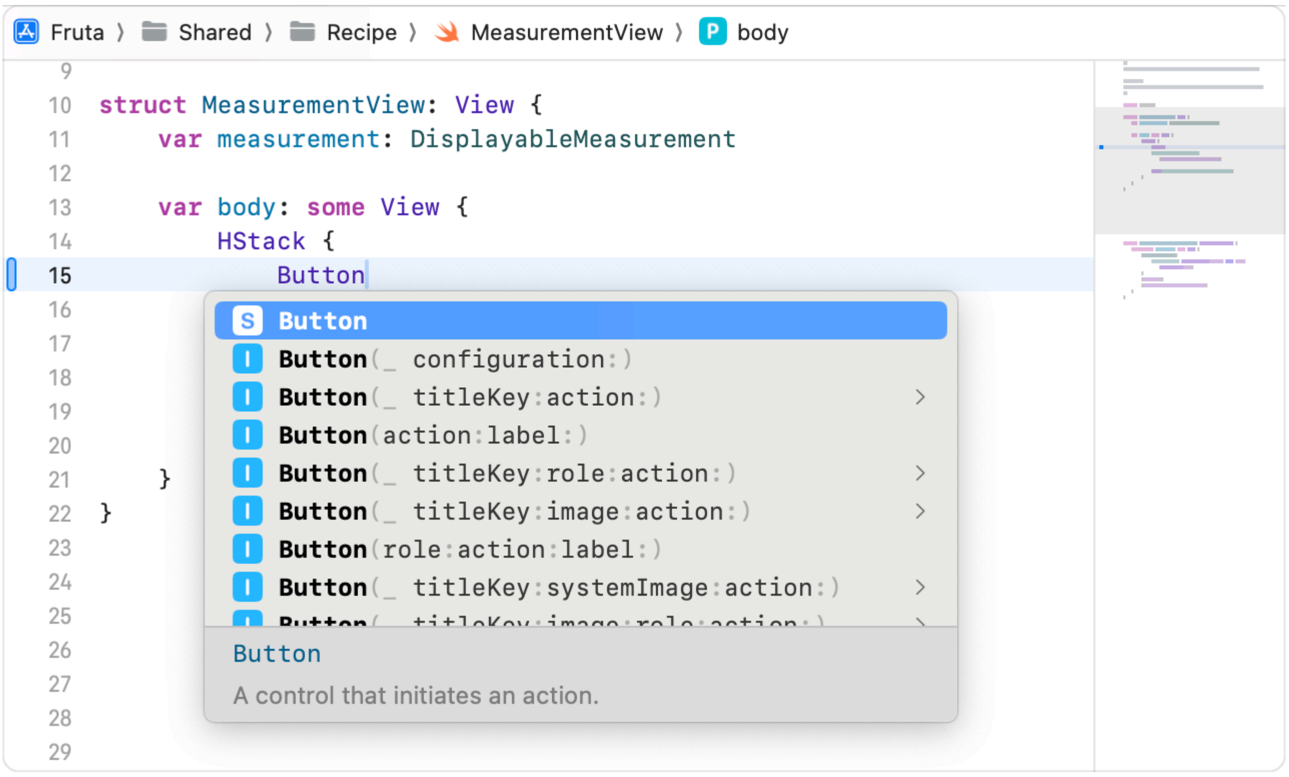
Inside a window or window tab, configure the workspace for your task:

- Show or hide the navigators to the left of the workspace.
- Show or hide the inspectors to the right of the workspace.
- Click the Adjust Editor Options button to show or hide the canvas for SwiftUI code, the assistant for editing related files or Interface Builder files, the minimap, authors, code coverage statistics, and invisible characters.

From the navigators, click a file to open it in the editor area. Xcode displays the filename in italics on the tab, indicating that the next file you click opens in the same tab. Double-click a file to open it in a new tab. Xcode displays the filename in regular text on the tab, indicating that the next file you click or double-click opens in a new tab.

Edit your source code

To open a source code file in the source editor, click the file in the Project navigator. Add code in the editor area. As you type, use code completion to assist with variable and function names.


Fruta
Shared
Recipe
MeasurementView
body

```
struct MeasurementView: View {
    var measurement: DisplayableMeasurement

    var body: some View {
        HStack {
            Button
    }
}
```

Button
Button(_ configuration:)
Button(_ titleKey:action:)
Button(action:label:)
Button(_ titleKey:role:action:)
Button(_ titleKey:image:action:)
Button(role:action:label:)
Button(_ titleKey:systemImage:action:)
Button(_ titleKey:image:role:action:)
Button
A control that initiates an action.

When you use code completion for a method or function with parameters, Xcode provides a placeholder for each parameter that you need to add. Navigate between the placeholders with Tab and Shift-Tab, or choose Navigate > Jump to Next Placeholder or Navigate > Jump to Previous Placeholder.

In Swift files, predictive code completion offers code suggestions at your insertion point. When you type, or press enter to go to a new line, Xcode predicts code for that location. For example, if you type “Image” into the source editor, Xcode predicts the rest of the code to finish the statement. When the code-completion window displays, use the Tab key to fill in the predictive code-completion suggestion, or Return to accept the regular code completion.

> **Note**
>
> Predictive code completion requires Xcode 16 and macOS 15 or later running Apple silicon, where it is automatically enabled. You can disable this feature in Xcode > Settings > Text Editing > Editing.

For common code structures, Xcode provides snippets you can add to your code. Click the Library button (+) in the upper-right corner of the project window, select the “Show the Snippets library” option, then select a snippet and drag it to the desired location in the editor. Navigate between the placeholders to customize the snippet for your needs.

To add your own code snippet:

1. Select some code for the snippet, then Control-click and choose Create Code Snippet. Alternatively, choose Editor > Create Code Snippet.

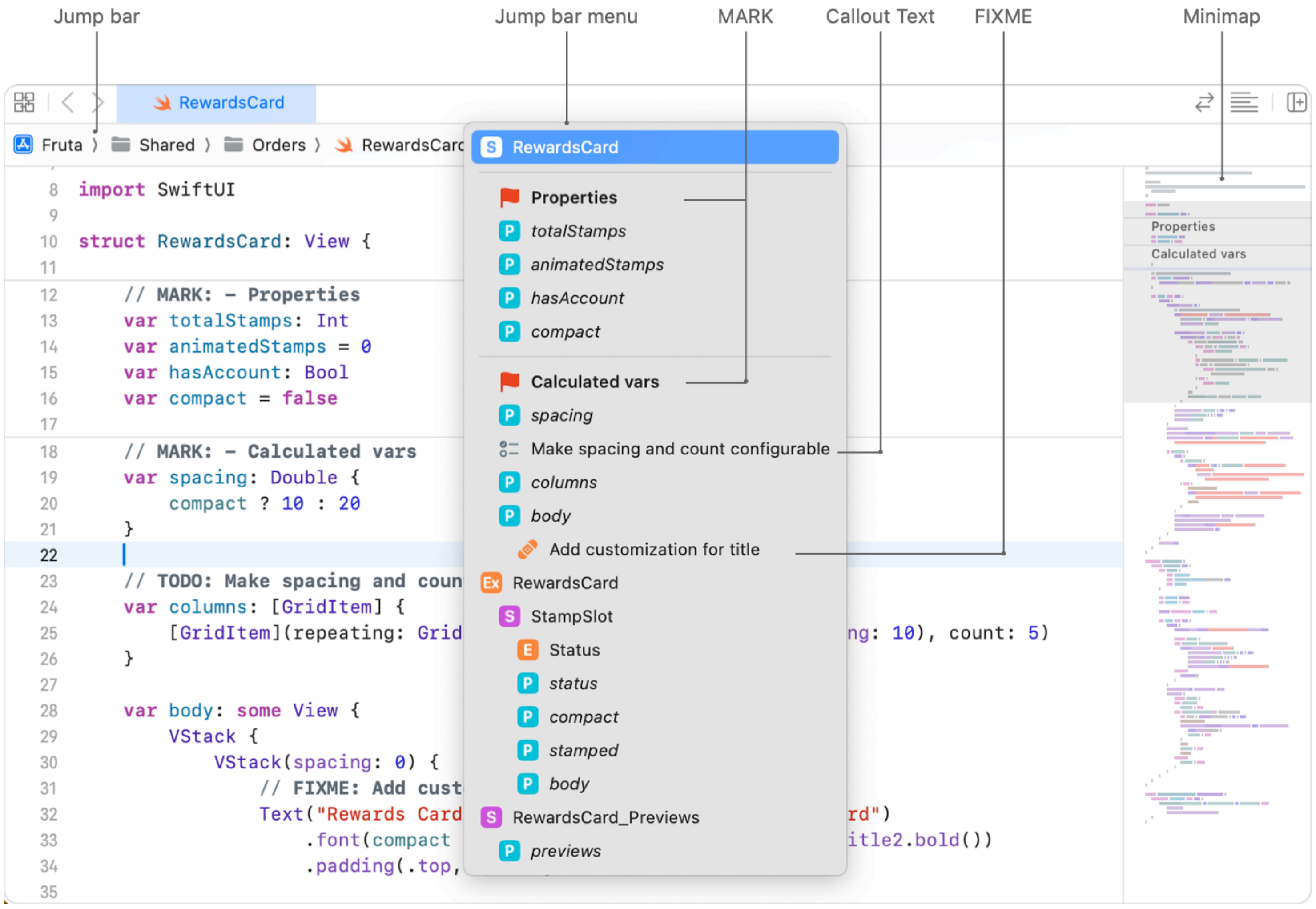
2. Specify a name and summary for your snippet.

3. If you don't select code when creating the snippet, enter code into the snippet editor. Mark any placeholders in your snippet with `<#placeholder name#>`.

4. Select the language and platform, enter any text to use for code completion, select the scope, and click Done.

Annotate your code for visibility

The jump bar and the minimap each provide a quick visual way to navigate your code in the Xcode source editor. Annotate your code with `MARK`, `TODO`, and `FIXME` comments to enhance the power of these tools when organizing your code.



```
import SwiftUI

struct RewardsCard: View {

    // MARK: - Properties
    var totalStamps: Int
    var animatedStamps = 0
    var hasAccount: Bool
    var compact = false

    // MARK: - Calculated vars
    var spacing: Double {
        compact ? 10 : 20
    }

    // TODO: Make spacing and coun
    var columns: [GridItem] {
        [GridItem](repeating: Grid                         ng: 10), count: 5)
    }

    var body: some View {
        VStack {
            VStack(spacing: 0) {
                // FIXME: Add cust
                Text("Rewards Card                        rd")
                    .font(compact                         itle2.bold())
                    .padding(.top,
```

Add a MARK comment to add a heading to a section of code. Include a dash in the comment to instruct Xcode to show a divider line before the section in the jump bar and minimap.

```
// MARK: - Properties

/// A Boolean value that indicates whether the card has an account.
var hasAccount: Bool
```

Add a TODO comment to indicate where you want to do future work. The jump bar highlights the TODO comment with an icon for easy identification.

```
// TODO: Make spacing and count configurable.
var columns: [GridItem] {
    [GridItem](repeating: GridItem(.flexible(minimum: 20), spacing: 10), count: 5)
}
```

Add a FIXME comment to note where you need to make a fix in your code. The jump bar highlights the FIXME comment with a different icon.

```
// FIXME: Add customization for title.
Text("Rewards Card", comment: "Header for rewards card")
```

In addition to the organizational benefits, consistent use of MARK, TODO, and FIXME comments improves your ability to search for common sections, future updates, and needed fixes.

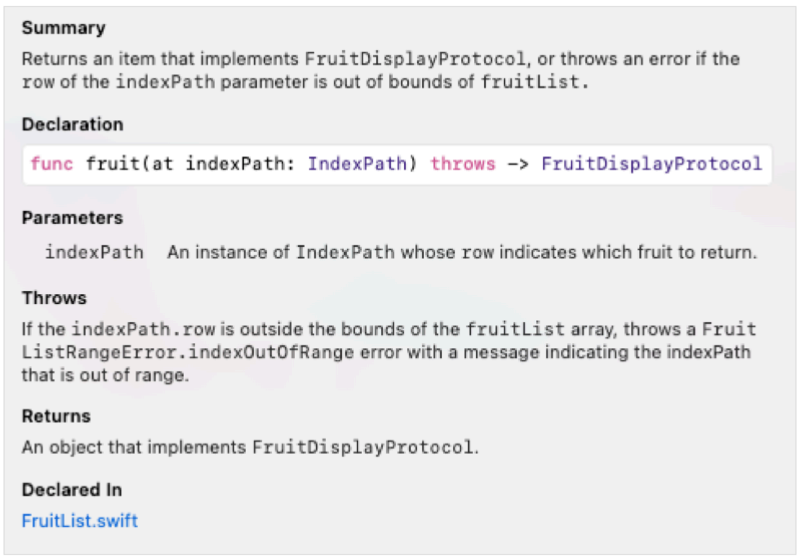
Add Quick Help comments to your code

View Quick Help comments for a symbol by selecting it and choosing View > Inspectors > Quick Help. Alternatively, Control-click a symbol and select Show Quick Help. Add Quick Help comments to your code to clarify usage for other developers.

To add Quick Help to a symbol in your code, hover over the symbol declaration, Command-click it and choose Add Documentation. Xcode adds lines of comments, preceded with ///, that have placeholders for a description, parameters, throws, and a return value, depending on the symbol declaration. Update the placeholders using Markup syntax to complete the comments and enable Quick Help for the symbol.

```
/// Returns an item that implements `FruitDisplayProtocol`, or throws an error if th
/// - Parameter indexPath: An instance of `IndexPath` whose `row` indicates which fr
/// - Throws: If the `indexPath.row` is outside the bounds of the `fruitList` array,
/// - Returns: An object that implements `FruitDisplayProtocol`.
func fruit(at indexPath: IndexPath) throws -> FruitDisplayProtocol {
    guard indexPath.row >= fruitList.startIndex && indexPath.row < fruitList.endInde
        throw FruitListRangeError.indexOutOfRange("Fruit not found for index path: \
    }
    return fruitList[indexPath.row]
}
```

Xcode formats and displays the information as Quick Help when you view it.



Summary

Returns an item that implements `FruitDisplayProtocol`, or throws an error if the row of the `indexPath` parameter is out of bounds of `fruitList`.

Declaration

```
func fruit(at indexPath: IndexPath) throws -> FruitDisplayProtocol
```

Parameters

`indexPath` An instance of `IndexPath` whose `row` indicates which fruit to return.

Throws

If the `indexPath.row` is outside the bounds of the `fruitList` array, throws a `FruitListRangeError.indexOutOfRange` error with a message indicating the indexPath that is out of range.

Returns

An object that implements `FruitDisplayProtocol`.

Declared In

FruitList.swift

Xcode also formats Quick Help from multiline comments that begin with `/**` and end with `*/`. For more information about the Markup syntax for Quick Help, see Markup Formatting Reference.

See Also

Source file creation, organization, and editing

Running code snippets using the playground macro

Add playgrounds to your code that run and display results in the canvas.