

[Vision](#) / Locating and displaying recognized text

## Sample Code

# Locating and displaying recognized text

Perform text recognition on a photo using the Vision framework's text-recognition request.

[Download](#)

iOS 18.0+ | iPadOS 18.0+ | Xcode 16.0+

## Overview

This sample code project demonstrates the Vision framework's ability to perform optical character recognition (OCR) on an image you capture using your device's camera. The `RecognizeTextRequest` structure generates a collection of objects that extract and describe an image's textual content. These objects provide information like the text string, the confidence of the observation's accuracy, and the bounding box around the text's location.

Along with extracting and displaying text from the image, the sample app helps you visualize where an observation occurs by using the bounding boxes to draw red rectangles around the text.

## Configure the sample code project

To run this sample app, you need the following:

- Xcode 16 or later
- iPhone with iOS 18 or later

Connect the iPhone to your Mac over USB-C. The first time you run this sample app, the system prompts you to grant the app access to the camera. You need to allow the sample app to access the camera for it to function correctly.

# Set up the camera

The sample app performs the request on a photo from a physical device. To enable use of the camera, the sample uses [AVFoundation](#) to create a custom camera interface. Tapping the Take a Photo button on the initial view presents the camera and calls the setup method. This is where the camera work begins in the sample.

```
CameraPreview(camera: $camera)
    .task {
        /// If the app has access to the camera, set up and display a live capture p
        if await camera.checkCameraAuthorization() {
            didSetup = camera.setup()
        }
        /// If the app doesn't have access, dismiss the camera and display an error.
        } else {
            showAccessError = true
            showCamera = false
        }

        if !didSetup {
            print("Camera setup failed.")
            showCamera = false
        }
    }
}
```

For more information on how to integrate the camera, see [Setting up a capture session](#) and [Capturing still and Live Photos](#).

## Customize the request

The Vision framework provides the ability to customize the way it handles text recognition. Through its text-recognition path, the app demonstrates how to change whether the request prioritizes speed or accuracy. It also shows how to customize the languages the request detects, and whether the request applies a language-correction model during the recognition process.

The two text-recognition paths are: fast and accurate. The fast path is similar to a traditional OCR approach, and the accurate path uses a neural network that's similar to how humans read text. By default, the request uses the accurate path, so the system sets the [recognitionLevel](#) property to accurate.

Depending on the recognition level and language correction settings, the available recognition languages change. To dynamically generate a list of available languages to choose from, the app uses the [supportedRecognitionLanguages](#) method. The app sets the recognition languages with an array of [Locale.Language](#) objects, and prioritizes the first element.

```

func updateRequestSettings() {
    /// A Boolean value that indicates whether the system applies the language-correction
    imageOCR.request.usesLanguageCorrection = languageCorrection

    imageOCR.request.recognitionLanguages = [selectedLanguage]

    switch selectedRecognitionLevel {
    case "Fast":
        imageOCR.request.recognitionLevel = .fast
    default:
        imageOCR.request.recognitionLevel = .accurate
    }
}

```

## Perform the request

After capturing a photo, the app creates an instance of the custom OCR class. This class provides an array to hold the results, the RecognizeTextRequest, and the performOCR method to handle the text recognition. The performOCR method performs the request on the image, which returns an array of RecognizedTextObservation objects. The method then adds each observation to the observations array.

```

@Observable
class OCR {
    /// The array of `RecognizedTextObservation` objects to hold the request's results
    var observations = [RecognizedTextObservation]()

    /// The Vision request.
    var request = RecognizeTextRequest()

    func performOCR(imageData: Data) async throws {
        /// Clear the `observations` array for photo recapture.
        observations.removeAll()

        /// Perform the request on the image data and return the results.
        let results = try await request.perform(on: imageData)

        /// Add each observation to the `observations` array.
        for observation in results {
            observations.append(observation)
        }
    }
}

```

```
}  
}
```

Initially, the app performs the request once when it captures an image. If the app detects any changes to the request settings (for example, the recognition level), it performs the request again. The Vision framework's `perform` method is asynchronous, so the system wraps the method in a `Task` block.

```
.onChange(of: settingChanges, initial: true) {  
    updateRequestSettings()  
    Task {  
        try await imageOCR.performOCR(imageData: imageData)  
    }  
}
```

## Create and display bounding boxes

This sample provides a custom implementation to display red bounding boxes where an observation occurs. An observation contains the location and the dimensions of the `BoundingBox` in the form of a `NormalizedRect`. To create a bounding box, the app first converts the `NormalizedRect` to a `CGRect`, and then returns a `Path` to draw the rectangle.

```
struct Box: Shape {  
    private let normalizedRect: NormalizedRect  
  
    init(observation: any BoundingBoxProviding) {  
        normalizedRect = observation.boundingBox  
    }  
  
    func path(in rect: CGRect) -> Path {  
        let rect = normalizedRect.toImageCoordinates(rect.size, origin: .upperLeft)  
        return Path(rect)  
    }  
}
```

To display the bounding boxes, the app uses the `overlay(alignment:content:)` method on the image, and creates a bounding box for each of the observations.

```
.overlay {  
    ForEach(imageOCR.observations, id: \.uuid) { observation in  
        Box(observation: observation)  
    }  
}
```

```
        .stroke(.red, lineWidth: 1)
    }
}
```

## Access the results

Finally, the app displays the extracted text from the image by iterating through the observations array. If the request doesn't recognize any text in the image, the app displays the "No text recognized" string.

```
/// Display the text from the captured image.
ForEach(imageOCR.observations, id: \.self) { observation in
    Text(observation.topCandidates(1).first?.string ?? "No text recognized")
        .textSelection(.enabled)
}
.foregroundColor(.gray)
```

## See Also

### Text detection

`{}` Recognizing tables within a document

Scan a document containing a contact table and extract the content within the table in a formatted way.

`struct RecognizeDocumentsRequest`

An image-analysis request to scan an image of a document and provide information about its structure.

`struct DocumentObservation`

Information about the sections of content that an image-analysis request detects in a document.

`struct DetectTextRectanglesRequest`

An image-analysis request that finds regions of visible text in an image.

`struct RecognizeTextRequest`

An image-analysis request that recognizes text in an image.

