

[UIKit](#) / [...](#) / [Preparing your UI to run in the background](#) / Using background tasks to update your app

## Article

# Using background tasks to update your app

Configure your app to perform tasks in the background to make efficient use of processing time and power.

## Overview

A *task* is a standalone activity that an app performs, often on a recurring basis. Examples of tasks include performing maintenance on a database, refining a machine learning model, or updating displayed data. You can configure your app to launch and run tasks in the background to take advantage of processing time when the device isn't used.

To schedule a task to run in the background, enable the background modes in Xcode, identify the specific tasks that you need, and then register the tasks with the [BGTaskScheduler](#) object.

## Enable and schedule background tasks

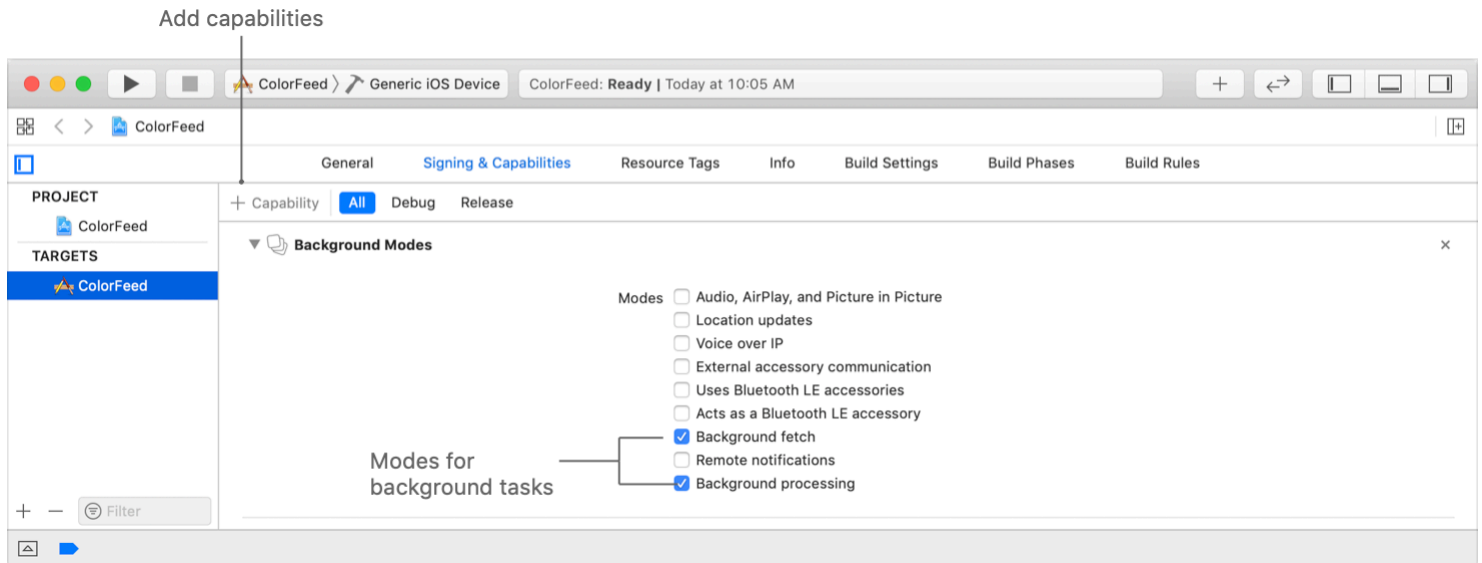
To configure your app to allow background tasks, enable the background capabilities that you need, and then create a list of unique identifiers for each task.

There are two types of background tasks: [BGAppRefreshTask](#) and [BGProcessingTask](#). [BGAppRefreshTask](#) is for short-duration tasks that expect quick results, such as downloading a stock quote. [BGProcessingTask](#) is for tasks that might be time-consuming, such as downloading a large file or synchronizing data. Your app can use one or both of these.

To add the capabilities:

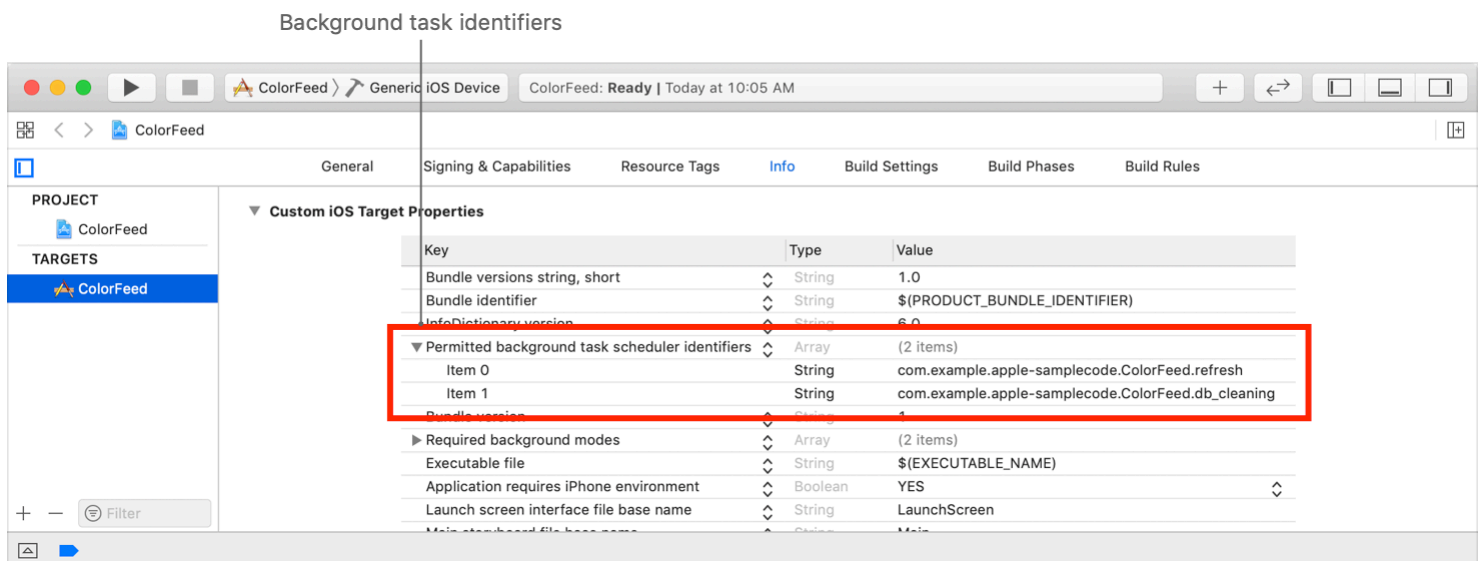
1. Open the project editor and select the desired target.
2. Click Signing & Capabilities.

3. Expand the Background Modes section. If the target doesn't have a Background Modes section, click + Capability, and then select Background Modes.
4. If you're using `BGAppRefreshTask`, select "Background fetch."
5. If you're using `BGProcessingTask`, select "Background processing."



You control which tasks run in the background by registering a list of permitted task identifiers. To create this list, add the identifiers to the `Info.plist` file.

1. Open the project navigator and select your target.
2. Click Info and expand Custom iOS Target Properties.
3. Add a new item to the list and choose "Permitted background task scheduler identifiers," which corresponds to the `BGTaskSchedulerPermittedIdentifiers` array.
4. Add the string for each authorized task identifier as a separate item in the array.



In iOS 13 and later, adding a `BGTaskSchedulerPermittedIdentifiers` key to the `Info.plist` disables the `application(:performFetchWithCompletionHandler:)` and `set`

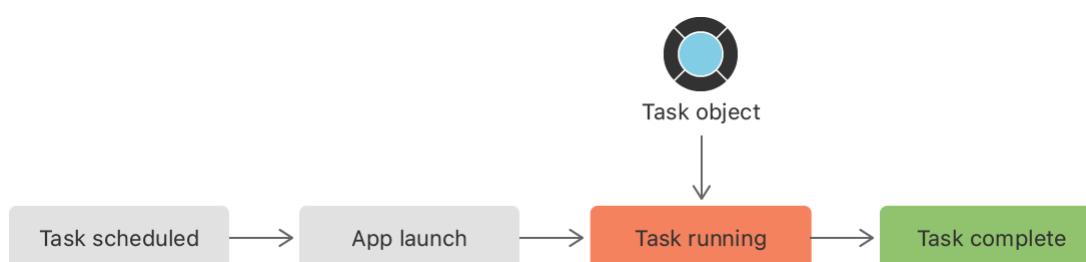
MinimumBackgroundFetchInterval( : ) methods.

## Register, schedule, and run tasks

For each task, provide the BGTaskScheduler object with a *launch handler* — a small block of code that runs the task — and a unique identifier. Register all of the tasks before the end of the app launch sequence. For more information, see About the app launch sequence.

### Note

An extension can schedule a task, but your main app must register the task. The system launches the app to run the task.



The following code registers a handler, `handleAppRefresh(task:)`, that's called when the system runs a task request with the identifier `com.example.apple-samplecode.ColorFeed.refresh`.

```
BGTaskScheduler.shared.register(forTaskWithIdentifier: "com.example.apple-samplecode.  
    self.handleAppRefresh(task: task as! BGAppRefreshTask)  
}
```

To submit a task request for the system to launch your app in the background at a later time, use `submit( : )`. When you resubmit a task, the new submission replaces the previous submission.

The code below schedules a refresh task request for the task identifier `com.example.apple-samplecode.ColorFeed.refresh` that you previously registered.

```
func scheduleAppRefresh() {  
    let request = BGAppRefreshTaskRequest(identifier: "com.example.apple-samplecode.C  
    // Fetch no earlier than 15 minutes from now.  
    request.earliestBeginDate = Date(timeIntervalSinceNow: 15 * 60)  
  
    do {  
        try BGTaskScheduler.shared.submit(request)  
    } catch {
```

```
        print("Could not schedule app refresh: \(error)")
    }
}
```

When the system opens your app in the background, it calls the launch handler to run the task.

Your task provides an expiration handler that the system calls if it needs to terminate your task.

You also add code to inform the system if the task completes successfully.

```
func handleAppRefresh(task: BGAppRefreshTask) {
    // Schedule a new refresh task.
    scheduleAppRefresh()

    // Create an operation that performs the main part of the background task.
    let operation = RefreshAppContentsOperation()

    // Provide the background task with an expiration handler that cancels the operation
    task.expirationHandler = {
        operation.cancel()
    }

    // Inform the system that the background task is complete
    // when the operation completes.
    operation.completionBlock = {
        task.setTaskCompleted(success: !operation.isCancelled)
    }

    // Start the operation.
    operationQueue.addOperation(operation)
}
```

## See Also

### Background execution

 [Extending your app's background execution time](#)

Ensure that critical tasks finish when your app moves to the background.

 [About the background execution sequence](#)

Learn the order in which your custom code is executed when your app moves to the background.