Article

Downloading files in the background

Create tasks that download files while your app is inactive.

Overview

For long-running and nonurgent transfers, you can create tasks that run in the background. These tasks continue to run even when your app is suspended, allowing your app to access the downloaded file when the app is resumed.

> Note
>
> You don’t have to do all background network activity with background sessions as described in this article. Apps that declare appropriate background modes can use default URL sessions and data tasks, just as if they were in the foreground.

Configure the background session

To perform a background download, configure a `URLSession` for background operation. The following example demonstrates this process.

1. Create a background `URLSessionConfiguration` object with the class method `background(withIdentifier:)` of `URLSession`, providing a session identifier that is unique within your app. Because most apps need only a few background sessions (usually one), you can use a fixed string for the identifier, rather than a dynamically generated identifier. The identifier doesn’t need to be unique globally.

2. To have the system to wake up your app when a task completes and the app is in the background, make sure the `sessionSendsLaunchEvents` property is set to `true` (the default).

3. For time-insensitive tasks, enable the `isDiscretionary` property, so the system can wait for optimal conditions to perform the transfer, such as when the device is plugged in or connected to Wi-Fi.

4. Use the `URLSessionConfiguration` instance to create a `URLSession` instance. Provide a delegate, to receive events from the background transfer.

Creating a background URL session

```
private lazy var urlSession: URLSession = {
    let config = URLSessionConfiguration.background(withIdentifier: "MySession")
    config.isDiscretionary = true
    config.sessionSendsLaunchEvents = true
    return URLSession(configuration: config, delegate: self, delegateQueue: nil)
}()
```

Note

For more visibility into how the system is scheduling and performing your background tasks, download and install the Background Networking Profile onto your iOS device from the Bug Reporting Profiles and Logs page.

Create and schedule the download task

You create download tasks from the session with either the `downloadTask(with:)` method that takes a URL, or the `downloadTask(with:)` method that takes a `URLRequest` instance. You set properties on this method to help the system optimize its behavior.

1. As shown in the following example, create a download task with `downloadTask(with:)`.

2. Optionally, set the `earliestBeginDate` property to schedule the download to begin at a particular point in the future. The download isn't guaranteed to begin at precisely this time, but it won't start sooner.

3. To help the system schedule network activity efficiently, set the properties `countOfBytesClientExpectsToSend` and `countOfBytesClientExpectsToReceive`. These values are best-guess upper bounds on the expected byte count, and should account for headers as well as body data.

4. To start the task, call `resume()`.

In the following example, the task is set to begin at least one hour in the future and is configured to send around 200 bytes of data and receive around 500 KB.

Creating a download task from a URL session

```
let backgroundTask = urlSession.downloadTask(with: url)
backgroundTask.earliestBeginDate = Date().addingTimeInterval(60 * 60)
backgroundTask.countOfBytesClientExpectsToSend = 200
backgroundTask.countOfBytesClientExpectsToReceive = 500 * 1024
backgroundTask.resume()
```

Handle app suspension

Different app states affect how your app interacts with the background download. In iOS, your app could be in the foreground, suspended, or even terminated by the system. See Managing your app's life cycle for more information about these states.

If your app is in the background, the system may suspend your app while the download is performed in another process. In this case, when the download finishes, the system resumes the app and calls the `UIApplicationDelegate` method `application(_:handleEventsForBackgroundURLSession:completionHandler:)`. This method receives the session identifier you created in `Creating a background URL session` as its second parameter.

This delegate method also receives a completion handler as its final parameter. Immediately store this handler wherever it makes sense for your app, perhaps as a property of your app delegate, or of your class that implements `URLSessionDownloadDelegate`. In the following example, this completion handler is stored in an app delegate property called `backgroundCompletionHandler`.

Storing the background download completion handler sent to the application delegate

```
func application(_ application: UIApplication,
                 handleEventsForBackgroundURLSession identifier: String,
                 completionHandler: @escaping () -> Void) {
        backgroundCompletionHandler = completionHandler
}
```

When all events have been delivered, the system calls the `urlSessionDidFinishEvents(forBackgroundURLSession:)` method of `URLSessionDelegate`. At this point, fetch the `backgroundCompletionHandler` stored by the app delegate in the previous example and execute it. The following example shows this process.

Note that because `urlSessionDidFinishEvents(forBackgroundURLSession:)` may be called on a secondary queue, it needs to explicitly execute the handler (which was received from a UIKit method) on the main queue.

Executing the background URL session completion handler on the main queue

```
func urlSessionDidFinishEvents(forBackgroundURLSession session: URLSession) {
    DispatchQueue.main.async {
        guard let appDelegate = UIApplication.shared.delegate as? AppDelegate,
            let backgroundCompletionHandler =
            appDelegate.backgroundCompletionHandler else {
                return
        }
        backgroundCompletionHandler()
    }
}
```

Access the file, or move it to a permanent location

Once your resumed app calls the completion handler, the download task finishes its work and calls the delegate's `urlSession(_:downloadTask:didFinishDownloadingTo:)` method. At this point, the file is fully downloaded, and will be available until your delegate method returns. If you only need to read it once, you can access the file immediately in its temporary location. If you want to preserve the file, move it to a permanent location like the `Documents` directory, as described in Downloading files from websites.

Recreate the session if the app was terminated

If the system terminated the app while it was suspended, the system relaunches the app in the background. As part of your launch time setup, recreate the background session (see `Creating a background URL session`), using the same session identifier as before, to allow the system to reassociate the background download task with your session. You do this so your background session is ready to go whether the app was launched by the user or by the system. Once the app relaunches, the series of events is the same as if the app had been suspended and resumed, as discussed earlier in Handle app suspension.

> **Note**
>
> In cases where the transfer is initiated while the app is in the background, the session configuration's `isDiscretionary` property is treated as being `true`.

Comply with background transfer limitations

With background sessions, the actual transfer is performed by a process that is separate from your app's process. Because restarting your app's process is fairly expensive, a few features are unavailable, resulting in the following limitations:

- The session *must* provide a delegate for event delivery. (For uploads and downloads, the delegates behave the same as for in-process transfers.)
- Only HTTP and HTTPS protocols are supported (no custom protocols).
- Redirects are always followed. As a result, even if you have implemented `urlSession(_:task:willPerformHTTPRedirection:newRequest:completionHandler:)`, it is *not* called.
- Only upload tasks from a file are supported (uploads from data instances or a stream fail after the app exits).

Use background sessions efficiently

When the system resumes or relaunches your app, it uses a rate limiter to prevent abuse of background downloads. When your app starts a new download task while in the background, the task doesn't begin until the delay expires. The delay increases each time the system resumes or relaunches your app.

As a result, if your app starts a single background download, gets resumed when the download completes, and then starts a new download, it will greatly increase the delay. Instead, use a small number of background sessions — ideally just one — and use these sessions to start many download tasks at once. This allows the system to perform multiple downloads at once, and resume your app when they have completed.

Keep in mind, though, that each task has its own overhead. If you find you need to launch thousands of download tasks, change your design to perform fewer, larger transfers.

> Note
>
> The delay is reset to 0 whenever the user brings your app to the foreground. It also resets if the delay period elapses without the system resuming or relaunching your app.

See Also

Downloading

Downloading files from websites

Download files directly to the filesystem.

Pausing and resuming downloads

Allow the user to resume a download without starting over.