Article

Pausing and resuming uploads

Pause and resume an upload without starting over, even when the connection is interrupted.

Overview

Your app or a person using it might need to cancel an in-progress upload and resume it later. By supporting resumable uploads, you save both the user’s time and network bandwidth.

New in iOS 17 and aligned OS versions, `URLSession` supports resumable uploads according to the latest resumable upload protocol draft. This protocol is currently being developed in the HTTP Working Group at the IETF. To take advantage of resumable uploads, the server you’re connecting to must also support this protocol.

You can use this technique to manually pause and resume an upload, or even resume an upload that failed due to a temporary loss of connectivity. In both cases, `URLSession` needs information on how to resume the upload, which is stored in a `resumeData` object.

Pause an upload

You can effectively pause a `URLSessionUploadTask` by calling `cancel(byProducingResumeData:)`. This method cancels the task and passes a `resumeData` parameter to its completion handler. If `resumeData` is not `nil`, you can use this token later to resume the upload. The listing below shows how to cancel an upload task and store `resumeData`, if it exists, in a property:

```
uploadTask.cancel { resumeData in
    guard let resumeData else {
      // The upload can't be resumed; remove the upload from the UI if necessary.
      return
    }
    self.resumeData = resumeData
}
```

Important

You can’t resume all uploads. The server must support the latest resumable upload protocol draft from the HTTP Working Group at the IETF. Also, uploads that use a background configuration handle resumption automatically, so manual resuming is only needed for non-background uploads.

Recover a failed upload

If there’s only a momentary network interruption, but the server is still reachable, `URLSession` automatically tries to resume the upload for you. No extra code is needed.

For broader losses of connectivity, you can resume a failed upload by checking for resume data in the task’s error.

When the upload fails, the session calls your `urlSession(_:task:didCompleteWithError:)` delegate method. If `error` is not `nil`, look in its `userInfo` dictionary for the key `NSURLSessionUploadTaskResumeData`. If the key exists, save the value associated with it to use later when you try to resume the upload. If the key doesn’t exist, you can’t resume the upload.

You can conveniently access resume data using the `uploadTaskResumeData` property of `URLError`.

You can also catch the error from asynchronous `URLSession` upload methods such as `upload(for:fromFile:)`. The listing below shows an implementation of this error handling that checks the error for resume data:

```
do {
    let (data, response) = try await session.upload(for: request, fromFile: fileURL
} catch let error as URLError {
    guard let resumeData = error.uploadTaskResumeData else {
        // The upload can't be resumed.
        return
    }
    self.resumeData = resumeData
}
```

Resume uploading

When it’s appropriate to resume the upload, create a new `URLSessionUploadTask` by using the `uploadTask(withResumeData:)` method of `URLSession`, passing in the `resumeData` object you stored earlier. Then, call `resume()` on the task to resume the upload:

```
guard let resumeData = self.resumeData else {
    // Inform the user that the upload can't be resumed.
    return
}

let uploadTask = session.uploadTask(withResumeData: resumeData)
uploadTask.resume()
self.uploadTask = uploadTask
```

See Also

Uploading

{} Building a resumable upload server with SwiftNIO

Support HTTP resumable upload protocol in SwiftNIO by translating resumable uploads to regular uploads.

Uploading data to a website

Post data from your app to servers.

Uploading streams of data

Send a stream of data to a server.