Article

Pausing and resuming downloads

Allow the user to resume a download without starting over.

Overview

Your app or the user may need to cancel an in-progress download and resume it later. By supporting resumable downloads, you save both the user's time and network bandwidth.

You can also use this technique to resume a download that fails due to a temporary loss of connectivity.

Store the resume data object when you cancel the download

You cancel a `URLSessionDownloadTask` by calling `cancel(byProducingResumeData:)`. This method takes a completion handler which is called once the cancellation is complete. The completion handler receives a `resumeData` parameter. If it it not `nil`, this is the token you use later to resume the download. The following example shows how to cancel a download task and store `resumeData`, if it exists, in a property.

Storing the resume data when canceling a download

```
downloadTask.cancel { resumeDataOrNil in
    guard let resumeData = resumeDataOrNil else {
      // download can't be resumed; remove from UI if necessary
      return
    }
    self.resumeData = resumeData
}
```

Important

Not all downloads can be resumed. See the discussion in `cancel(byProducingResumeData:)` for a list of conditions that must be met for a download to be resumable. Also, downloads that use a background configuration will handle resumption automatically, so manual resuming is only needed for non-background downloads.

Store the resume data object when a download fails

You can also resume a download that has failed due to a temporary loss of connectivity, as when the user walks out of WiFi range.

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

The following example shows an implementation of `urlSession(_:task:didCompleteWithError:)` that retrieves and saves the `resumeData` object, if any, from the error.

```
func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError er
    guard let error = error else {
        // Handle success case.
        return
    }
    let userInfo = (error as NSError).userInfo
    if let resumeData = userInfo[NSURLSessionDownloadTaskResumeData] as? Data {
        self.resumeData = resumeData
    }
    // Perform any other error handling.
}
```

Use the stored resume data object to resume downloading

When it's appropriate to resume the download, create a new `URLSessionDownloadTask` by using the `downloadTask(withResumeData:)` or `downloadTask(withResumeData:completionHandler:)` method of `URLSession`, passing in the `resumeData` object you stored earlier. Then call `resume()` on the task to resume the download.

Creating and starting a download task from resume data

```
guard let resumeData = resumeData else {
    // inform the user the download can't be resumed
    return
}
let downloadTask = urlSession.downloadTask(withResumeData: resumeData)
downloadTask.resume()
self.downloadTask = downloadTask
```

If the download resumes successfully, the task calls your delegate's `urlSession(_:downloadTask:didResumeAtOffset:expectedTotalBytes:)` method. You can use the offset and byte count parameters to inform the user that the download has resumed and has preserved its earlier progress.

See Also

Downloading

- Downloading files from websites
  Download files directly to the filesystem.
- Downloading files in the background
  Create tasks that download files while your app is inactive.