Article

Uploading data to a website

Post data from your app to servers.

Overview

Many apps work with servers that accept uploads of files like images or documents, or use web service API endpoints that accept structured data like JSON. To upload data from your app, you use a `URLSession` instance to create a `URLSessionUploadTask` instance. The upload task uses a `URLRequest` instance that details how the upload is to be performed.

Prepare your data for upload

The data to upload can be the contents of a file, a stream, or data, as is the case in the following example.

Many web service endpoints take JSON-formatted data, which you create by using the `JSONEncoder` class on `Encodable` types like arrays and dictionaries. As shown in the following example, you can declare a structure that conforms to `Codable`, create an instance of this type, and use `JSONEncoder` to encode the instance to JSON data for upload.

Preparing JSON data for upload

```
struct Order: Codable {
    let customerId: String
    let items: [String]
}

// ...

let order = Order(customerId: "12345",
                  items: ["Cheese pizza", "Diet soda"])
```

```
guard let uploadData = try? JSONEncoder().encode(order) else {
    return
}
```

There are many other ways to create a data instance, such as encoding an image as JPEG or PNG data, or converting a string to data by using an encoding like UTF-8.

Configure an upload request

An upload task requires a `URLRequest` instance. As shown in the following example, set the `httpMethod` property of the request to "``POST``" or "PUT", depending on what the server supports and expects. Use the `setValue(_:forHTTPHeaderField:)` method to set the values of any HTTP headers that you want to provide, except the `Content-Length` header. The session figures out content length automatically from the size of your data.

Configuring a URL request

```
let url = URL(string: "https://example.com/post")!
var request = URLRequest(url: url)
request.httpMethod = "POST"
request.setValue("application/json", forHTTPHeaderField: "Content-Type")
```

Create and start an upload task

To begin an upload, call `uploadTask(with:from:completionHandler:)` on a `URLSession` instance to create an uploading `URLSessionTask` instance, passing in the request and the data instances you've previously set up. Because tasks start in a suspended state, you begin the network loading process by calling `resume()` on the task. The following example uses the shared `URLSession` instance, and receives its results in a completion handler. The handler checks for transport and server errors before using any returned data.

Starting an upload task

```
let task = URLSession.shared.uploadTask(with: request, from: uploadData) { data, res
    if let error = error {
        print ("error: \(error)")
        return
    }
    guard let response = response as? HTTPURLResponse,
        (200...299).contains(response.statusCode) else {
        print ("server error")
```

```
        return
    }
    if let mimeType = response.mimeType,
        mimeType == "application/json",
        let data = data,
        let dataString = String(data: data, encoding: .utf8) {
        print ("got data: \(dataString)")
    }
}
task.resume()
```

Alternatively, upload by setting a delegate

As an alternative to the completion handler approach, you can instead set a delegate on a session you configure, and then create the upload task with `uploadTask(with:from:)`. In this scenario, you implement methods from the `URLSessionDelegate` and `URLSessionTaskDelegate` protocols. These methods receive the server response and any data or transport errors.

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 streams of data

Send a stream of data to a server.

Pausing and resuming uploads

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