

URLSession

An object that coordinates a group of related network data transfer tasks.

Declaration

```
class URLSession : NSObject
```

Overview

The `URLSession` class and related classes provide an API for downloading data from and uploading data to endpoints indicated by URLs. The API also enables your app to perform background downloads when your app isn't running or, in iOS, while your app is suspended. A rich set of delegate methods support authentication and allow your app to be notified of events like redirection.

SDKs

iOS 7.0+

macOS 10.9+

tvOS 9.0+

watchOS 2.0+

Framework

Foundation

On This Page

Declaration ☑

Overview ☑

Topics ☑

Relationships ☑

See Also ☑

Important

The `URLSession` API involves many different classes working together in a fairly complex way that may not be obvious if you read the reference documentation by itself. Before using the API, read the overview in the [URL Loading System](#) topic. The articles in the [First Steps](#), [Uploading](#), and [Downloading](#) sections offer examples of performing common tasks with `URLSession`.

Using the `URLSession` API, your app creates one or more sessions, each of which coordinates a group of related data transfer tasks. For example, if you're creating a web browser, your app might create one session per tab or window, or one session for interactive use and another for background downloads. Within each session, your app adds a series of tasks, each of which represents a request for a specific URL (following HTTP redirects, if necessary).

Types of URL Sessions

The tasks within a given URL session share a common session configuration object, which defines connection behavior, like the maximum number of simultaneous connections to make to a single host, whether to allow connections over a cellular network, and so on.

`URLSession` has a singleton shared session (which has no configuration object) for basic requests. It's not as customizable as sessions you create, but it serves as a good starting point if you have very limited requirements. You access this session by calling the shared class method. For other kinds of sessions, you instantiate a `URLSession` with one of three kinds of configurations:

- A *default session* behaves much like the shared session, but allows more configuration, and allows you to obtain data incrementally with a delegate.
- *Ephemeral sessions* are similar to shared sessions, but don't write caches, cookies, or credentials to disk.
- *Background sessions* let you perform uploads and downloads of content in the background while your app isn't running.

See [Creating a Session Configuration Object in the `URLSessionConfiguration` class](#) for details on creating each type of configuration.

Types of URL Session Tasks

Within a session, you create tasks that optionally upload data to a server and then retrieve data from the server either as a file on disk or as one or more `NSData` objects in memory. The `URLSession` API provides three types of tasks:

- *Data tasks* send and receive data using `NSData` objects. Data tasks are intended for short, often interactive requests to a server.
- *Upload tasks* are similar to data tasks, but they also send data (often in the form of a file), and support background uploads while the app isn't running.
- *Download tasks* retrieve data in the form of a file, and support background downloads and uploads while the app isn't running.

Using a Session Delegate

The tasks in a session also share a common delegate that lets you provide and obtain information when various events occur—when authentication fails, when data arrives from the server, when data is ready to be cached, and so on. If you don't need any of the features provided by a delegate, you can use this API without providing one by passing `nil` when you create a session.

Important

The session object keeps a strong reference to the delegate until your app exits or explicitly invalidates the session. If you don't invalidate the session, your app leaks memory until it exits.

Asynchronicity and URL Sessions

Like most networking APIs, the `NSURLSession` API is highly asynchronous. It returns data to your app in one of two ways, depending on the methods you call:

- By calling a completion handler block when a transfer finishes successfully or with an error.
- By calling methods on the session's delegate as data is received and when the transfer is complete.

In addition to delivering this information to delegates, the `NSURLSession` API provides status and progress properties that you can query if you need to make programmatic decisions based on the current state of the task (with the caveat that its state can change at any time).

URL sessions also support canceling, restarting, resuming, and suspending tasks, and provide the ability to resume suspended, canceled, or failed downloads where they left off.

Protocol Support

The `NSURLSession` class natively supports the `data`, `file`, `ftp`, `http`, and `https` URL schemes, with transparent support for proxy servers and SOCKS gateways, as configured in the user's system preferences.

`NSURLSession` supports the HTTP/1.1, SPDY, and HTTP/2 protocols. HTTP/2 support is described by RFC 7540, and requires a server supporting either Application-Layer Protocol Negotiation (ALPN) or Next Protocol Negotiation (NPN).

You can also add support for your own custom networking protocols and URL schemes (for your app's private use) by subclassing `URLProtocol`.

App Transport Security (ATS)

Starting in iOS 9.0 and OS X 10.11, a new security feature called App Transport Security (ATS) is enabled by default for all HTTP connections made with `NSURLSession`. ATS requires that HTTP connections use HTTPS (RFC 2818).

For more information, see `NSAppTransportSecurity` in the Information Property List Key Reference.

NSCopying Behavior

Session and task objects conform to the `NSCopying` protocol as follows:

- When your app copies a session or task object, you get the same object back.
- When your app copies a configuration object, you get a new copy that you can independently modify.

Thread Safety

The URL session API itself is fully thread-safe. You can freely create sessions and tasks in any thread context. When your delegate methods call the provided completion handlers, the work is automatically scheduled on the correct delegate queue.

Warning

The system may call the `urlSessionDidFinishEvents(forBackground URLSession:)` session delegate method on a secondary thread. However, in iOS, your implementation of that method may need to call a completion handler provided to you in your `application(_:handleEventsForBackgroundURLSession:completionHandler:)` app delegate method. You *must* call that completion handler on the main thread.

Topics

Using the Shared Session

```
class var shared: URLSession
```

The shared singleton session object.

Creating a Session

```
init(configuration: URLSessionConfiguration)
```

Creates a session with the specified session configuration.


```
init(configuration: URLSessionConfiguration, delegate: URLSessionDelegate?, delegateQueue: OperationQueue?)
```

Creates a session with the specified session configuration, delegate, and operation queue.


```
class URLSessionConfiguration
```

A configuration object that defines behavior and policies for a URL session.


```
var configuration: URLSessionConfiguration
```

A copy of the configuration object for this session.

Working with a Delegate

```
var delegate: URLSessionDelegate?
```

The delegate assigned when this object was created.

```
protocol URLSessionDelegate
```

A protocol defining methods that URL session instances call on their delegates to handle session-level events, like session life cycle changes.

```
protocol URLSessionTaskDelegate
```

A protocol defining methods that URL session instances call on their delegates to handle task-level events.

```
var delegateQueue: OperationQueue
```

The operation queue provided when this object was created.

Adding Data Tasks to a Session

```
func dataTask(with: URL) -> URLSessionDataTask
```

Creates a task that retrieves the contents of the specified URL.

```
func dataTask(with: URL, completionHandler: (Data?,  
URLResponse?, Error?) -> Void) -> URLSessionDataTask
```

Creates a task that retrieves the contents of the specified URL, then calls a handler upon completion.

```
func dataTask(with: URLRequest) -> URLSessionDataTask
```

Creates a task that retrieves the contents of a URL based on the specified URL request object.

```
func dataTask(with: URLRequest, completionHandler: (Data?,  
URLResponse?, Error?) -> Void) -> URLSessionDataTask
```

Creates a task that retrieves the contents of a URL based on the specified URL request object, and calls a handler upon completion.

```
class URLSessionDataTask
```

A URL session task that returns downloaded data directly to the app in memory.

```
protocol URLSessionDataDelegate
```

A protocol defining methods that URL session instances call on their delegates to handle task-level events specific to data and upload tasks.

Adding Download Tasks to a Session

```
func downloadTask(with: URL) -> URLSessionDownloadTask
```

Creates a download task that retrieves the contents of the specified URL and saves the results to a file.

```
func downloadTask(with: URL, completionHandler: (URL?,  
URLResponse?, Error?) -> Void) -> URLSessionDownloadTask
```

Creates a download task that retrieves the contents of the specified URL, saves the results to a file, and calls a handler upon completion.

```
func downloadTask(with: URLRequest) -> URLSessionDownloadTask
```

Creates a download task that retrieves the contents of a URL based on the specified URL request object and saves the results to a file.

```
func downloadTask(with: URLRequest, completionHandler: (URL?,  
URLResponse?, Error?) -> Void) -> URLSessionDownloadTask
```

Creates a download task that retrieves the contents of a URL based on the specified URL request object, saves the results to a file, and calls a handler upon completion.

```
func downloadTask(withResumeData: Data) -> URLSessionDownload  
Task
```

Creates a download task to resume a previously canceled or failed download.

```
func downloadTask(withResumeData: Data, completionHandler:  
(URL?, URLResponse?, Error?) -> Void) -> URLSessionDownloadTask
```

Creates a download task to resume a previously canceled or failed download and calls a handler upon completion.

```
class URLSessionDownloadTask
```

A URL session task that stores downloaded data to file.

```
protocol URLSessionDownloadDelegate
```

A protocol defining methods that URL session instances call on their delegates to handle task-level events specific to download tasks.

Adding Upload Tasks to a Session

```
func uploadTask(with: URLRequest, from: Data) -> URLSessionUploadTask
```

Creates a task that performs an HTTP request for the specified URL request object and uploads the provided data.

```
func uploadTask(with: URLRequest, from: Data?, completion  
Handler: (Data?, URLResponse?, Error?) -> Void) -> URLSessionUploadTask
```

Creates a task that performs an HTTP request for the specified URL request object, uploads the provided data, and calls a handler upon completion.

```
func uploadTask(with: URLRequest, fromFile: URL) -> URLSessionUploadTask
```

Creates a task that performs an HTTP request for uploading the specified file.

```
func uploadTask(with: URLRequest, fromFile: URL, completion  
Handler: (Data?, URLResponse?, Error?) -> Void) -> URLSessionUploadTask
```

Creates a task that performs an HTTP request for uploading the specified file, then calls a handler upon completion.

```
func uploadTask(withStreamedRequest: URLRequest) -> URLSessionUploadTask
```

Creates a task that performs an HTTP request for uploading data based on the specified URL request.

```
class URLSessionUploadTask
```

A URL session task that uploads data to the network in a request body.

```
protocol URLSessionDataDelegate
```

A protocol defining methods that URL session instances call on their delegates to handle task-level events specific to data and upload tasks.

Adding Stream Tasks to a Session

```
func streamTask(withHostName: String, port: Int) -> URLSessionStreamTask
```

Creates a task that establishes a bidirectional TCP/IP connection to a specified hostname and port.

```
func streamTask(with: NetService) -> URLSessionStreamTask
```

Creates a task that establishes a bidirectional TCP/IP connection using a specified network service.

```
class URLSessionStreamTask
```

A URL session task that is stream-based.

```
protocol URLSessionStreamDelegate
```

A protocol defining methods that URL session instances call on their delegates to handle task-level events specific to stream tasks.

Managing the Session

```
func finishTasksAndInvalidate()
```

Invalidates the session, allowing any outstanding tasks to finish.

```
func flush(completionHandler: () -> Void)
```

Flushes cookies and credentials to disk, clears transient caches, and ensures that future requests occur on a new TCP connection.

```
func getTasksWithCompletionHandler(([URLSessionDataTask],  
[URLSessionUploadTask], [URLSessionDownloadTask]) -> Void)
```

Asynchronously calls a completion callback with all data, upload, and download tasks in a session.

```
func getAllTasks(completionHandler: ([URLSessionTask]) -> Void)
```

Asynchronously calls a completion callback with all tasks in a session

```
func invalidateAndCancel()
```

Cancels all outstanding tasks and then invalidates the session.

```
func reset(completionHandler: () -> Void)
```

Empties all cookies, caches and credential stores, removes disk files, flushes in-progress downloads to disk, and ensures that future requests occur on a new socket.

```
var sessionDescription: String?
```

An app-defined descriptive label for the session.

Handling Errors

📖 [NSURLSession-Specific NSError userInfo Dictionary Keys](#)

Keys used in conjunction with NSError objects returned by the NSURLSession API.

📖 [Background Task Cancellation](#)

Constants that indicate why a background task was canceled.

Relationships

Inherits From


NSObject

Conforms To

`CVarArg`, `Equatable`, `Hashable`

See Also

First Steps

 [Fetching Website Data into Memory](#)

Receive data directly into memory by creating a data task from a URL session.

`class` `URLSessionTask`

A task, like downloading a specific resource, performed in a URL session.