Cookies and Custom Protocols

If your app needs to manage cookies programmatically, such as adding and deleting cookies or determining which cookies should be accepted, read Cookie Storage.

If your app needs to support a URL-based protocol that `NSURL` does not support natively, you can register your own custom protocol class that provides the needed support. To learn more, read Protocol Support.

Cookie Storage

Due to the stateless nature of the HTTP protocol, clients often use cookies to provide persistent storage of data across URL requests. The URL loading system provides interfaces to create and manage cookies, to send cookies as part of an HTTP request, and to receive cookies when interpreting a web server's response.

The `NSHTTPCookie` class encapsulates a cookie, providing accessors for many of the common cookie attributes. This class also provides methods to convert HTTP cookie headers to `NSHTTPCookie` instances and convert an `NSHTTPCookie` instance to headers suitable for use with an `NSURLRequest` object. The URL loading system automatically sends any stored cookies appropriate for an `NSURLRequest` object unless the request specifies not to send cookies. Likewise, cookies returned in an `NSURLResponse` object are accepted in accordance with the current cookie acceptance policy.

The `NSHTTPCookieStorage` class provides the interface for managing the collection of `NSHTTPCookie` objects shared by all apps.

> **iOS Note:** Cookies are not shared between apps in iOS.

`NSHTTPCookieStorage` allows an app to specify a cookie acceptance policy. The cookie acceptance policy controls whether cookies should always be accepted, never be accepted, or be accepted only from the same domain as the main document URL.

> **Note:** Changing the cookie acceptance policy in an app affects the cookie acceptance policy for all other running apps.

When another app changes the cookie storage or the cookie acceptance policy, `NSHTTPCookieStorage` notifies an app by posting the `NSHTTPCookieManagerCookiesChangedNotification` and `NSHTTPCookieStorageAcceptPolicyChangedNotification` notifications.

For more information, see *NSHTTPCookieStorage Class Reference* and *NSHTTPCookie Class Reference*.

Protocol Support

The URL loading system design allows a client app to extend the protocols that are supported for transferring data. The URL loading system natively supports the `http`, `https`, `file`, `ftp`, and `data` protocols.

You can implement a custom protocol by subclassing `NSURLProtocol` and then registering the new class with the URL loading system using the `NSURLProtocol` class method `registerClass:`. When an `NSURLSession`, `NSURLConnection`, or `NSURLDownload` object initiates a connection for an `NSURLRequest` object, the URL loading system consults each of the registered classes in the reverse

order of their registration. The first class that returns `YES` for a `canInitWithRequest:` message is used to handle the request.

If your custom protocol requires additional properties for its requests or responses, you support them by creating categories on the `NSURLRequest`, `NSMutableURLRequest`, and `NSURLResponse` classes that provide accessors for those properties. The `NSURLProtocol` class provides methods for setting and getting property values in those accessors.

The URL loading system is responsible for creating and releasing `NSURLProtocol` instances when connections start and complete. Your app should never create an instance of `NSURLProtocol` directly.

When an `NSURLProtocol` subclass is initialized by the URL loading system, it is provided a client object that conforms to the `NSURLProtocolClient` protocol. The `NSURLProtocol` subclass sends messages from the `NSURLProtocolClient` protocol to the client object to inform the URL loading system of its actions as it creates a response, receives data, redirects to a new URL, requires authentication, and completes the load. If the custom protocol supports authentication, then it must conform to the `NSURLAuthenticationChallengeSender` protocol.

For more information, see *NSURLProtocol Class Reference*.
