Understanding Cache Access

The URL loading system provides a composite on-disk and in-memory cache of responses to requests. This cache allows an application to reduce its dependency on a network connection and increase its performance.

Using the Cache for a Request

An `NSURLRequest` instance specifies how the local cache is used by setting the cache policy to one of the `NSURLRequestCachePolicy` values: `NSURLRequestUseProtocolCachePolicy`, `NSURLRequestReloadIgnoringCacheData`, `NSURLRequestReturnCacheDataElseLoad`, or `NSURLRequestReturnCacheDataDontLoad`.

The default cache policy for an `NSURLRequest` instance is `NSURLRequestUseProtocolCachePolicy`. The `NSURLRequestUseProtocolCachePolicy` behavior is protocol specific and is defined as being the best conforming policy for the protocol.

Setting the cache policy to `NSURLRequestReloadIgnoringCacheData` causes the URL loading system to load the data from the originating source, ignoring the cache completely.

The `NSURLRequestReturnCacheDataElseLoad` cache policy causes the URL loading system to use cached data, ignoring its age or expiration date, and to load the data from the originating source only if there is no cached version.

The `NSURLRequestReturnCacheDataDontLoad` policy allows an application to specify that only data in the cache should be returned. Attempting to create an `NSURLConnection` or `NSURLDownload` instance with this cache policy returns `nil` immediately if the response is not in the local cache. This is similar in function to an “offline” mode and never brings up a network connection.

Note: Currently, only responses to HTTP and HTTPS requests are cached. The FTP and `file` protocols attempt to access the originating source as allowed by the cache policy. Custom `NSURLProtocol` classes can optionally provide caching.

Cache Use Semantics for the HTTP Protocol

The most complicated cache use situation is when a request uses the HTTP protocol and has set the cache policy to `NSURLRequestUseProtocolCachePolicy`.

If an `NSCachedURLResponse` does not exist for the request, then the URL loading system fetches the data from the originating source.

If a cached response exists for the request, the URL loading system checks the response to determine if it specifies that the contents must be revalidated.

If the contents must be revalidated, the URL loading system makes a HEAD request to the originating source to see if the resource has changed. If it has not changed, then the URL loading system returns the cached response. If it has changed, the URL loading system fetches the data from the originating source.

If the cached response doesn’t specify that the contents must be revalidated, the URL loading system examines the maximum age or expiration specified in the cached response. If the cached response is recent enough, then the URL loading system returns the cached response. If the response is stale, the URL loading system makes a HEAD request to the originating source to determine whether the resource has changed. If so, the URL loading system fetches the resource from the originating source. Otherwise, it returned the cached response.

RFC 2616, Section 13 (http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13) specifies the semantics involved in detail.

Controlling Caching Programmatically

By default, the data for a request is cached based on the request's cache policy, as interpreted by the `NSURLProtocol` subclass that handles the request.

If your app needs more precise programmatic control over caching (and if the protocol supports caching), you can implement a delegate method that allows your app to determine on a per-request basis whether a particular response should be cached.

- For `NSURLSession` data and upload tasks, implement the `URLSession:dataTask:willCacheResponse:completionHandler:` method.

  This delegate method is called *only* for data and upload tasks. The caching policy for download tasks is determined by the specified cache policy exclusively.

- For `NSURLConnection`, implement the `connection:willCacheResponse:` method.

For `NSURLSession`, your delegate method calls a completion handler block to tell the session what to cache. For `NSURLConnection`, your delegate method returns the object that the connection should cache.

In either case, the delegate typically provides one the following values:

- The provided response object to allow caching
- A newly created response object to cache a modified response—for example, a response with a storage policy that allows caching to memory but not to disk
- `NULL` to prevent caching

Your delegate method can also insert objects into the `userInfo` dictionary associated with an `NSCachedURLResponse` object, causing those objects to be stored in the cache as part of the response.

Important: If you are using `NSURLSession` and you implement this delegate method, your delegate method *must always* call the provided completion handler. Otherwise, your app leaks memory.

The example in Listing 7-1 prevents the on-disk caching of HTTPS responses. It also adds the current date to the user info dictionary for responses that are cached.

Listing 7-1 Example `connection:withCacheResponse:` implementation

```
-(NSCachedURLResponse *)connection:(NSURLConnection *)connection
                 willCacheResponse:(NSCachedURLResponse *)cachedResponse
{
    NSCachedURLResponse *newCachedResponse = cachedResponse;

    NSDictionary *newUserInfo;
    newUserInfo = [NSDictionary dictionaryWithObject:[NSDate date]
                                              forKey:@"Cached Date"];
    if ([[[[cachedResponse response] URL] scheme] isEqual:@"https"]) {
#if ALLOW_IN_MEMORY_CACHING
        newCachedResponse = [[NSCachedURLResponse alloc]
                                initWithResponse:[cachedResponse response]
```

```
                                    data:[cachedResponse data]
                                    userInfo:newUserInfo

storagePolicy:NSURLCacheStorageAllowedInMemoryOnly];
#else // !ALLOW_IN_MEMORY_CACHING
        newCachedResponse = nil
#endif // ALLOW_IN_MEMORY_CACHING
    } else {
        newCachedResponse = [[NSCachedURLResponse alloc]
                                initWithResponse:[cachedResponse response]
                                    data:[cachedResponse data]
                                    userInfo:newUserInfo
                                    storagePolicy:[cachedResponse storagePolicy]];
    }
    return newCachedResponse;
}
```