Using NSURLDownload

In OS X, NSURLDownload gives an application the ability to download the contents of a URL directly to disk. It provides an interface similar to NSURLConnection, adding an additional method for specifying the destination of the file. NSURLDownload can also decode commonly used encoding schemes such as MacBinary, BinHex, and gzip. Unlike NSURLConnection, data downloaded using NSURLDownload is not stored in the cache system.

If your application is not restricted to using Foundation classes, the WebKit framework includes WebDownload, a subclass of NSURLDownload that provides a user interface for authentication.

iOS Note: The NSURLDownload class is not available in iOS, because downloading directly to the file system is discouraged. Use the NSURLSession or NSURLConnection class instead. See Using NSURLSession and Using NSURLConnection for more information.

Downloading to a Predetermined Destination

One usage pattern for NSURLDownload is downloading a file to a predetermined filename on disk. If the application knows the destination of the download, it can set it explicitly using setDestination:allowOverwrite:. Multiple setDestination:allowOverwrite: messages to an NSURLDownload instance are ignored.

The download starts immediately upon receiving the initWithRequest:delegate: message. It can be canceled any time before the delegate receives a downloadDidFinish: or download:didFailWithError: message by sending the download a cancel message.

The example in Listing 3-1 sets the destination, and thus requires the delegate only implement the download:didFailWithError: and downloadDidFinish: methods.

Listing 3-1 Using NSURLDownload with a predetermined destination file location

```
- (void)startDownloadingURL:sender
{
    // Create the request.
    NSURLRequest *theRequest = [NSURLRequest requestWithURL:[NSURL
URLWithString:@"http://www.apple.com"]

cachePolicy:NSURLRequestUseProtocolCachePolicy
                                          timeoutInterval:60.0];


    // Create the connection with the request and start loading the data.
NSURLDownload  *theDownload = [[NSURLDownload alloc] initWithRequest:theRequest
                                             delegate:self];
    if (theDownload) {
        // Set the destination file.
        [theDownload setDestination:@"/tmp" allowOverwrite:YES];
    } else {
        // inform the user that the download failed.
    }
}
```

```
- (void)download:(NSURLDownload *)download didFailWithError:(NSError *)error
{
    // Dispose of any references to the download object
    // that your app might keep.
    ...

    // Inform the user.
    NSLog(@"Download failed! Error - %@ %@",
          [error localizedDescription],
          [[error userInfo] objectForKey:NSURLErrorFailingURLStringErrorKey]);
}

- (void)downloadDidFinish:(NSURLDownload *)download
{
    // Dispose of any references to the download object
    // that your app might keep.
    ...

    // Do something with the data.
    NSLog(@"%@",@"downloadDidFinish");
}
```

The delegate can implement additional methods to customize the handling of authentication, server redirects, and file decoding.

Downloading a File Using the Suggested Filename

Sometimes the application must derive the destination filename from the downloaded data itself. This requires you to implement the delegate method `download:decideDestinationWithSuggestedFilename:` and call `setDestination:allowOverwrite:` with the suggested filename. The example in Listing 3-2 saves the downloaded file to the desktop using the suggested filename.

Listing 3-2 Using `NSURLDownload` with a filename derived from the download

```
- (void)startDownloadingURL:sender
{
    // Create the request.
    NSURLRequest *theRequest = [NSURLRequest requestWithURL:[NSURL
URLWithString:@"http://www.apple.com/index.html"]

cachePolicy:NSURLRequestUseProtocolCachePolicy
                                                       timeoutInterval:60.0];


    // Create the download with the request and start loading the data.
NSURLDownload  *theDownload = [[NSURLDownload alloc] initWithRequest:theRequest
delegate:self];
```

```
    if (!theDownload) {
        // Inform the user that the download failed.
    }
}

- (void)download:(NSURLDownload *)download decideDestinationWithSuggestedFilename:
(NSString *)filename
{
    NSString *destinationFilename;
    NSString *homeDirectory = NSHomeDirectory();

    destinationFilename = [[homeDirectory stringByAppendingPathComponent:@"Desktop"]
        stringByAppendingPathComponent:filename];
    [download setDestination:destinationFilename allowOverwrite:NO];
}



- (void)download:(NSURLDownload *)download didFailWithError:(NSError *)error
{
    // Dispose of any references to the download object
    // that your app might keep.
    ...

    // Inform the user.
    NSLog(@"Download failed! Error - %@ %@",
          [error localizedDescription],
          [[error userInfo] objectForKey:NSURLErrorFailingURLStringErrorKey]);
}

- (void)downloadDidFinish:(NSURLDownload *)download
{
    // Dispose of any references to the download object
    // that your app might keep.
    ...

    // Do something with the data.
    NSLog(@"%@",@"downloadDidFinish");
}
```

The downloaded file is stored on the user's desktop with the name `index.html`, which was derived from the downloaded content. Passing `NO` to `setDestination:allowOverwrite:` prevents an existing file from being overwritten by the download. Instead, a unique filename is created by inserting a sequential number after the filename—for example, `index-1.html`.

The delegate is informed when a file is created on disk if it implements the `download:didCreateDestination:` method. This method also gives the application the opportunity to determine the finalized filename with which the download is saved.

The example in Listing 3-3 logs the finalized filename.

Listing 3-3 Logging the finalized filename using `download:didCreateDestination:`

```
-(void)download:(NSURLDownload *)download didCreateDestination:(NSString *)path
{
    // path now contains the destination path
    // of the download, taking into account any
    // unique naming caused by -setDestination:allowOverwrite:
    NSLog(@"Final file destination: %@",path);
}
```

This message is sent to the delegate after it has been given an opportunity to respond to the `download:shouldDecodeSourceDataOfMIMEType:` and `download:decideDestinationWithSuggestedFilename:` messages.

Displaying Download Progress

You can determine the progress of a download by implementing the delegate methods `download:didReceiveResponse:` and `download:didReceiveDataOfLength:`.

The `download:didReceiveResponse:` method provides the delegate an opportunity to determine the expected content length from the `NSURLResponse`. The delegate should reset the progress each time it receives this message.

The example implementation in Listing 3-4 demonstrates using these methods to provide progress feedback to the user.

Listing 3-4 Displaying the download progress

```
- (void)setDownloadResponse:(NSURLResponse *)aDownloadResponse
{
    // downloadResponse is an instance variable defined elsewhere.
    downloadResponse = aDownloadResponse;
}

- (void)download:(NSURLDownload *)download didReceiveResponse:(NSURLResponse
*)response
{
    // Reset the progress, this might be called multiple times.
    // bytesReceived is an instance variable defined elsewhere.
    bytesReceived = 0;

    // Store the response to use later.
    [self setDownloadResponse:response];
}

- (void)download:(NSURLDownload *)download didReceiveDataOfLength:(unsigned)length
{
    long long expectedLength = [[self downloadResponse] expectedContentLength];

    bytesReceived = bytesReceived + length;

    if (expectedLength != NSURLResponseUnknownLength) {
        // If the expected content length is
        // available, display percent complete.
        float percentComplete = (bytesReceived/(float)expectedLength)*100.0;
        NSLog(@"Percent complete - %f",percentComplete);
    } else {
        // If the expected content length is
        // unknown, just log the progress.
        NSLog(@"Bytes received - %d",bytesReceived);
    }
}
```

The delegate receives a `download:didReceiveResponse:` message before it begins receiving `download:didReceiveDataOfLength:` messages.

Resuming Downloads

In some cases, you can resume a download that was canceled or that failed while in progress. To do so, first make sure your original download doesn't delete its data upon failure by passing `NO` to the download's `setDeletesFileUponFailure:` method. If the original download fails, you can obtain its data with the `resumeData` method. You can then initialize a new download with the `initWithResumeData:delegate:path:` method. When the download resumes, the download's delegate receives the `download:willResumeWithResponse:fromByte:` message.

You can resume a download only if both the protocol of the connection and the MIME type of the file being downloaded support resuming. You can determine whether your file's MIME type is supported with the `canResumeDownloadDecodedWithEncodingMIMEType:` method.

Decoding Encoded Files

`NSURLDownload` provides support for decoding the MacBinary, BinHex, and gzip file formats. If `NSURLDownload` determines that a file is encoded in a supported format, it attempts to send the delegate a `download:shouldDecodeSourceDataOfMIMEType:` message. If the delegate implements this method, it should examine the passed MIME type and return `YES` if the file should be decoded.

The example in Listing 3-5 compares the MIME type of the file and allows decoding of MacBinary and BinHex encoded content.

Listing 3-5 Example implementation of `download:shouldDecodeSourceDataOfMIMEType:` method

```
- (BOOL)download:(NSURLDownload *)download
     shouldDecodeSourceDataOfMIMEType:(NSString *)encodingType
{
    BOOL shouldDecode = NO;

    if ([encodingType isEqual:@"application/macbinary"]) {
        shouldDecode = YES;
    } else if ([encodingType isEqual:@"application/binhex"]) {
        shouldDecode = YES;
    } else if ([encodingType isEqual:@"application/x-gzip"]) {
        shouldDecode = NO;
    }
    return shouldDecode;
}
```