Customized Loading in WKWebView

Session 220

Brady Eidson, Safari and WebKit Engineer
Alex Christensen, Safari and WebKit Engineer
Introducing Touch Bar.

Everyone Can Code
We’re giving everyone the power to learn, write, and teach code.

Apple Pay
An easier way to pay within apps and websites.
Everyone Can Code
We’re giving everyone the power to learn, write, and teach code.
> return "Hello WWDC 2017"
"Hello WWDC 2017"
> return "Whoa, I know Javascript"
"Whoa, I know Javascript"
Your Native Code
Your Native Code
Your Native Code
Developer’s Requests

- Manage cookies
- Filter content
- Custom resources
Manage cookies

Filter unwanted content
Manage cookies
Filter unwanted content
Provide custom resources
Manage cookies
Filter unwanted content
Provide custom resources
SampleCart

username

password

Log In

I forgot my password

I am new customer, sign me up!

SampleChat is a registered trademark of Sample LLC, a subsidiary of Example Corp.

By logging in to this example site you agree to our Terms of Service and our Privacy Policy. Our Privacy Policy is way longer than humanly possible to read. However, our Terms of Service are human-readable.

In short, our terms of service state that you agree to the following:

1. You agree to adopt WKWebView in your application.
2. You agree that this service is available as-is, and there is no implied contract of support.
3. You agree that this service is to be used as an example to illustrate a point.
Do not Eat!!!
Manage Cookies
WKHTTPCookieStore
Manage Cookies
WKHTTPCookieStore

Add and remove individual cookies
Manage Cookies
WKHTTPCookieStore

Add and remove individual cookies

Access all cookies visible to a WKWebView
Manage Cookies
WKHTTPCookieStore

Add and remove individual cookies

Access all cookies visible to a WKWebView

Including HTTP-only cookies
Manage Cookies
WKHTTPCookieStore

Add and remove individual cookies

Access all cookies visible to a WKWebView

Including HTTP-only cookies

Observe the cookie store for changes
Get to a WKWebView's cookie store through its `websiteDataStore`

```swift
open class WKWebsiteDataStore : NSObject, NSCoding {
    open var httpCookieStore: WKHTTPCookieStore { get }
}

let cookieStore = webView.configuration.websiteDataStore.httpCookieStore;
```
Add a cookie

```swift
let cookie = HTTPCookie(properties: [
    HTTPCookiePropertyKey.domain: "canineschool.org",
    HTTPCookiePropertyKey.path: "/",
    HTTPCookiePropertyKey.secure: true,
    HTTPCookiePropertyKey.name: "LoginSessionID",
    HTTPCookiePropertyKey.value: "5bd9d8cabc46041579a311230539b8d1"
])

cookieStore.setCookie(cookie!) {
    webView.load(loggedInURLRequest)
}
```
Add a cookie

```swift
let cookie = HTTPCookie(properties: [
    HTTPCookiePropertyKey.domain: "canineschool.org",
    HTTPCookiePropertyKey.path: "/",
    HTTPCookiePropertyKey.secure: true,
    HTTPCookiePropertyKey.name: "LoginSessionID",
    HTTPCookiePropertyKey.value: "5bd9d8cabc46041579a311230539b8d1"
])

cookieStore.setCookie(cookie!) {
    webView.load(loggedInURLRequest)
}
```
Add a cookie

```swift
let cookie = HTTPCookie(properties: [
    HTTPCookiePropertyKey.domain: "canineschool.org",
    HTTPCookiePropertyKey.path: "/",
    HTTPCookiePropertyKey.secure: true,
    HTTPCookiePropertyKey.name: "LoginSessionID",
    HTTPCookiePropertyKey.value: "5bd9d8cabc46041579a311230539b8d1"
])

cookieStore.setCookie(cookie!) {
    webView.load(loggedInURLRequest)
}
```
Manage Cookies
WKHTTPCookieStore

Retrieve the set of all cookies in a WKWebsiteDataStore

cookieStore.getAllCookies() { (cookies) in
    for cookie in cookies {
        // Find the login cookie
    }
}
Retrieve the set of all cookies in a WKWebsiteDataStore

```swift
cookieStore.getAllCookies() { (cookies) in
    for cookie in cookies {
        // Find the login cookie
    }
}
```
Delete a cookie

```swift
cookieStore.delete(cookie!) {
    webView.load(loggedOutURLRequest)
}
```
Delete a cookie

cookieStore.delete(cookie!) {
    webView.load(loggedOutURLRequest)
}

Manage cookies

Filter unwanted content

Provide custom resources
Manage cookies
Filter unwanted content
Provide custom resources
Filter Unwanted Content
WKContentRuleList
Filter Unwanted Content
WKContentRuleList

Same syntax as Content Blocker extensions for Safari*

Filter Unwanted Content
WKContentRuleList

Same syntax as Content Blocker extensions for Safari*
• Block loads

Filter Unwanted Content
WKContentRuleList

Same syntax as Content Blocker extensions for Safari*

• Block loads
• Make content invisible

Filter Unwanted Content
WKContentRuleList

Same syntax as Content Blocker extensions for Safari*
• Block loads
• Make content invisible
• Make insecure loads secure

Filter Unwanted Content

WKContentRuleList

Same syntax as Content Blocker extensions for Safari*

• Block loads
• Make content invisible
• Make insecure loads secure

WebKit compiles your rules into efficient bytecode

You supply rules in JSON

```json
[{
    "trigger": {
        "url-filter": ".*"
    },
    "action": {
        "type": "make-https"
    }
}]
```
Compile your JSON using WKContentRuleListStore

```swift
let jsonString = loadJSONFromBundle()

WKContentRuleListStore.default().compileContentRuleList(
    forIdentifier: "ContentBlockingRules",
    encodedContentRuleList: jsonString) { (contentRuleList, error) in
    if let error = error {
        return
    }

    createWebViewWithContentRuleList(ruleList!)
}
```
Filter Unwanted Content
WKContentRuleList

Compile your JSON using WKContentRuleListStore

```swift
let jsonString = loadJSONFromBundle()

WKContentRuleListStore.default().compileContentRuleList(
    forIdentifier: "ContentBlockingRules",
    encodedContentRuleList: jsonString) { (contentRuleList, error) in
        if let error = error {
            return
        }

        createWebViewWithContentRuleList(ruleList!)
    }
```
Filter Unwanted Content
WKContentViewRuleList

Compile your JSON using WKContentViewRuleListStore

```swift
let jsonString = loadJSONFromBundle()

WKContentViewRuleListStore.default().compileContentRuleList(
  forIdentifier: "ContentBlockingRules",
  encodedContentRuleList: jsonString) { (contentRuleList, error) in
    if let error = error {
      return
    }

    createWebViewWithContentRuleList(ruleList!)
  }
```
Filter Unwanted Content

WKContentRuleList

Access previously compiled WKContentRuleList

```swift
WKContentRuleListStore.default().lookUpContentRuleList(forIdentifier: "ContentBlockingRules")
{(contentRuleList, error) in
    // Use previously compiled content rule list
}
```
Filter Unwanted Content
WKContentRuleList

Add compiled WKContentRuleList to your WKWebView’s configuration

```swift
let configuration = WKWebViewConfiguration()
configuration.userContentController.add(contentRuleList)
```
Demo
Managing cookies and filtering content

Alex Christensen
Managing Cookies and Filtering Content Demo
Managing Cookies and Filtering Content Demo

Set a cookie before making a request
Managing Cookies and Filtering Content Demo

Set a cookie before making a request
Filter unwanted insecure resources
Manage cookies
Filter unwanted content
Provide custom resources
"Easier than covfefe," remarks newbie.

THE ARENA — In a stunning display of battle hardened prowess, Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Provide Custom Resources
WKURLConnectionHandler
Provide Custom Resources
WKURLSchemeHandler

Allows your app to handle resource loads for a URL scheme
Provide Custom Resources

WKURLSchemeHandler

Allows your app to handle resource loads for a URL scheme

Only custom URL schemes that WebKit doesn’t handle itself

```swift
let customScheme = "local"
```
Provide Custom Resources
WKURLSchemeHandler

Allows your app to handle resource loads for a URL scheme

Only custom URL schemes that WebKit doesn’t handle itself

You should future-proof your custom scheme

```swift
let customScheme = "apple-local"
```
Provide Custom Resources
WKURLSchemeHandler

Simple protocol you implement

```swift
class MyCustomSchemeHandler : NSObject, WKURLSchemeHandler {
    func webView(_ webView: WKWebView, start urlSchemeTask: WKURLSchemeTask) {
    }

    func webView(_ webView: WKWebView, stop urlSchemeTask: WKURLSchemeTask) {
    }
}
```
Provide Custom Resources

WKURLSchemeHandler

Set scheme handlers on the WKWebViewConfiguration

```swift
let configuration = WKWebViewConfiguration()
configuration.setURLSchemeHandler(MyCustomSchemeHandler(), forURLScheme: "apple-local")
```
Load something in your view that uses your scheme

```swift
let configuration = WKWebViewConfiguration()
configuration.setURLSchemeHandler(MyCustomSchemeHandler(), forURLScheme: "apple-local")

let webView = WKWebView(frame: getFrame(), configuration: configuration)
webView.load(URLRequest(url: URL(string: "http://example.com/demoContent")))
```
Provide Custom Resources
WKURLSchemeHandler

Load something in your view that uses your scheme

```swift
let configuration = WKWebViewConfiguration()
configuration.setURLSchemeHandler(MyCustomSchemeHandler(), forURLScheme: "apple-local")

let webView = WKWebView(frame: getFrame(), configuration: configuration)
webView.load(URLRequest(url: URL(string: "apple-local:top")))
```
Let's set up a custom URL scheme handler and load content using it.

```swift
let configuration = WKWebViewConfiguration()
configuration.setURLSchemeHandler(MyCustomSchemeHandler(), forURLScheme: "apple-local")

let webView = WKWebView(frame: getFrame(), configuration: configuration)
webView.load(URLRequest(url: URL(string: "apple-local:top")!))
```
The WKURLSchemeTask sent to your handler represents a resource load.

```swift
protocol WKURLSchemeTask : NSObjectProtocol {
    var request: URLRequest

    func didReceive(_ response: URLResponse)
    func didReceive(_ data: Data)
    func didFinish()
    func didFailWithError(_ error: Error)
}
```
The WKURLSchemeTask sent to your handler represents a resource load.

```swift
protocol WKURLSchemeTask : NSObjectProtocol {
    var request: URLRequest
    func didReceive(_ response: URLResponse)
    func didReceive(_ data: Data)
    func didFinish()
    func didFailWithError(_ error: Error)
}
```
The WKURLSchemeTask sent to your handler represents a resource load.

```swift
protocol WKURLSchemeTask : NSObjectProtocol {
    var request: URLRequest

    func didReceive(_ response: URLResponse)
    func didReceive(_ data: Data)
    func didFinish()
    func didFailWithError(_ error: Error)
}
```
You first prepare a response

```swift
func webView(_ webView: WKWebView, start urlSchemeTask: WKURLSchemeTask) {
    let resourceData = createHTMLResourceData()

    let response = URLResponse(
        url: urlSchemeTask.request.url!,
        mimeType: "text/html",
        expectedContentLength: resourceData.count,
        textEncodingName: nil)
}
```
func webView(_ webView: WKWebView, start urlSchemeTask: WKURLSchemeTask) {
    let resourceData = createHTMLResourceData()

    let response = URLResponse(
        url: urlSchemeTask.request.url!,
        mimeType: "text/html",
        expectedContentLength: resourceData.count,
        textEncodingName: nil)
}

The response must include the MIME type

Provide Custom Resources
WKURLSchemeHandler
Provide Custom Resources
WKURLSchemeHandler

Deliver the response

```swift
func webView(_ webView: WKWebView, start urlSchemeTask: WKURLSchemeTask) {
    let resourceData = createHTMLResourceData()

    let response = ...

    urlSchemeTask.didReceive(response)
    urlSchemeTask.didReceive(resourceData)
    urlSchemeTask didFinish()
}
```
Provide Custom Resources
WKURLSchemeHandler

Deliver the resource data

```swift
func webView(_ webView: WKWebView, start urlSchemeTask: WKURLSchemeTask) {
    let resourceData = createHTMLResourceData()

    let response = ...

    urlSchemeTask.didReceive(response)
    urlSchemeTask.didReceive(resourceData)
    urlSchemeTask didFinish()
}
```
func webView(_ webView: WKWebView, start urlSchemeTask: WKURLSchemeTask) {
    let resourceData = createHTMLResourceData()

    let response = ...

    urlSchemeTask.didReceive(response)
    urlSchemeTask.didReceive(resourceData)
    urlSchemeTask didFinish()
Demo
Providing custom resources

Alex Christensen
Providing Custom Resources Demo
Providing Custom Resources Demo

Chose a future-proof URL scheme
Providing Custom Resources Demo

Chose a future-proof URL scheme

Delivered image data asynchronously
WKURLSchemeHandler
More Information

# Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s New in Safari View Controller</td>
<td>Executive Ballroom</td>
<td>Thursday 10:00AM</td>
</tr>
<tr>
<td>Introducing Safari View Controller</td>
<td></td>
<td>WWDC 2015</td>
</tr>
<tr>
<td>Safari Extensibility: Content Blocking and Shared Links</td>
<td></td>
<td>WWDC 2015</td>
</tr>
<tr>
<td>Introducing the Modern WebKit API</td>
<td></td>
<td>WWDC 2014</td>
</tr>
<tr>
<td>Labs</td>
<td>Technology Lab</td>
<td>Time</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>----------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Safari, WebKit, and Password AutoFill Lab</td>
<td>Technology Lab B</td>
<td>Wed 5:10PM–6:10PM</td>
</tr>
<tr>
<td>Safari, WebKit, and Password AutoFill Lab</td>
<td>Technology Lab K</td>
<td>Thur 11:00AM–1:00PM</td>
</tr>
</tbody>
</table>