WKWebView

Overview

WKWebView is an open source project to support the `WKWebView` API with Titanium SDK.

Requirements

- Titanium SDK
- iOS 9+
- Xcode 7+

Titanium SDK 8.0.0 release

With Titanium SDK 8.0.0, we now use WKWebView to implement Ti.UI.WebView (as Apple has deprecated UIWebView).

Adding the following to your HTML will make a `WKWebView` scale its content the same was as the old UIWebView.

WKWebView scaling

```
<html>
 <head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
 </head>
</html>
```

WKWebView also has few restriction specially with local file accessing. For supporting custom-fonts with WKWebView a little modification is required in the HTML files:

```
<style>
  @font-face
    {
      font-family: 'Lato-Regular';
      src: url('fonts/Lato-Regular.ttf');
    }
</style>
```

Installation

Download the stable release, unpack the module, and place it inside the `modules/iphone` directory of your project. Edit the iOS and modules section of your `tiapp.xml` file to include this module:

tiapp.xml update

```
<ios>
  <min-ios-ver>9.0</min-ios-ver>
</ios>
<modules>
  <module>ti.wkwebview</module>
</modules>
```

Features

API's

Name	Example
`WebView`	`WK.createWebView(args)`
`ProcessPool`	`WK.createProcessPool()`
`Configuration`	`WK.createConfiguration()`

WebView

Properties

Name	Type
`disableBounce`	`Boolean`
`scalesPageToFit`	`Boolean`
`disableZoom`	`Boolean`
`allowsBackForwardNavigationGestures`	`Boolean`
`allowsLinkPreview`	`Boolean`
`scrollsToTop`	`Boolean`
`disableContextMenu`	`Boolean`
`userAgent`	`String`
`url`	`String`
`data`	`Ti.Blob`, `Ti.File`
`html`	`String`
`title`	`String`
`progress`	`Number`
`backForwardList`	`Object`
`ignoreSslError`	`Boolean`

basicAuthentication	Object • username (String) • password (String) • persistence (CREDENTIAL_PERSISTENCE_*)
cachePolicy	CACHE_POLICY_*
timeout	Number
selectionGranularity	SELECTION_GRANULARITY_*
allowedURLSchemes	Array<String>
touchEnabled	Boolean
willHandleTouches	Boolean
requestHeaders	Object<String, Any>
zoomLevel	Number
keyboardDisplayRequiresUserAction	Boolean

Methods

Name	Parameter	Return
stopLoading	-	Void
setHtml	html, options (Object - { baseURL, mimeType }, optional)	Void
reload	-	Void
repaint	-	Void
goBack	-	Void
goForward	-	Void
canGoBack	-	Boolean
canGoForward	-	Boolean
isLoading	-	Boolean
evalJS **	Code (String), Callback (Function)	
startListeningToProperties	Array	Void
stopListeningToProperties	Array	Void
fireEvent	Name (String), Payload (Object)	Void
addEventListener	Name (String), Callback (Function)	Void
removeEventListener	Name (String), Callback (Function)	Void
takeSnapshot *	Callback (Function)	Void
addUserScript	Object: source (String), injectionTime (INJECTION_TIME_*), mainFrameOnly (Boolean)	Void
removeAllUserScripts	-	Void
addScriptMessageHandler	Name (String)	Void
removeScriptMessageHandler	Name (String)	Void

* Available on iOS 11.0+

** Since iOS 12.0, using evalJS in sync is not possible anymore, because the internal callback is now called on the main thread, causing a deadlock in the run-loop that is used to return the value synchronously. To fix this, use the asynchronous solution:

```
webView.evalJS('document.title', function (e) {
 alert(e.result);
});
```

Events

Name	Properties
`message`	name, body, url, isMainFrame
`progress`	value, url
`beforeload`	url, navigationType, title
`load`	url, title
`redirect`	url, title
`error`	success, url, error, code
`sslerror`	url
`handleurl`	url
`blacklisturl`	url

Constants

Name	Property
`CREDENTIAL_PERSISTENCE_NONE`	basicAuthentication.persistence
`CREDENTIAL_PERSISTENCE_FOR_SESSION`	basicAuthentication.persistence
`CREDENTIAL_PERSISTENCE_PERMANENT`	basicAuthentication.persistence
`CREDENTIAL_PERSISTENCE_SYNCHRONIZABLE`	basicAuthentication.persistence
`AUDIOVISUAL_MEDIA_TYPE_NONE`	mediaTypesRequiringUserActionForPlayback
`AUDIOVISUAL_MEDIA_TYPE_AUDIO`	mediaTypesRequiringUserActionForPlayback
`AUDIOVISUAL_MEDIA_TYPE_VIDEO`	mediaTypesRequiringUserActionForPlayback
`AUDIOVISUAL_MEDIA_TYPE_ALL`	mediaTypesRequiringUserActionForPlayback
`CACHE_POLICY_USE_PROTOCOL_CACHE_POLICY`	cachePolicy
`CACHE_POLICY_RELOAD_IGNORING_LOCAL_CACHE_DATA`	cachePolicy
`CACHE_POLICY_RETURN_CACHE_DATA_ELSE_LOAD`	cachePolicy
`CACHE_POLICY_RETURN_CACHE_DATA_DONT_LOAD`	cachePolicy
`SELECTION_GRANULARITY_AUTOMATIC`	selectionGranularity
`SELECTION_GRANULARITY_CHARACTER`	selectionGranularity
`ACTION_POLICY_CANCEL`	handleurl (Event)
`ACTION_POLICY_ALLOW`	handleurl (Event)
`INJECTION_TIME_DOCUMENT_START`	addUserScript.injectionTime
`INJECTION_TIME_DOCUMENT_END`	addUserScript.injectionTime

WebView <-> App Communication

You can send data from the Web View to your native app by posting messages like this:

Send data from the Web View

```
window.webkit.messageHandlers.Ti.postMessage({ message: 'Titanium rocks!' },'*');
```

Please note that you should use the `Ti` message handler to ensure the `message` event is triggered. This also ensures that your app does not receive unwanted messages by remote pages.

After sending the message from your HTML file, it will trigger the `message` event with the following event keys:

- `url`: The url of the triggered message
- `body`: The message body. In this case: `{message: 'Titanium rocks'}`
- `name`: The name of the message. In this case: `Ti`
- `isMainFrame`: A boolean determine if the message was sent from the main-frame

For sending messages from the app to the Web View, use `evalJS` to call your JS methods like this:

Send messages from the app to the Web View

```
webView.evalJS('myJSMethod();');
```

Check out the example file for sending and receiving message back and forth.

Note 1: Since 2.3.0, this module also supports synchronous communication via `evalJSSync`. Instead of `evalJS`, it takes only one parameter (the code to evaluate) and no callback. Instead, it returns the JavaScript evaluation result directly. While we understand there might be use-cases for this functionality, we highly recommend using asynchronous communication instead since synchronous method calls with block the user interface and may cause in an app-termination for long-running evaluations.

Note 2: Since 2.4.0, this modules also supports Ti.App-like events that allows the developer to fire events and add event-listeners to ease the communication between the app and the web-view. These are:

- fireEvent
- addEventListener
- removeEventListener

Different to the Ti.App events, these ones are fired on the top-level module instance.

Ti.App events

```
var WK = require('ti.wkwebview');

// Fire event
WK.fireEvent('myEvent', { message: 'Titanium rocks!' });

// Listen to events
WK.addEventListener('myEvent', function(e) { ... });
```

The above events can be used on the JavaScript side as well. The following is an example on how to use them in your HTML file (ensure that there is no concurring WK variable):

Ti.App events in HTML

```
<!DOCTYPE html>
<html>
 <head>
  <title>Local HTML</title>
 </head>
 <body>
  <p>Hello world!</p>
  <script type="text/javascript">
   // Listen to the native Titanium event
   WK.addEventListener('testEvent', function(e) { // Fire back an event to the native
Titanium app
    WK.fireEvent('testEventBack', { message: 'It worked!' });
   });
  </script>
 </body>
</html>
```

Saving Images

If you want to allow users to browse the internet (e.g. by using a remote site), ensure to include the `NSPhotoLibraryAddUsageDescription` key in `tiapp.xml`. Different to the `UIWebView` and its `Ti.UI.WebView`implementation, the `WKWebView` is more configurable but require the developer to act more carefully.

Inter-App Communication

For using the URL schemes `mailto`, `tel`, `sms` and `itms-services` you only have to set the `allowedURLSchemes` property. Other URL schemes (i.e. `fb`) additionally need the corresponding `LSApplicationQueriesSchemes` key in the iOS plist dictionary of your `tiapp.xml`:

iOS plist dictionary

```
<key>LSApplicationQueriesSchemes</key>
<array>
 <string>fb</string>
</array>
```

Generic Property Observing

You can listen to changes in some of the native properties (see the native KVO-capability).

```
// Start listening for changes
webView.startListeningToProperties([ 'title' ]); // Add an event listener for the
change
webView.addEventListener( 'title' , function(e) { // Check for e.value }); // Remove
listening to changes (remove the event listener as well to keep it more clean)
webView.stopListeningToProperties([ 'title' ]);
```

Configuration

Use the configuration API to configure the initial web-view. This property can only be set when creating the webview and will be ignored when set afterwards.

Configuration API

```
var WK = require('ti.wkwebview');
var config = WK.createConfiguration({ allowsPictureInPictureMediaPlaback: true });
var webView = WK.createWebView({ configuration: config });
```

Properties

Name	Type
`mediaTypesRequiringUserActionForPlayback`	`AUDIOVISUAL_MEDIA_TYPE_*`
`suppressesIncrementalRendering`	`Boolean`
`preferences`	`Object` • `minimumFontSize (Double)` • `javaScriptEnabled (Boolean)` • `javaScriptCanOpenWindowsAutomatically (Boolean)`
`allowsInlineMediaPlayback`	`Boolean`
`allowsAirPlayMediaPayback`	`Boolean`
`allowsPictureInPictureMediaPlaback`	`Boolean`
`processPool`	ProcessPool

Process pool

Use process pools to share cookies between webviews. Process pools do not take arguments, just pass the same reference to multiple web views.

Process pools

```
var WK = require( 'ti.wkwebview' );
var pool = WK.createProcessPool();
var config1 = WK.createConfiguration({ processPool: pool });
var config2 = WK.createConfiguration({ processPool: pool });
var firstWebView = WK.createWebView({ configuration: config1 });
var secondWebView = WK.createWebView({ configuration: config2 });
```

Handle custom URL-schemes

- The custom url-scheme has to be registered in the `allowedURLSchemes` array-property
- Passes the `url` for a custom url-scheme
- The event was introduced because iOS 10+ causes issues for some custom url-schemes and forwarding the url in an event is a simpler short term solution than implementing the `WKURLSchemeHandler` which is iOS 11+ only.

```
// Add an event listener to listen for a custom URL-scheme
webView.addEventListener('handleurl', function(e) {
  var handler = e.handler;

  Ti.Platform.openURL(e.url);
  handler.invoke(WK.ACTION_POLICY_CANCEL); // ACTION_POLICY_CANCEL or
ACTION_POLICY_ALLOW
});
```