Getting Started with the Facebook SDK for JavaScript

The Facebook SDK for JavaScript provides a rich set of client-side functionality that:

- Enables you to use the Like Button and other Social Plugins on your site.
- Enables you to use Facebook Login to lower the barrier for people to sign up on your site.
- Makes it easy to call into Facebook's primary API, called the Graph API.
- Launch Dialogs that let people perform various actions like sharing stories.
- Facilitates communication when you're building a game or an app tab on Facebook.

The SDK, social plugins and dialogs work on both desktop and mobile web browsers.

In this guide:

Loading and Initialization

The following code will load and initialize the JavaScript SDK with the most common options. Replace `YOUR_APP_ID` and `WWW.YOUR_DOMAIN.COM` with the appropriate values from the App Dashboard.

This code should be placed directly after the opening `<body>` tag.

```
<div id="fb-root"></div>
<script>
  window.fbAsyncInit = function() {
    // init the FB JS SDK
    FB.init({
      appId      : 'YOUR_APP_ID',                        // App ID from the app dashboard
      channelUrl : '//WWW.YOUR_DOMAIN.COM/channel.html', // Channel file for x-domain comms
      status     : true,                                 // Check Facebook Login status
      xfbml      : true                                  // Look for social plugins on the page
    });

    // Additional initialization code such as adding Event Listeners goes here
  };

  // Load the SDK asynchronously
  (function(d, s, id){
     var js, fjs = d.getElementsByTagName(s)[0];
     if (d.getElementById(id)) {return;}
     js = d.createElement(s); js.id = id;
     js.src = "//connect.facebook.net/en_US/all.js";
     fjs.parentNode.insertBefore(js, fjs);
   }(document, 'script', 'facebook-jssdk'));
</script>
```

If your web site or online service, or a portion of your web site or service, is directed to children under 13, please read the instructions on how to modify your initialization code.

Asynchronous Loading

Using the method above, the SDK is loaded asynchronously so it does not block loading other elements of your page. The function assigned to `window.fbAsyncInit` is run as soon as the SDK source has finished loading. Any code that you want to run after the SDK is loaded should be placed within this function and after the call to `FB.init`. For example, this is where you would test the logged in status of the user or subscribe to any Facebook events in which your application is interested.

You can also use the SDK alongside other JavaScript libraries. Take a look at our how-tos for using the SDK with other libraries:

- Using the Facebook SDK for JavaScript with jQuery
- Using the Facebook SDK for JavaScript with RequireJS

Initialization Parameters

In this example, the `en_US` version of the SDK is initialized, which means that all the dialogs and UI will be in US English. Take a look at Localization to load locale-specific versions of the SDK.

By setting `status` to `true`, the SDK will attempt to get information about the current user by hitting the OAuth endpoint. Setting `status` to `false` will improve page load times, but you'll need to manually check for login status to get an authenticated user. You can find out more about this process by looking at Facebook Login.

With `xfbml` set to `true`, the SDK will parse the DOM to find and initialize social plugins. If you're not using social plugins on the page, setting `xfbml` to `false` will improve page load times. You can find out more about this by looking at Social Plugins.

See the FB.init documentation for a full list of available initialization options.

The `fb-root` tag

The JavaScript SDK requires the `fb-root` element to be present in the page.

The `fb-root` element must not be hidden using `display: none` or `visibility: hidden`, or some parts of the SDK will not work properly in Internet Explorer.

The SDK inserts elements into `fb-root` which expect to be positioned relative to the body or relative to an element close to the top of the page. It is best if the `fb-root` element is not inside of an element with `position: absolute` or `position: relative`. If you must place the `fb-root` element inside of a positioned element, then you should also give it a position close to the top of the body or some parts of the SDK may not work properly.

Adding a Channel File

Adding a Channel File greatly improves the performance of the JS SDK by addressing issues with cross-domain communication in certain browsers. The contents of the `channel.html` file should be just a single line:

```
<script src="//connect.facebook.net/en_US/all.js"></script>
```

The channel file should be set to be cached for as long as possible. When serving this file, you should send valid `Expires` headers with a long expiration period. This will ensure the channel file is cached by the browser and not reloaded with each page refresh. Without proper caching, users will suffer a severely degraded experience. A simple way to do this in PHP is:

```
<?php
$cache_expire = 60*60*24*365;
header("Pragma: public");
header("Cache-Control: max-age=".$cache_expire);
header('Expires: ' . gmdate('D, d M Y H:i:s', time()+$cache_expire) . ' GMT');
?>
<script src="//connect.facebook.net/en_US/all.js"></script>
```

The `channelUrl` parameter within `FB.init()` is optional, but strongly recommended. Providing a channel file can help address three specific known issues.

- Pages that include code to communicate across frames may cause Social Plugins to show up as blank without a `channelUrl`.
- if no `channelUrl` is provided and a page includes auto-playing audio or video, the user may hear two streams of audio because the page has been loaded a second time in the background for cross domain communication.
- a channel file will prevent inclusion of extra hits in your server-side logs. If you do not specify a `channelUrl`, you should remove page views containing `fb_xd_bust` or `fb_xd_fragment` parameters from your logs to ensure proper counts.

The `channelUrl` must be a fully qualified URL matching the page on which you include the SDK. In other words, the channel file domain must include `www` if your site is served using `www`, and if you modify `document.domain` on your page you must make the same `document.domain` change in the `channel.html` file as well. The protocols must also match. If your page is served over `https`, your `channelUrl` must also be `https`. Remember to use the matching protocol for the script `src` as well. The sample code above uses protocol-relative URLs which should handle most `https` cases properly.

Debugging

To improve performance, the JavaScript SDK is loaded minified. You can also load a debug version of the JavaScript SDK that includes more logging and stricter argument checking as well as being unminified. To do so, change the URL you load to this:

```
js.src = "//connect.facebook.net/en_US/all/debug.js";
```

Localization

The JavaScript SDK is available in all locales that are supported by Facebook.

To change the locale of the SDK to match the locale of your site, change `en_US` to a supported locale code when loading the SDK's source. For example, if your site is in Spanish, using the following code to load the SDK will cause all Social Plugins to be rendered in Spanish.

```
<script>
  (function(d){
     var js, id = 'facebook-jssdk'; if (d.getElementById(id)) {return;}
     js = d.createElement('script'); js.id = id; js.async = true;
     js.src = "//connect.facebook.net/es_LA/all.js";
     d.getElementsByTagName('head')[0].appendChild(js);
   }(document));
</script>
```

Note that if you use a Channel File, you should also change the URL to match the URL from which you load the JS SDK on your main pages.

Social Plugins

Social Plugins such as the Like Button and Comments Plugin allow users to take lightweight social actions across your site.

They are embedded in your web page with special-named tags. They can be specified in one of two formats.

HTML5 style (preferred):

```
<div class="fb-like" data-send="true" data-width="450" data-show-faces="true"></div>
```

Or XHTML style:

```
<html xmlns:fb="http://ogp.me/ns/fb#">
...
<fb:like send="true" width="450" show_faces="true"></fb:like>
```

The SDK will automatically scan your page for either of these style of tags if you set `xfbml: true` in the options you pass into `FB.init()` and set up the plugin for you.

You can leave `xfbml` to `false` if you don't have any social plugins on your page to improve performance.

Dialogs

Facebook Dialogs provide a simple way to provide functionality such as sharing content to a users timeline, taking payments, sending messages and sending requests. To display a dialog, you'll use the JS SDK's `FB.ui()` method.

Dialogs displayed with the JS SDK are automatically formatted for the context in which they are loaded - mobile web, or desktop web.

The following example uses the `FB.ui()` method to invoke the Feed Dialog to allow a user to post a link to their timeline:

```
FB.ui(
  {
   method: 'feed',
   name: 'The Facebook SDK for Javascript',
   caption: 'Bringing Facebook to the desktop and mobile web',
   description: (
      'A small JavaScript library that allows you to harness ' +
      'the power of Facebook, bringing the user\'s identity, ' +
      'social graph and distribution power to your site.'
   ),
   link: 'https://developers.facebook.com/docs/reference/javascript/',
   picture: 'http://www.fbrell.com/public/f8.jpg'
  },
  function(response) {
    if (response && response.post_id) {
      alert('Post was published.');
    } else {
      alert('Post was not published.');
    }
  }
);
```

Facebook Login

Facebook Login allows users to register or sign in to your app with their Facebook identity.

We have a larger guide on how to use the JS SDK to implement Facebook Login.

The JS SDK methods you will use when implementing Facebook Login include:

- `FB.login()` — ask the user to login or request additional permissions
- `FB.getLoginStatus()` — asynchronous method to get the current Facebook login status of the user

In addition to the above methods, there are several relevant events that you may subscribe to using `FB.Event.subscribe()`:

- `auth.login`
- `auth.logout`
- `auth.statusChange`
- `auth.authResponseChange`

Calling the Graph API

To read or write data to the Graph API, you'll use the JS SDK's `FB.api()` method.

We have a larger guide on getting started with the Graph API.

This simple example reads the currently logged-in users name:

```
FB.api('/me', function(response) {
  alert('Your name is ' + response.name);
});
```

Games and Page Tab Apps

The JavaScript SDK provides a way for Canvas pages on Facebook to communicate with the parent facebook.com page. This is useful for resizing the Canvas iframe, collecting performance data, and optimizing static resources. For more information see the Apps on Facebook guide.

Custom Flash Hiding Callback

A special consideration for Flash Developers on Canvas - if you are hosting an Adobe Flash Application it is recommended that you set the `wmode` of the Flash object to `opaque`.

If you must use `wmode` values of `window` or `direct`, Canvas will automatically hide and display the Flash object when Dialogs, Ticket flyouts, Chat Tabs and Notifications display.

Developers who wish to provide a custom hide and display experience may pass a JavaScript function in the `hideFlashCallback` option for FB.init. This function will be executed whenever the Flash object is hidden or displayed due to user behavior (clicking on a Notification, etc.) and can be used by a developer to take the appropriate actions: hiding or displaying their Flash object. It receives a parameter of type `object` that contains two properties:

- `state`: Supports values `'opened'` or `'closed'`.
- `elem`: The HTML element that will be hidden or displayed.

The custom action must complete within 200ms or the Flash object will be hidden automatically regardless.

Sample implementation:

```
function(params) {
  if (params.state == 'opened') {
    // Hide the Flash object
    FB.Canvas.hideFlashElement(params.elem);
  } else {
    // Display the Flash object
    FB.Canvas.showFlashElement(params.elem);
}
```

As displayed in the example above, Flash content can be manually hidden and displayed via FB.Canvas.hideFlashElement and FB.Canvas.showFlashElement.

Web Apps loaded inside an iOS or Android Webview

The Javascript SDK is compatible web pages loaded inside webviews within in the native Facebook iOS or Android apps. When users follow links to web sites or web apps within the native Facebook app, they'll remain inside the native Facebook app experience, and will see the link they clicked on within a UIWebView (iOS) or WebView (Android).

While webviews in native apps are normally sandboxed from the standard browser session, the Facebook app injects a session into the webview so that your social plugins and authentication still work as if they were being loaded in the default device browser's shared session space.

If your app uses the SDK to launch dialogs via `FB.ui()`, or authenticate the user with `FB.login()`, these features will be replaced by their native equivalents.

약 3달 전에 업데이트됨