JS Load Problems and Using Require JS

JavaScript Loading Issues

MENTORLABS℠

- ➢ JavaScript is a truly amazing tool for front-end programming, creating interactive, feature-rich websites and fast, seamless web applications.

- ➢ Every **front-end developer** knows JavaScript, but when used without caution or expertise, it can be a double-edged sword.

- ➢ Poorly written JavaScript code can slow your website, negatively affecting load times and rendering speed. In this article, we'll cover some useful tools to help you avoid the "dark side effects" of JavaScript.

➢ Javascript libraries that can add dozens or even hundreds of kilobytes to your page

➢ First, it's important that all elements in the <head> section are pre-loaded, before the visitor sees anything in browser, then all subsequent elements are ordered to load in a logical way.

➢ Any JavaScript inside the <head> section can slow down a page's rendering. Here's a look at the difference between an optimized and an unoptimized page load:</head></head>
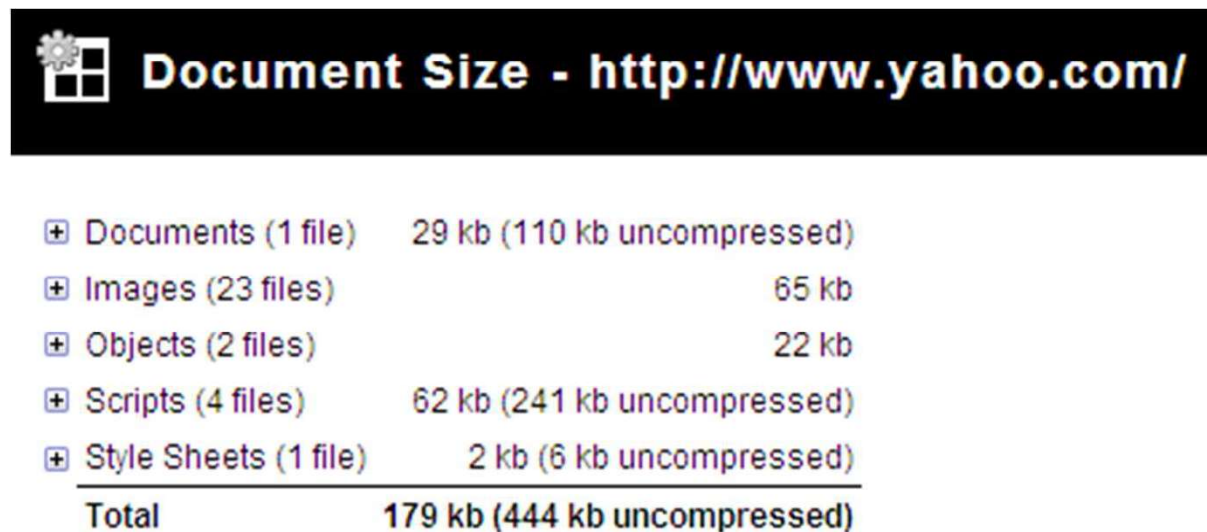
MENTORLABS℠

| | 0s | 2.0s | 2.0s | 2.0s | 2.0s |

MENTORLABS

➢ Like any optimization technique, it helps to measure and figure out what parts are taking the longest.

➢ You might find that your images and HTML outweigh your scripts. Here's a few ways to investigate:

➢ The **Firefox** <u>**web-developer toolbar**</u> lets you see a breakdown of file sizes for a page (Menu > Web Developer). Look at the breakdown and see what is eating the majority if your bandwidth, and which files:
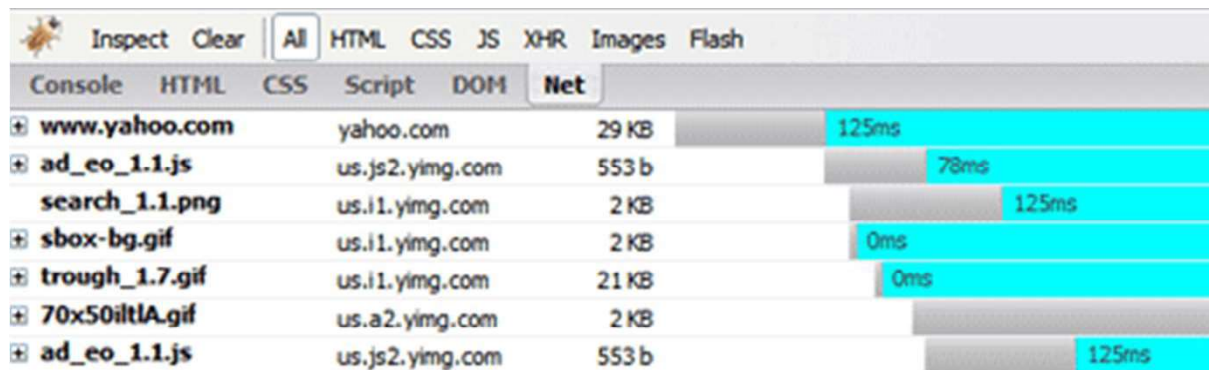
Document Size - http://www.yahoo.com/

⊞ Documents (1 file)	29 kb (110 kb uncompressed)
⊞ Images (23 files)	65 kb
⊞ Objects (2 files)	22 kb
⊞ Scripts (4 files)	62 kb (241 kb uncompressed)
⊞ Style Sheets (1 file)	2 kb (6 kb uncompressed)
Total	**179 kb (444 kb uncompressed)**

➢ The **Firebug Plugin** also shows a breakdown of files – just go to the "Net" tab. You can also filter by file type:

MentorLabs

➢ First, you can try to make the javascript file smaller itself. There are lots of utilities to "crunch" your files by removing whitespace and comments.

1. **Run JSLint** (online or downloadable version) to analyze your code and make sure it is well-formatted.

2. Use YUI Compressor to compress your javascript from the command line. There are some online packers, but the YUI Compressor (based on Rhino) actually analyzes your source code so it has a low chance of changing it as it compresses, and it is scriptable.

Install the YUI Compressor (it requires Java), then run it from the command-line (x.y.z is the version you downloaded):

```
java -jar yuicompressor-x.y.z.jar myfile.js -o myfile-min.js
```

MENTORLABS

➢ **Place your javascript at the end of your HTML** file if possible. Notice how Google analytics and other stat tracking software wants to be right before the closing </body> tag.

➢ This allows the majority of page content (like images, tables, text) to be loaded and rendered first. The user sees content loading, so the page looks responsive. At this point, the heavy javascripts can begin loading near the end.

➢ Only core files that are absolutely needed in the beginning of the page load should be there. The rest, like cool menu effects, transitions, etc. can be loaded later. You want the page to appear responsive (i.e., something is loading) up front.

Delay Your Javascript

➢ Rather than loading your javascript on-demand (which can cause a noticeable gap), **load your script in the background, after a delay**.

```javascript
var delay = 5;
setTimeout("loadExtraFiles();", delay * 1000);
```

➢ This will call loadExtraFiles() after 5 seconds, which should load the files you need

MENTORLABS
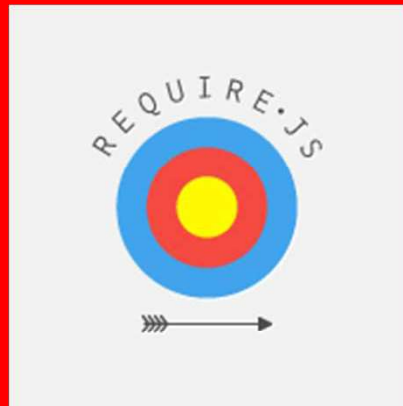
Require JS

MENTORLABS℠

Introduction

➢ RequireJS is a JavaScript library that is used for loading in JavaScript files. It implements a specification called AMD, which stands for *Asynchronous Module Definition*.

MENTORLABS

```
//exporter.js
var sayHi = function(): void {
    console.log("Hello!");
}


export = sayHi;
```

```
//importer.js
import sayHi = require('./exporter');
sayHi();
```

➢ Normally, when including multiple JavaScript libraries, an HTML file would include a bunch of script tags similar to the following:

```
<script src="jQuery.js"></script>
<script src="library1.js"></script>
<script src="library2.js"></script>
<script src="library3.js"></script>
```

➢ Loading libraries in this way isn't an issue when there are only a few libraries to include, but it can become a real problem when there are a large number of JavaScript files, as it can be a nightmare to manage and also requires a request for each and every library upon page load.

MENTORLABS℠

➢ Dependencies can be an issue too. If a library is dependent on another that has not been loaded yet, it will cause an error.

➢ (It is possible that you have seen the "$ is undefined" error once or twice.) If we change the import order of the script tags to place jQuery at the bottom, and the libraries are dependent on jQuery, the application will throw errors.

```
<script src="library1.js"></script>
<script src="library2.js"></script>
<script src="library3.js"></script>
<script src="jQuery.js"></script>
```

MENTORLABS℠

Solution

➤ RequireJS solves the preceding problems, as it takes care of loading in the JavaScript libraries and ensures that they are loaded in the correct order and when they are needed.

➤ In order to understand how this works, we must first take a look at the AMD specification.

➤ The AMD specification outlines a single function, called define, which looks like this:

```
define(id?, dependencies?, factory);
```

MENTORLABS

1. id: This is the id for the module that is being defined. The id argument is optional, and we won't really be using it in the context of JET.

2. dependencies: This is an array of JavaScript libraries that must be loaded for the factory to run.

3. factory: This is the code that will be running once all the dependencies are loaded.

MentorLabs

➢ The define syntax in RequireJS is fairly straightforward, and by applying the AMD define function, we can write the following:

```
define(['jquery', 'myLibrary'], function($) {
    // Application code logic here
});
```

➢ The argument that is passed into the factory function ($) will reference the corresponding library from the dependency list. In this case, it will be used to reference jQuery, as it is the first item in the dependency list.

MENTORLABS℠

➢ Ordering is important here. Although define will take care of JavaScript loading dependencies, the order of the arguments into the function matters. Swapping the dependency ordering can cause mapping inconsistencies.

```
define(['myLibrary', 'jquery'], function($, myLib) {
    // Application code logic here
});
```

MentorLabs

➢ RequireJS takes the dependency list you specify, works out the correct ordering, and then adds the script tags into the head of the page dynamically.

1. Defining Block Skeleton with dependencies

```
1    define(['jquery', 'library1', 'library2', 'library3'], function ($) {
2
3    });
```

2. The resulting HTML script tags.

```html
<!DOCTYPE html>
▼ <html>
  ▼ <head>
      <title>Example require</title>
      <script data-main="js/app" src="js/libs/require.js"></script>
      <script type="text/javascript" charset="utf-8" async data-
      requirecontext="_" data-requiremodule="app" src="js/
      app.js"></script>
      <script type="text/javascript" charset="utf-8" async data-
      requirecontext="_" data-requiremodule="app/main" src="js/libs/
      ../app/main.js"></script>
      <script type="text/javascript" charset="utf-8" async data-
      requirecontext="_" data-requiremodule="jquery" src="js/libs/
      jquery.js"></script>
      <script type="text/javascript" charset="utf-8" async data-
      requirecontext="_" data-requiremodule="library1" src="js/libs/
      library1.js"></script>
      <script type="text/javascript" charset="utf-8" async data-
      requirecontext="_" data-requiremodule="library2" src="js/libs/
      library2.js"></script>
      <script type="text/javascript" charset="utf-8" async data-
      requirecontext="_" data-requiremodule="library3" src="js/libs/
      library3.js"></script>
    </head>
    <body> </body> = $0
</html>
```

MENTORLABS

require vs. define

➢ We have looked into the define function, but RequireJS also has a function called require.

➢ It is possible to use both the require and define functions within RequireJS.

➢ As a rule of thumb, require should be used to run immediate functionalities, and define should be used to define modules of code that can be used in multiple locations within an application.

➢ In modular applications, this will result in a single require block to load in all the immediate application logic that is necessary for the application to initialize and run, followed by multiple define blocks used across the application for individual modules of code.

Download RequireJS from its home page: **http://requirejs.org/docs/download.html.**

➢ Now that you understand what problems RequireJS solves, let's take a look at how it is implemented within an Oracle JET application.

➢ RequireJS is included out of the box when you scaffold your JET application, and the following is added at the bottom of the index.html file of a new project:

```html
<script type="text/javascript" src="js/libs/require/require.js"></script>
<script type="text/javascript" src="js/main.js"></script>
```

- ➢ The main.js file is an important piece of the RequireJS puzzle that comes bundled with a JET application.

- ➢ The file contains a list of all the library dependencies required for the project to run (also known as the configuration block).

- ➢ It will also include a require block, which loads all the libraries required to kick-start the JET application.

- ➢ It is essentially an entry point into your application code.

Configuration

➢ Within the main.js file, you will have what is called the "configuration block."

➢ The configuration essentially sets the base location (baseUrl) of the JavaScript modules and declares all the libraries that the application will be using.

Configuration Options

Following are the configuration options which can be set while loading the first application module –

- **baseUrl** – It is a route path for all modules which are loaded through RequireJS. The baseUrl is indicated by a string starting with "slash (/)", containing a protocol and ending with ".js" extension. If there is no baseUrl specified, then RequireJS uses the *data-main* attribute path as baseUrl.

- **paths** – It specifies the path mappings for modules which are relative to the baseUrl. It automatically adds the .js extension to a path when mapping the module name.

- **shim** – It provides the usage of non AMD libraries with RequireJS by configuring their dependencies and exporting their global values.

- **map** – For the given module, an application uses same module of different versions for different objectives by sharing their ids to make use of same code for different conditions.

- **config** – It provides the configuration to a module by using the *config* option and this can be done by using the special dependency "module" and calling its *module.config()* function.

```
requirejs.config(
{
  baseUrl: 'js',

  // Path mappings for the logical module names
  // Update the main-release-paths.json for release mode when updating the
  mappings
  paths:
  //injector:mainReleasePaths
  {
    'knockout': 'libs/knockout/knockout-3.4.2.debug',
    'jquery': 'libs/jquery/jquery-3.3.1',
    'jqueryui-amd': 'libs/jquery/jqueryui-amd-1.12.1',
    'promise': 'libs/es6-promise/es6-promise',
    'hammerjs': 'libs/hammer/hammer-2.0.8',
    'ojdnd': 'libs/dnd-polyfill/dnd-polyfill-1.0.0',
    'ojs': 'libs/oj/v6.0.0/debug',
```

MENTORLABS

```
    'ojL10n': 'libs/oj/v6.0.0/ojL10n',
    'ojtranslations': 'libs/oj/v6.0.0/resources',
    'text': 'libs/require/text',
    'signals': 'libs/js-signals/signals',
    'customElements': 'libs/webcomponents/custom-elements.min',
    'proj4': 'libs/proj4js/dist/proj4-src',
    'css': 'libs/require-css/css',
}  //endinjector
,
// Shim configurations for modules that do not expose AMD
shim:
{
```

```
        'jquery':
        {
          exports: ['jQuery', '$']
        }
      }
    }
    );
```

MENTORLABS

require Block

➢ A require block is used to load all the modules that are required to initialize the application, such as *ojModule* and *ojRouter*. Both are Oracle JET libraries required at this level to initialize the application routing logic.

➢ The *appController* module is also imported at this point, which is an Oracle JET specific file that contains application wide logic and properties, such as the application name and routing configuration.

• Code Snippet shows what this require block will look like when you create an Oracle JET application for the first time.

```
require(['ojs/ojcore', 'knockout', 'appController', 'ojs/ojknockout',
  'ojs/ojmodule', 'ojs/ojrouter', 'ojs/ojnavigationlist', 'ojs/ojbutton',
'ojs/ojtoolbar'],
  function (oj, ko, app) { // this callback gets executed when all required
  modules are loaded

    $(function() {

      function init() {
        oj.Router.sync().then(
          function () {
            app.loadModule();
            // Bind your ViewModel for the content of the whole page body.
            ko.applyBindings(app, document.getElementById('globalBody'));
          },
```

```
        function (error) {
          oj.Logger.error('Error in root start: ' + error.message);
        }
      );
    }

    // If running in a hybrid (e.g. Cordova) environment, we need to wait
    for the deviceready
    // event before executing any code that might interact with Cordova
    APIs or plugins.
    if ($(document.body).hasClass('oj-hybrid')) {
      document.addEventListener("deviceready", init);
    } else {
      init();
    }

  });

  }
);
```

Include RequireJS in your application home page as follows:

```html
<html>
  <head>
    <script data-main="js/main" src="js/require.js"></script>
  </head>
  <body> <h1>Require JS with Oracle JET </h1> </body>
</html>
```

➢ Here, the data-main attribute refers to the main.js dependency represented as js/main.

MENTORLABS

From here on, we can include the dependencies for each module explicitly as follows:

```
// +js/main.js
require(["js/customers"], function(cust) {
  cust.createCustomers();
});

// +js/customers.js
define(["js/orders"], function(ord) {
  return function createCustomer() {
    ord.raiseOrder();
  }
}

// +js/orders.js
define(function() {
  return function raiseOrder() {
    // doSomething
  }
}
```

Defining Modules

➢ Module is defined using the **define()** function; the same function is used for loading the module as well.

➢ Simple Name/Value Pairs

```
define({
    state: "karnataka",
    city: "bangalore"
});
```

MENTORLABS℠

Defining Functions

➢ A module can also use a function for frameworks, without having dependencies. This can be done by using the following syntax −
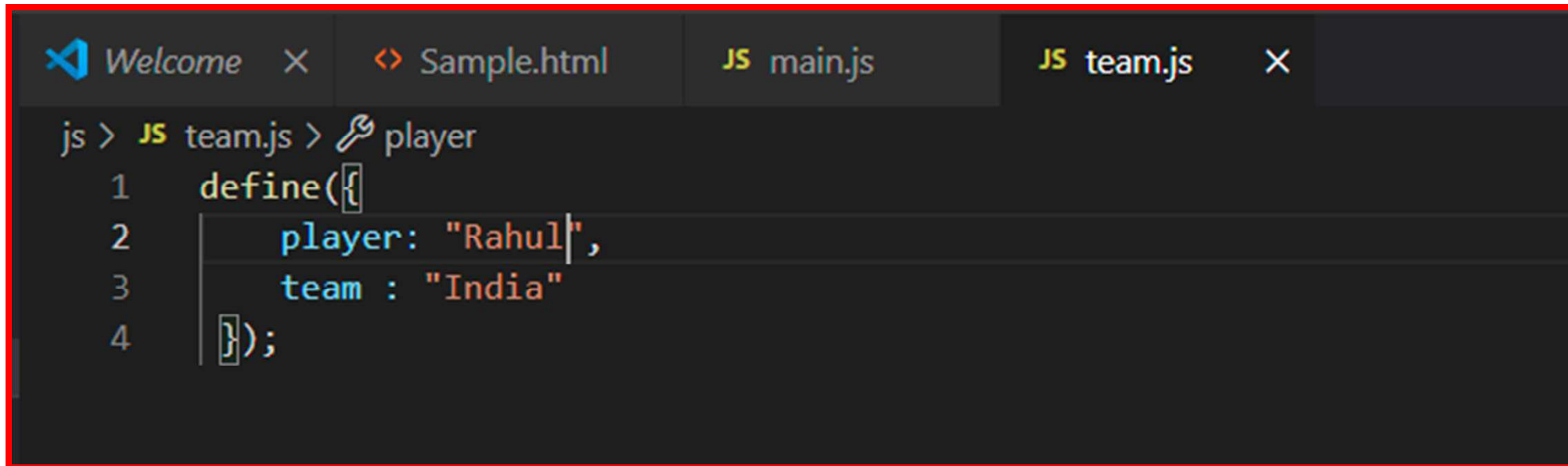
```
define(function () {

    //Do setup work here
    return {
        state: "karnataka",
        city: "bangalore"
    }
});
```

MENTORLABS

Main.js


```
Welcome        <> Sample.html      JS main.js    ×

js > JS main.js > ...
  1    define(function (require) {
  2        var myteam = require("./team");
  3        var mylogger = require("./player");
  4        alert("Player Name : " + myteam.player);
  5        mylogger.myfunc();
  6    });
```

Team.js

```
Welcome    <> Sample.html    JS main.js    JS team.js  X

js > JS team.js > 🔧 player
1    define({
2        player: "Rahul",
3        team : "India"
4    });
```

MENTORLABS℠

Player.js

```
Welcome    <> Sample.html    JS main.js    JS team.js    JS player.js    ✕

js > JS player.js > ...
1    define(function (require) {
2        var myteam = require("./team");
3
4        return {
5            myfunc: function () {
6                document.write("Name: " + myteam.player + ", Country: " + myteam.team);
7            }
8        };
9    });
```

We can add any application startup code to the callback function. For example, a knockout binding call to the callback function for the dialogWrapper element is added in the following snippet:

```
require(['app', 'ojs/ojcore', 'knockout', 'jquery',
         'ojs/ojmodel', 'ojs/ojknockout-model', 'ojs/ojdialog'],

function(app, oj, ko) // obtaining a reference to the oj namespace
{
  ko.applyBindings(new app()/*View Model instance*/,
                   document.getElementById('dialogWrapper'));
}
);
```

oj nameSpace

➢ The oj namespace is reserved by the Oracle JET framework, meaning we cannot use the same namespace in our Oracle JET application code.

➢ The oj namespace is defined in the ojs/ojcore module, and other modules which are loaded after oj belongs to the objects from them inside the oj namespace.

List of Oracle JET modules

Oracle JET Module	Description	When to Use?
ojs/ojcore	Core framework module that defines a base Oracle JET object. Includes support for prototype-based JavaScript object inheritance and extending JavaScript objects by copying their properties. The module returns oj namespace. You must include this module in any Oracle JET application.	Always
ojs/ojmodel	Oracle JET's Common Model	Use if your application uses the Oracle JET Common Model.
ojs/ojknockout-model	Utilities for integrating Oracle JET's Common Model into Knockout.js	Use if your application uses the Oracle JET Common Model, and you want to integrate with Knockout.js.
ojs/ojvalidation	Data validation and conversion services	Use if your application uses Oracle JET validators or converters outside of Oracle JET editable components.
ojs/ojknockout-validation	Support for the invalidComponentTracker binding option on input components	Use if your application uses validators or converters, and you want to track the validity of the Oracle JET components in your application.

MentorLabs

| ojs/ojcomponent:

Examples:

ojs/ojbutton

ojs/ojtoolbar

ojs/ojtabs | Oracle JET component modules. Most Oracle JET components have their own module with the same name in lowercase, except for the following components:

ojButtonset: ojs/ojbutton
ojInputPassword: ojs/ojinputtext
ojTextArea: ojs/ojinputtext
ojCombobox: ojs/ojselectcombobox
ojSelect: ojs/ojselectcombobox
ojSparkChart: ojs/ojchart
ojDialGauge: ojs/ojgauge
ojLedGauge: ojs/ojgauge
ojRatingGauge: ojs/ojgauge
ojStatusMeterGauge: ojs/ojgauge | Use component modules that correspond to any Oracle JET component in your application. |

ojs/ojknockout	Oracle JET ojComponent binding and services for Knockout.js	Use if your application includes Oracle JET components and you want to use ojComponent binding for these components in Knockout.js.
ojs/ojrouter	Class for managing routing in single page applications	Use if your single page application uses oj.Router for routing.
ojs/ojmodule	Binding for Knockout.js that implements navigation within a region of a single page application	Use if your single page application uses the ojModule binding for managing content replacement within a page region.
ojs/ojoffcanvas	Methods for controlling off-canvas regions	Use if your application uses oj.offCanvasUtils for managing off-canvas regions.
ojs/ojcube	Class for aggregating data values in ojDataGrid	Use if your application renders aggregated cubic data in an ojDataGrid component.

MentorLabs℠