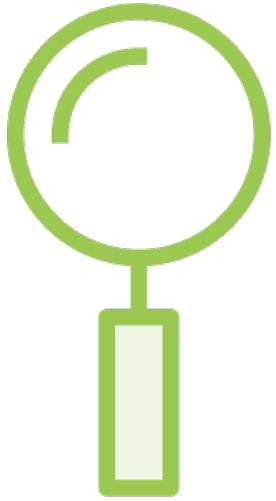
Configuring RequireJS

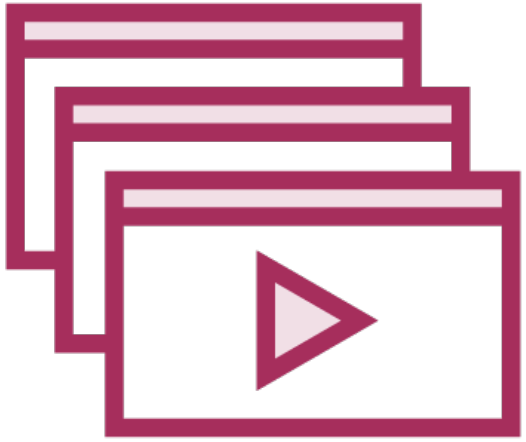
Kevin Murray

murmeister@hotmail.com

Configuration Data Easy to Find



Simple internet search

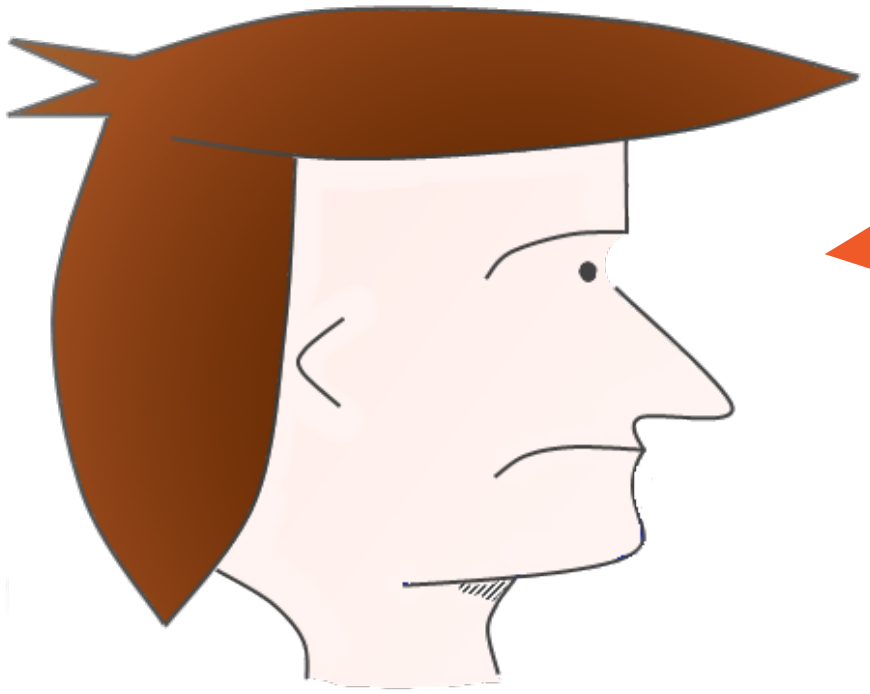


Numerous results

Copied examples

Sample Configuration

```
requirejs.config({
    shim: {
        'backbone': {
            deps: ['underscore', 'jquery'],
            exports: 'Backbone'
        },
        'underscore': {
            exports: '_'
        },
        'foo': {
            deps: ['bar'],
            exports: 'Foo',
            init: function (bar) {
                return this.Foo.noConflict();
            }
        }
    }
});
```



That sample doesn't relate to my own projects

Overview

Build on previous project

Configuration properties

- Focus on a few

Better project structure

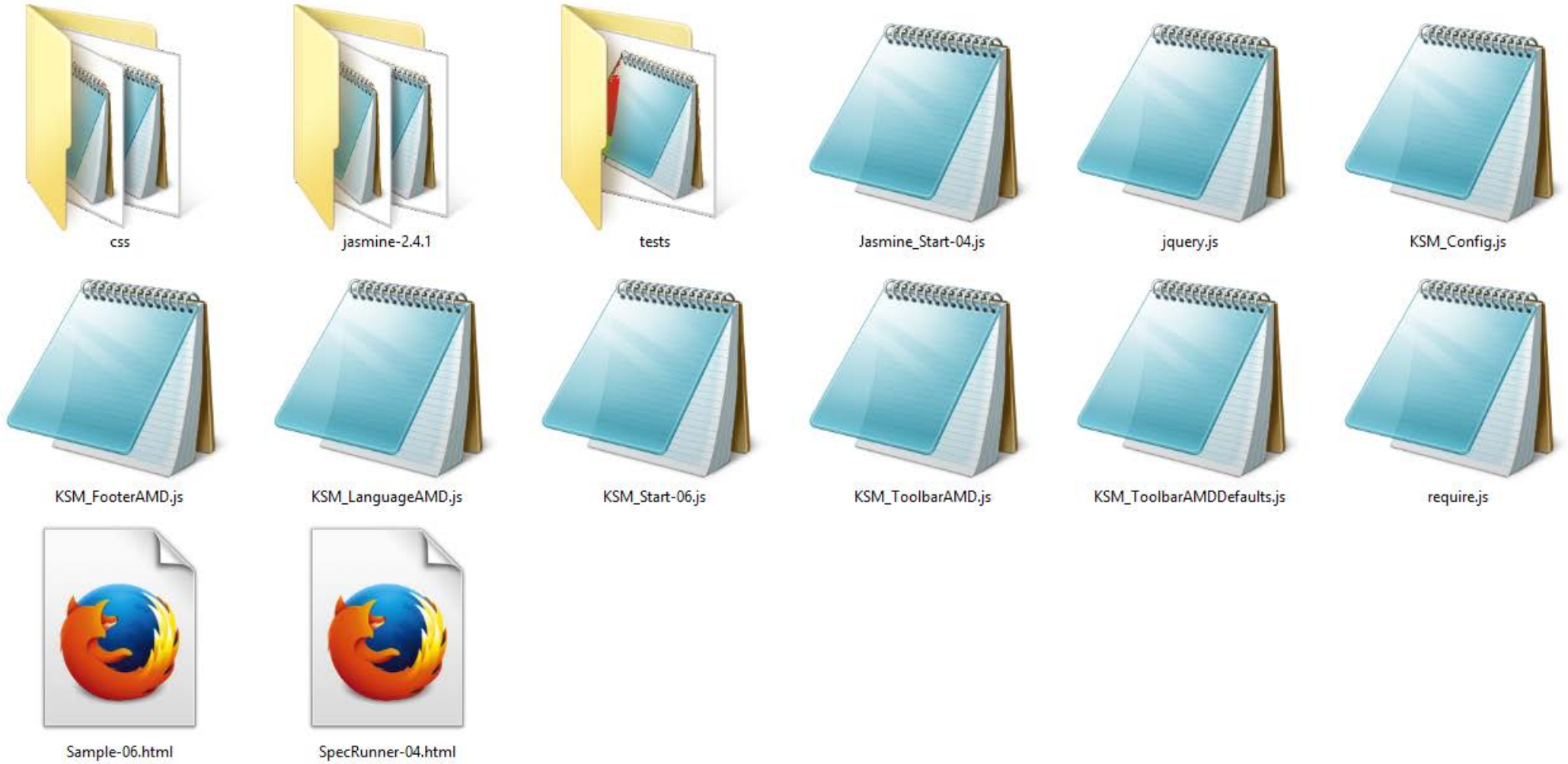
Handling legacy JavaScript libraries

“shim” property

RequireJS “data-main” Attribute

```
...
<script src="./scripts/require.js"
data-main="Components/KSM_Start-07"></script>
...
```

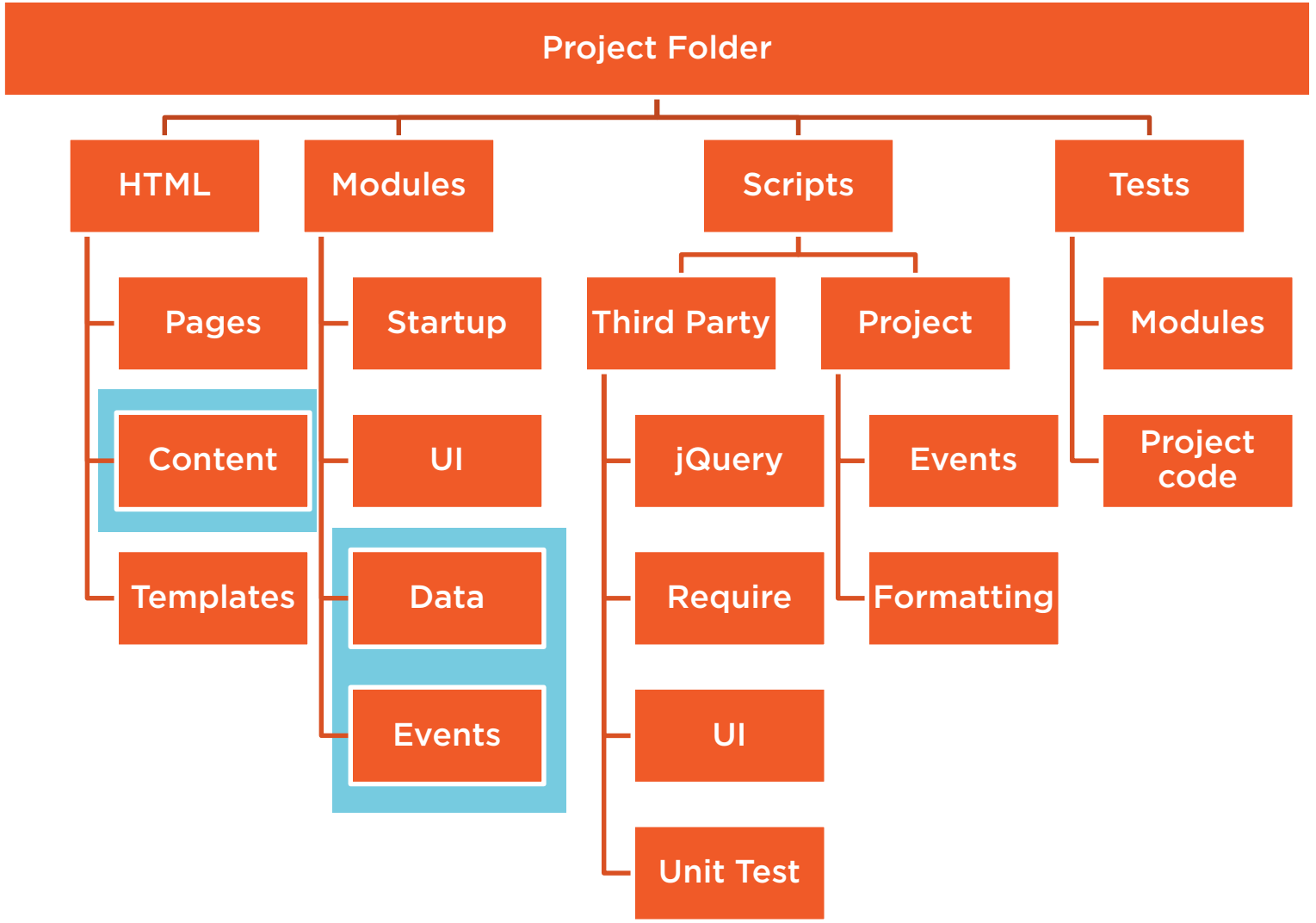
Project Folder Contents



Unit Testing Code

```
require(['KSM_ToolbarAMD', 'KSM_LanguageAMD', 'jasmine-boot'],
function(toolbar, language)
{
    window._KSM = { Toolbar: toolbar, Language: language };

    require(['tests/KSM_Toolbar-05_Spec'], function()
    {
        window.onload();
    })
});
```

Typical Project



Configuring RequireJS

File locations can be specified in configuration

Files can be organized without including path in file name

Using “baseUrl” Property

“scripts/jquery”

Look for jquery.js in the scripts folder

“../common/library”

Look for library.js in the sibling folder called “common”

Directory Access

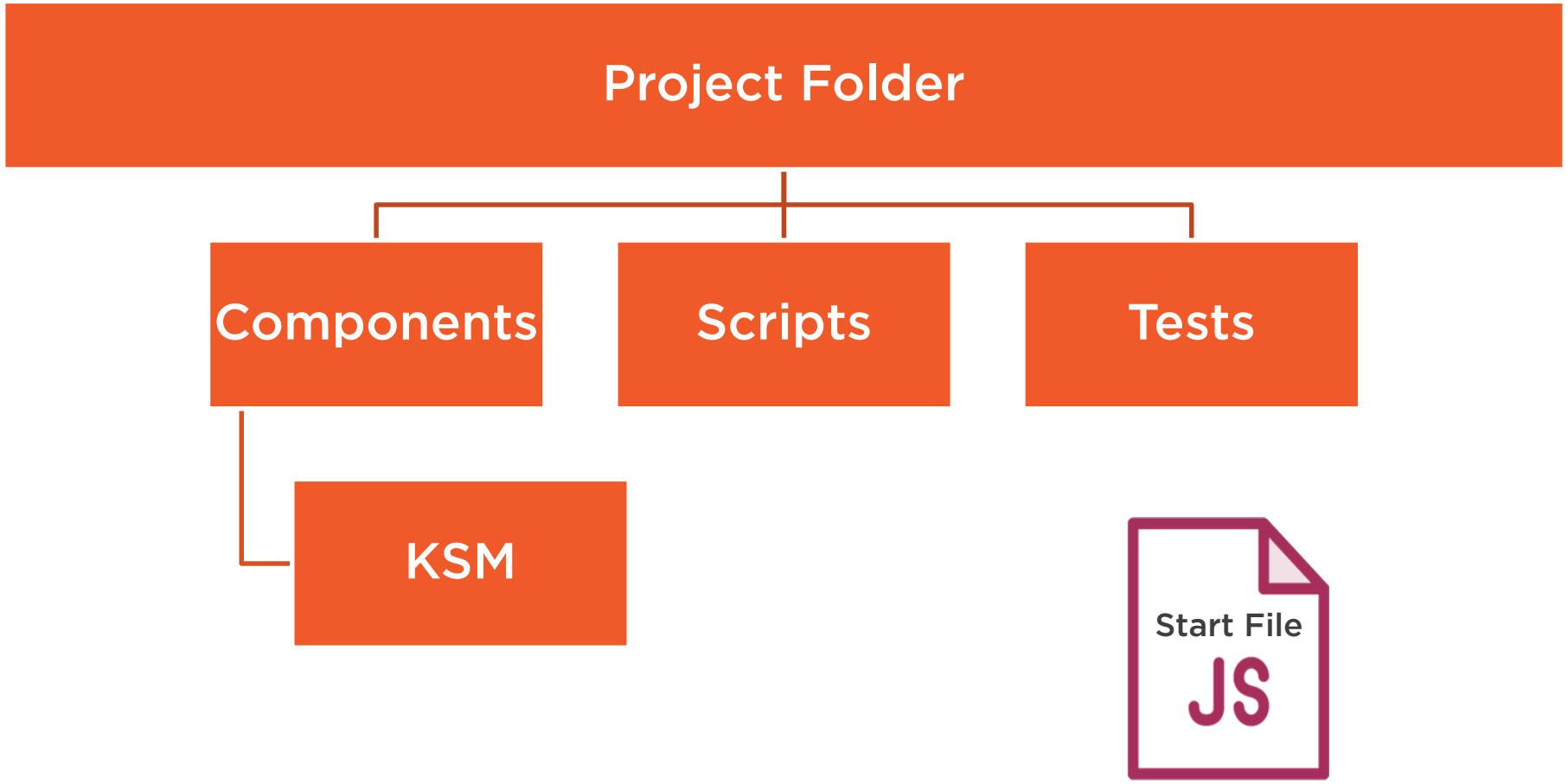
Possible security restrictions on sibling folders

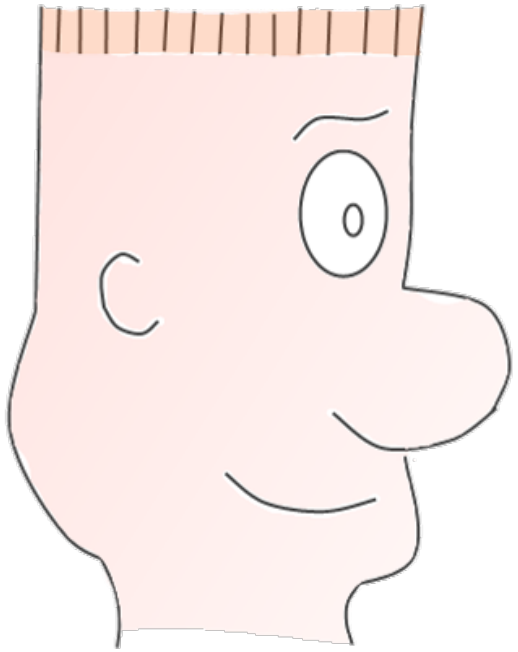
No restrictions when all files are in one folder

Absence of “baseUrl” property implies simple project

Without a “baseUrl” property configured, RequireJS looks for files in the same location as the HTML page that loads RequireJS

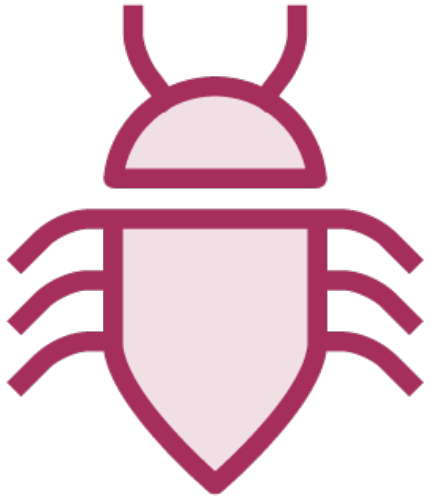
New Folder Structure



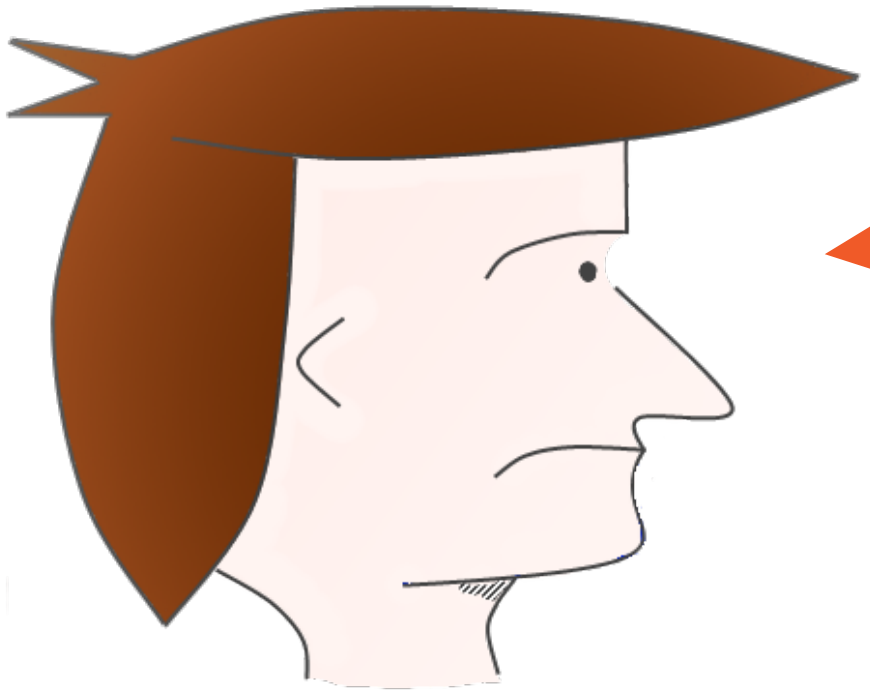


I think the start file should be in a folder called "Startup Code" or something.

Reorganizing Files



Changes to the environment change behavior

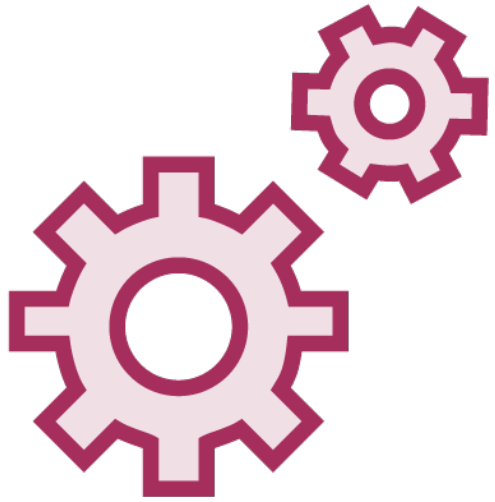


I see a difference!

Footer Module Code

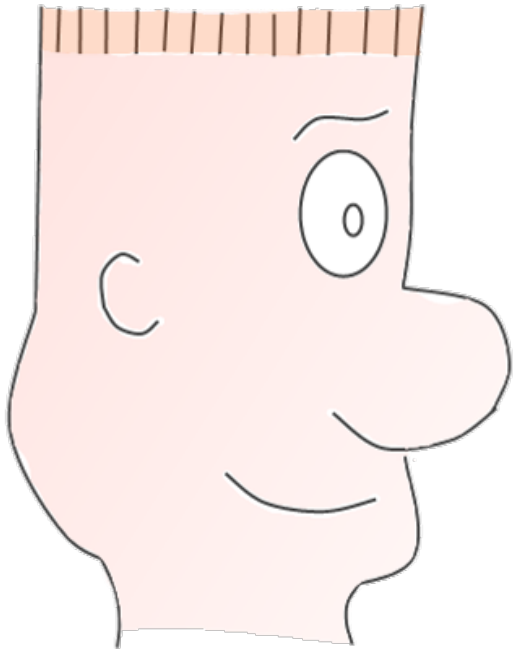
```
/* KSM_FooterAMD.js
*/
require(['jquery'], function($)
{   $('<p>')
        .addClass('footer')
        .html('Sample web page for Pluralsight course')
        .insertAfter('.page');
});
```

Using Configuration Properties

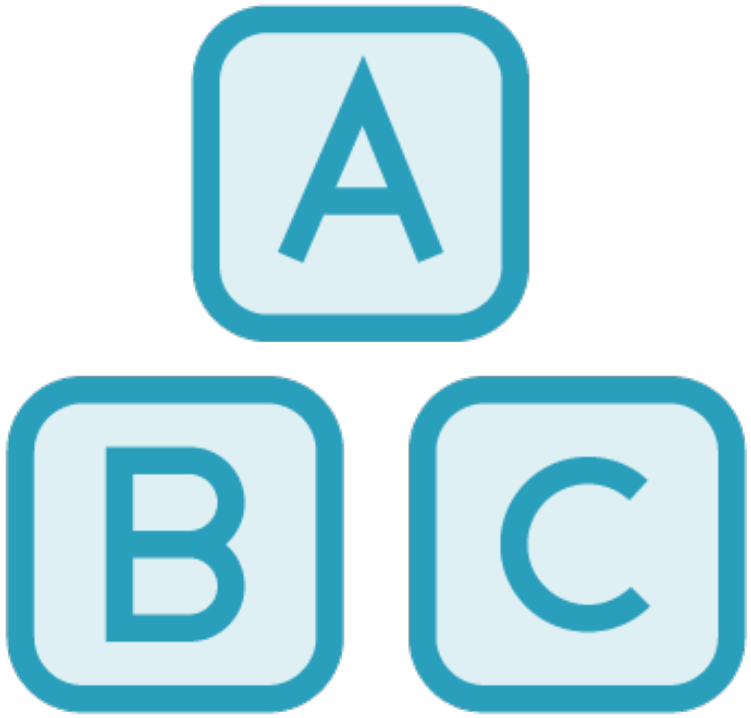


Configuration properties must come before startup code

Configuration properties must be set before RequireJS executes the callback function of the startup code



Most cases?!



Multiple possible entry points

- User permissions
- URL parameters
- Something else

Configuration may not be set

- Causing errors

Unusual circumstance

Configuration in HTML page

“require” Variable

Once RequireJS loads, it looks for the global object named “require” and uses it for configuration

Placing configuration in master HTML ensures configuration for all logic paths

Warning

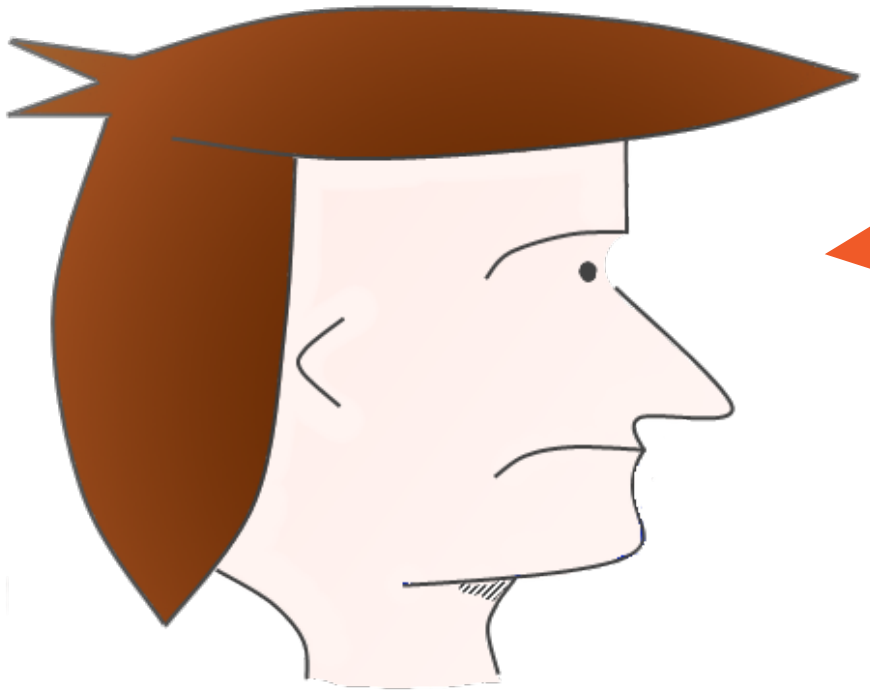
```
var require =
{
    baseUrl: ‘./’,
    paths:
...
}
```

Warning

```
window.require =
{
    baseUrl: ‘./’,
    paths:
...
}
```

“Note: It is best to use var require = {} and do not use window.require = {}, it will not behave correctly in IE.”

requirejs.org/docs/api.html#config



Internet Explorer %&^@#$!

Configuration Properties

Place configuration in HTML for dynamic web sites

Startup file is usually best place for configuration

Remaining samples use startup configuration

Using a Module Alias

Risk reduction during tests

Replacing modules or changing code

Multiple files may be involved

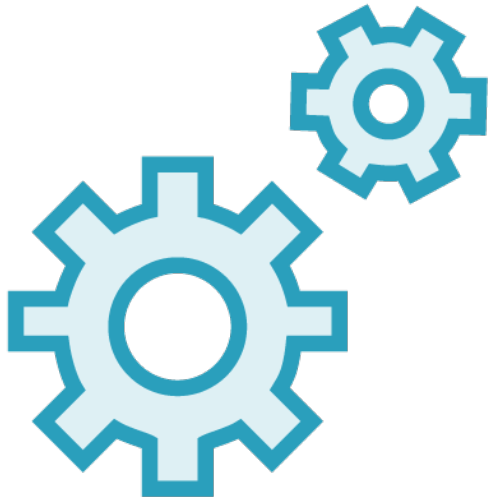
Single configuration change

Using a module alias allows for additional abstraction

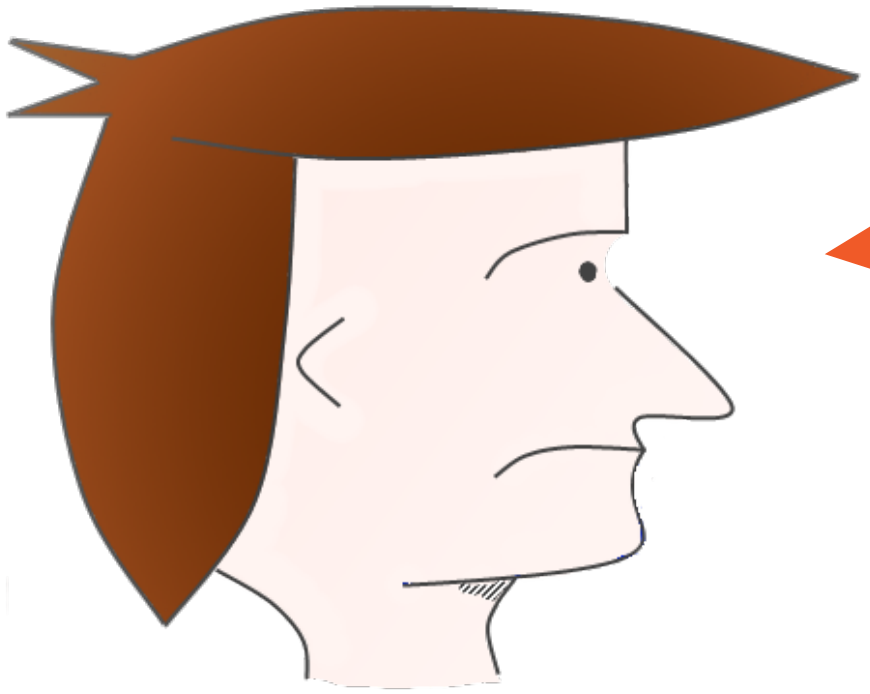
Abstraction Needed?

Confession Time

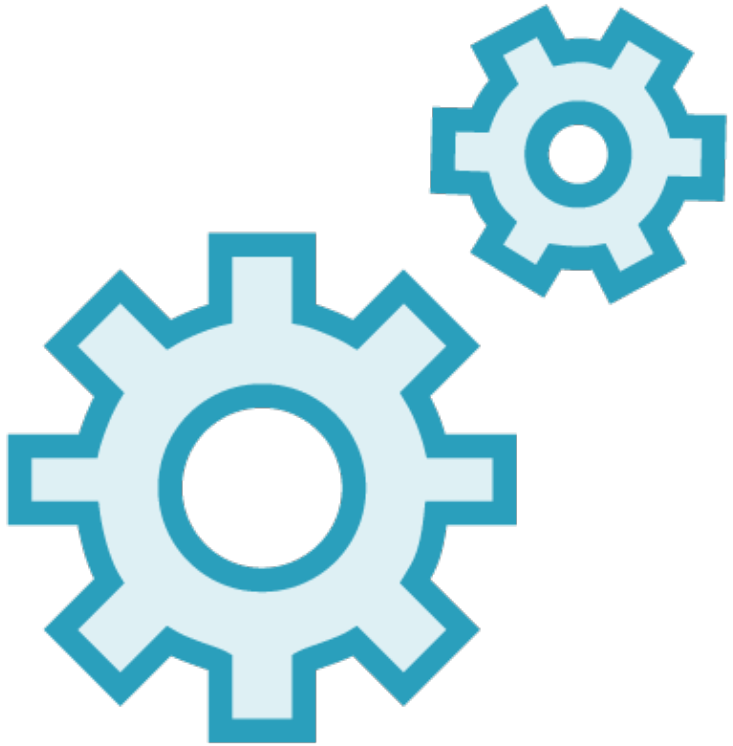
Converting legacy libraries



Configuration can eliminate need for conversion



That sure was a lot of work for nothing!



RequireJS and legacy libraries

Script loader

Configure for dependencies

Use Sample-3 from earlier

Legacy Libraries

Refactored libraries before

Not changing code this time

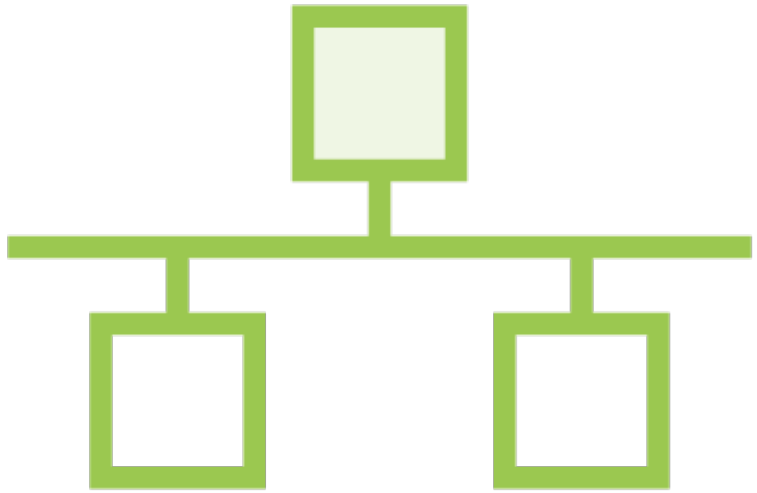
The configuration property “shim” is only required for legacy non-modular libraries

“shim” Property

No need to use “shim” for regular modules

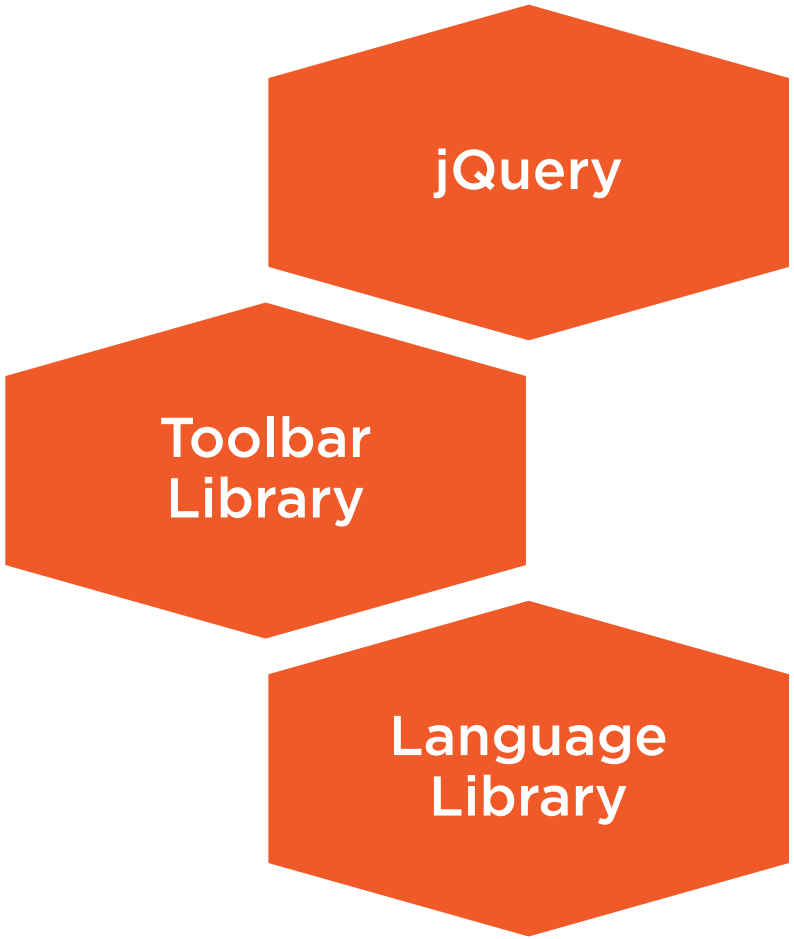
“shim” Dependencies

List sequence is not load sequence

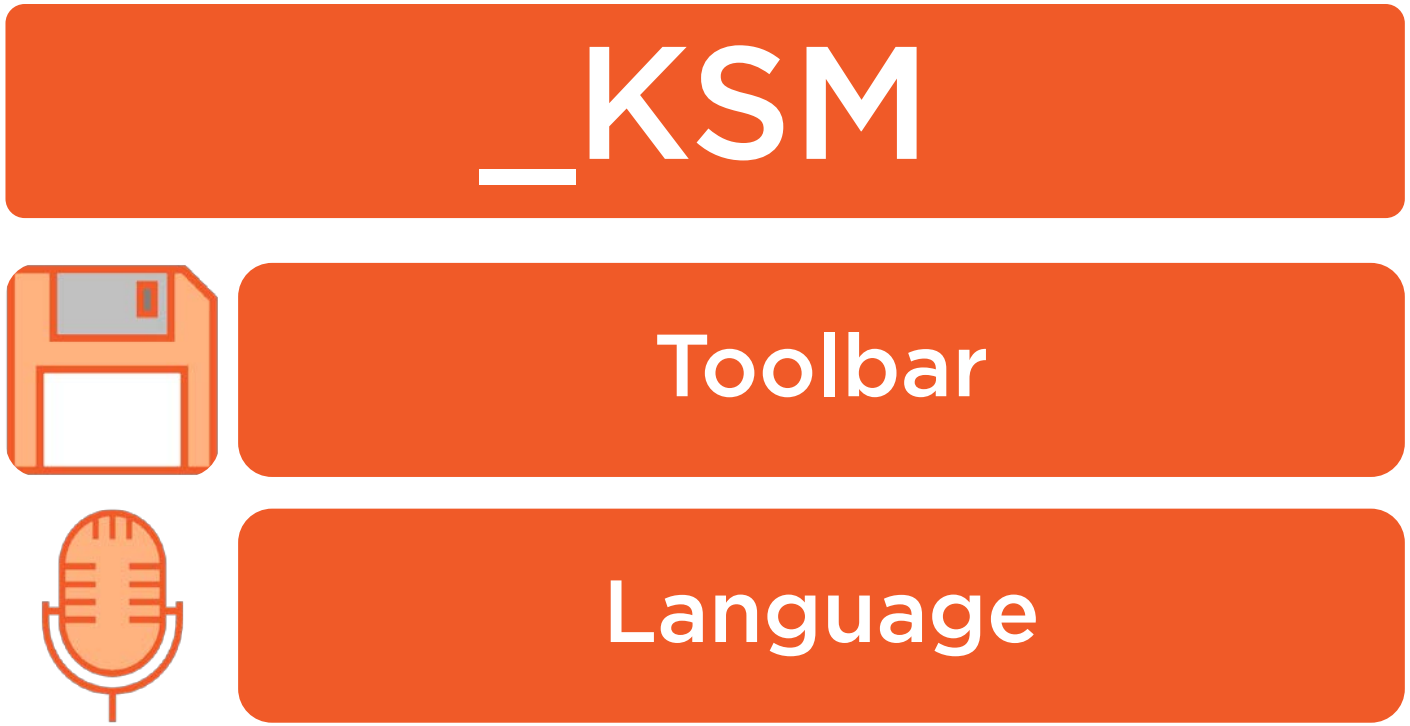


Nested dependencies require separate entries

Dependencies



_KSM Global Object



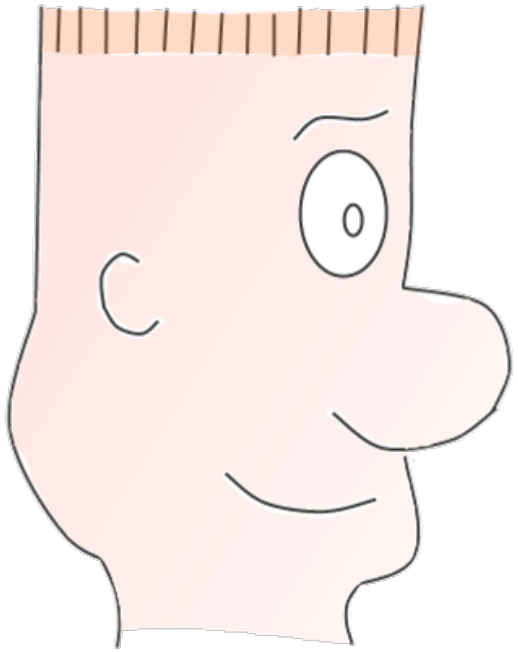
RequireJS and Legacy Libraries

Essential information

Online samples fall short

Configuration Sample

```
requirejs.config({
    shim: {
        'backbone': {
            deps: ['underscore', 'jquery'],
            exports: 'Backbone'
        },
        'underscore': {
            exports: '_'
        },
        'foo': {
            deps: ['bar'],
            exports: 'Foo',
            init: function (bar) {
                return this.Foo.noConflict();
            }
        }
    }
});
```



I read the comments and things still aren't clear

Iterative changes

Understanding “shim” properties

Need visible results

- Errors offer immediate feedback
- Debug console could be used
- Make changes to web page

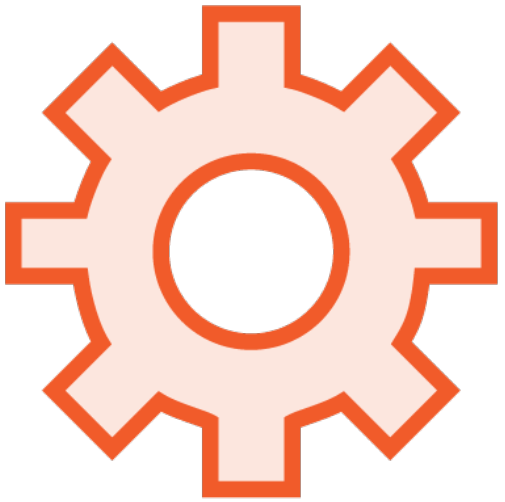
Starting Simple

Verify jQuery is loaded

jQuery Is Available

Loaded on global namespace

“arguments” array



Available in all functions

Determine if jQuery is passed as parameter

Not for production code

Double Negation

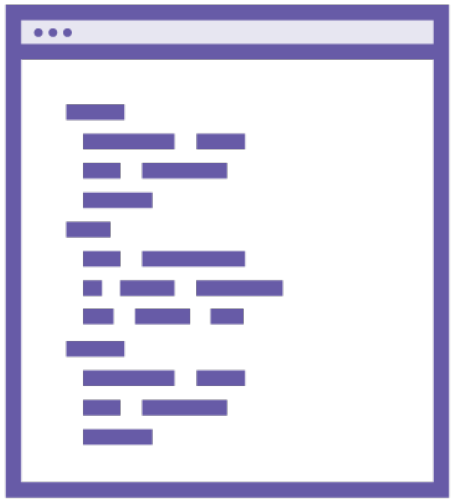
```
!!(arguments[0]))

...

!(undefined) -> TRUE

!(some value) -> FALSE

!!(undefined) -> FALSE

!!(some value) -> TRUE
```

Using Double Negation

Produces desired result for undefined and null values

Easier to code than an “if” block

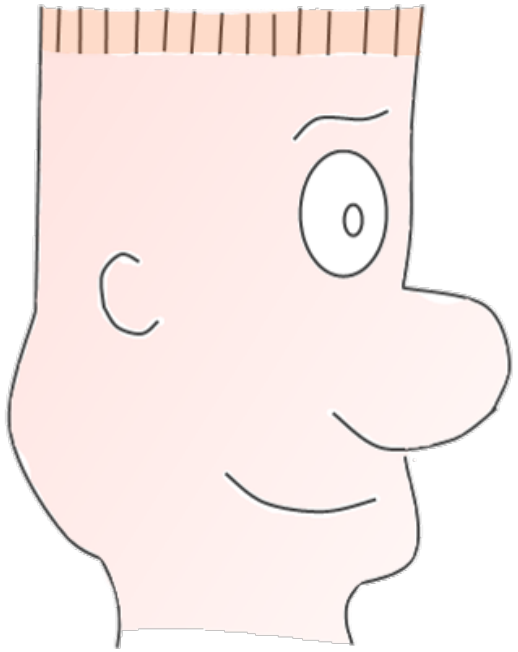
Add AMD Module



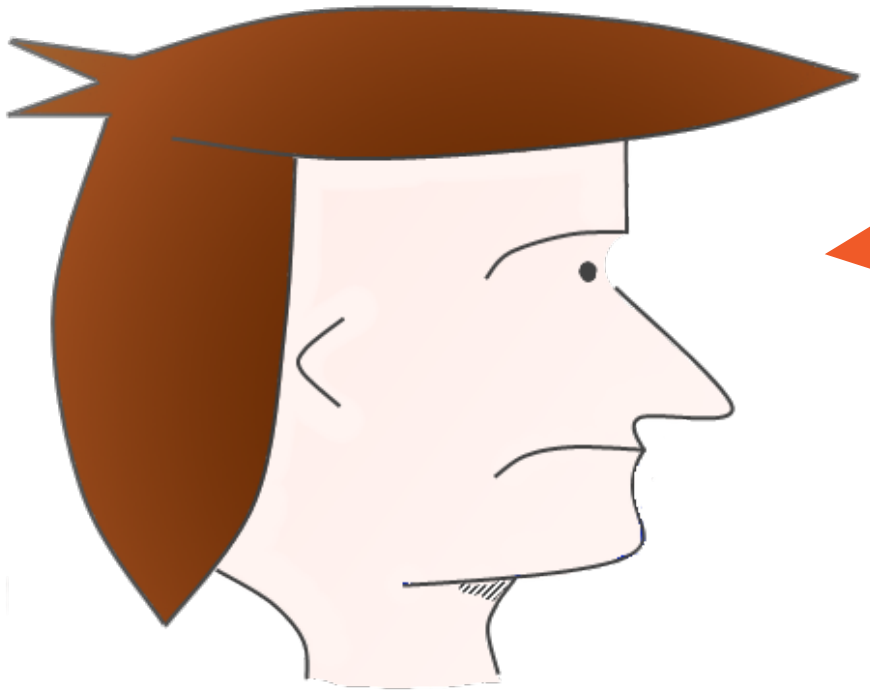
Use footer module as sample

Another Pop Quiz

Will the footer module produce an element in the “arguments” array?



RequireJS will probably return a pointer to the footer module



We? Don't you mean YOU are messing up?

jQuery Is Special

Written to behave differently in AMD environment

Library reference is returned to callback function

Length of “arguments”

“arguments” array length matches dependency array length

Placeholder for reference but value is undefined

RequireJS Callback Parameters

Language library returned no value

Footer module returned no value

“arguments” array contained undefined references

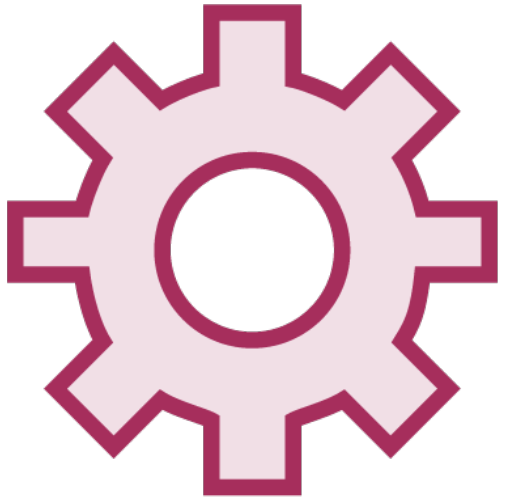
RequireJS had nothing to return after loading

jQuery and RequireJS

Specifically designed to return a value to module loader

Legacy Library Values

Directly reference values on global namespace



Return reference value to callback function

“exports” Property

Only used for legacy libraries

Used to specify a callback reference

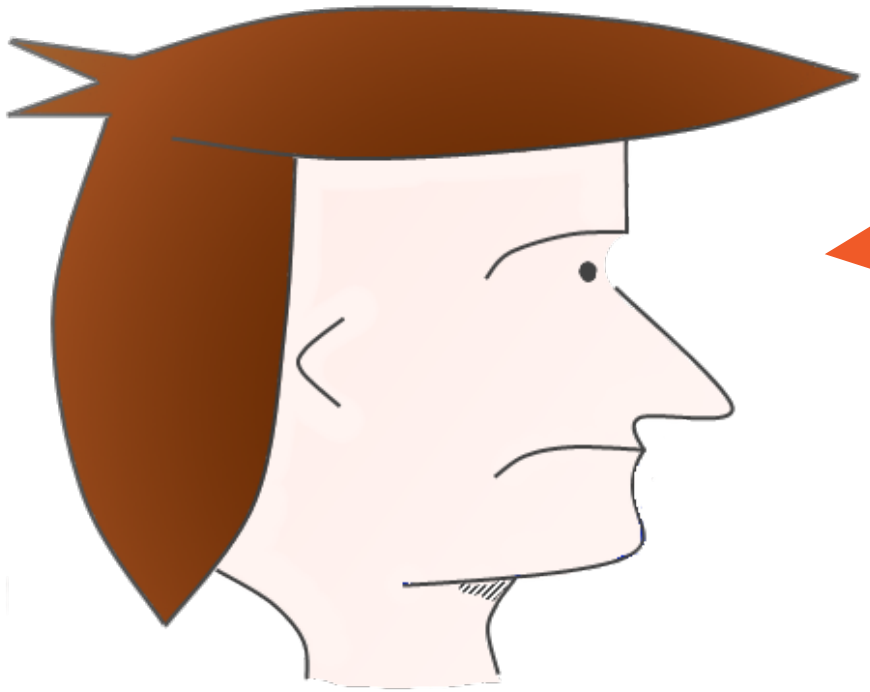
Without it, callback reference is undefined

_KSM.Language Object

_KSM object resides on global namespace

Callback parameter for convenience

Still exists after callback function terminates



Online samples imply “exports” is required for legacy libraries

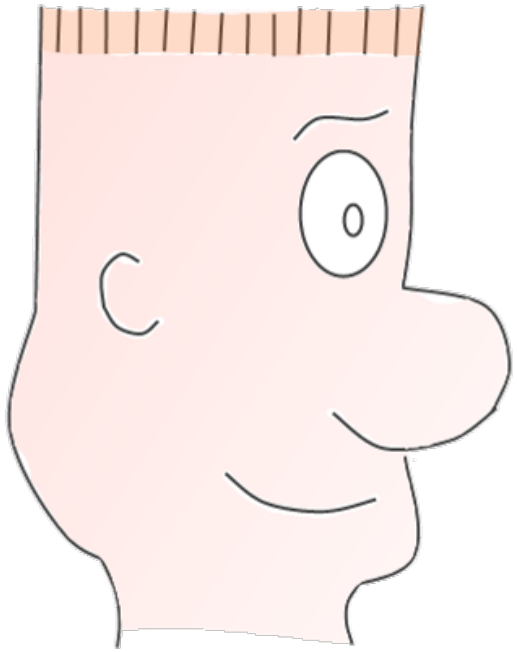
The purpose of the “exports” property is to tell RequireJS what portion of a legacy library to return to a callback function

Base object not the only option

- We used _KSM.Language

Makes sense for most libraries

- jQuery ($)
- Backbone (backbone)
- Underscore (_)



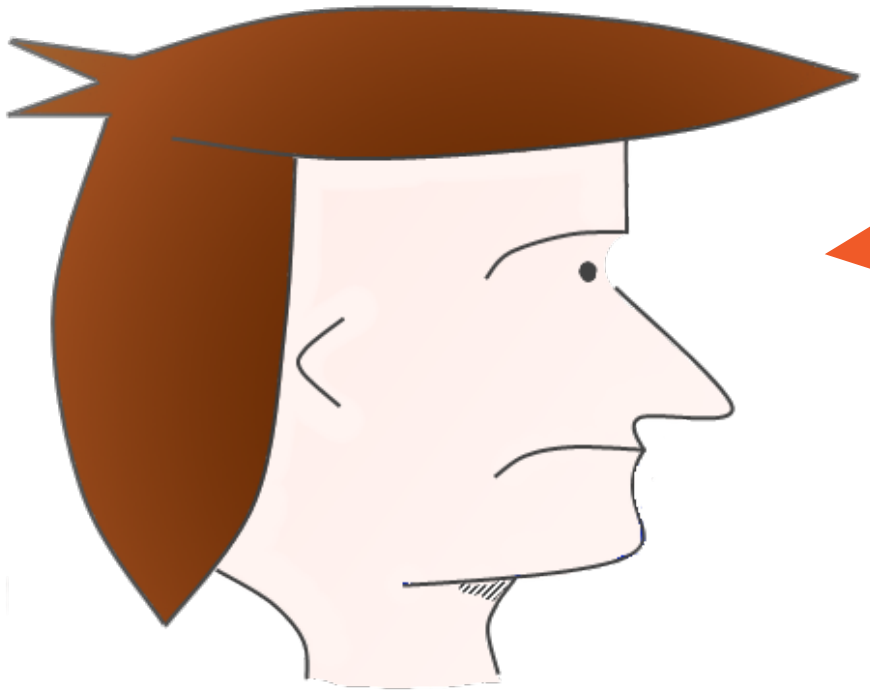
Is it lunchtime, yet? My head is starting to hurt.

“init” Function

Purpose of “init” Function

Perform maintenance logic

Execute before callback function



What kind of maintenance?

Possible Use for jQuery

Extend jQuery functionality

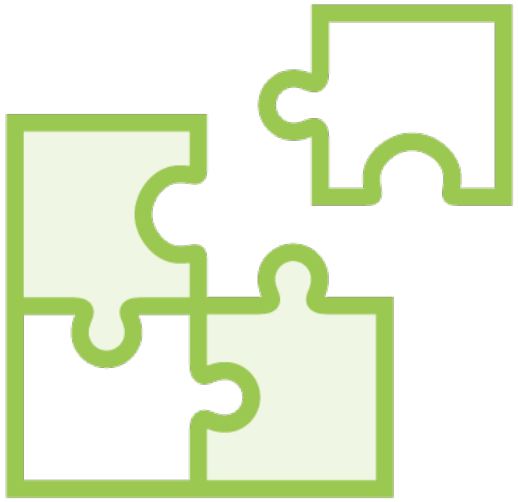
Add selectors, plugins, and more

"init" function is perfect fit

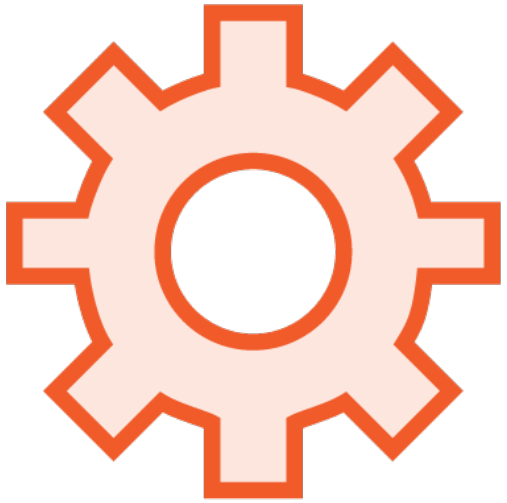
Features added prior to use

Single execution guaranteed

Other Libraries, Too



Third-Party



Custom

Think “init”

“this” in “init”

“this” is a reference to the global object

In a browser, “window” will be referenced

“init” Return Value

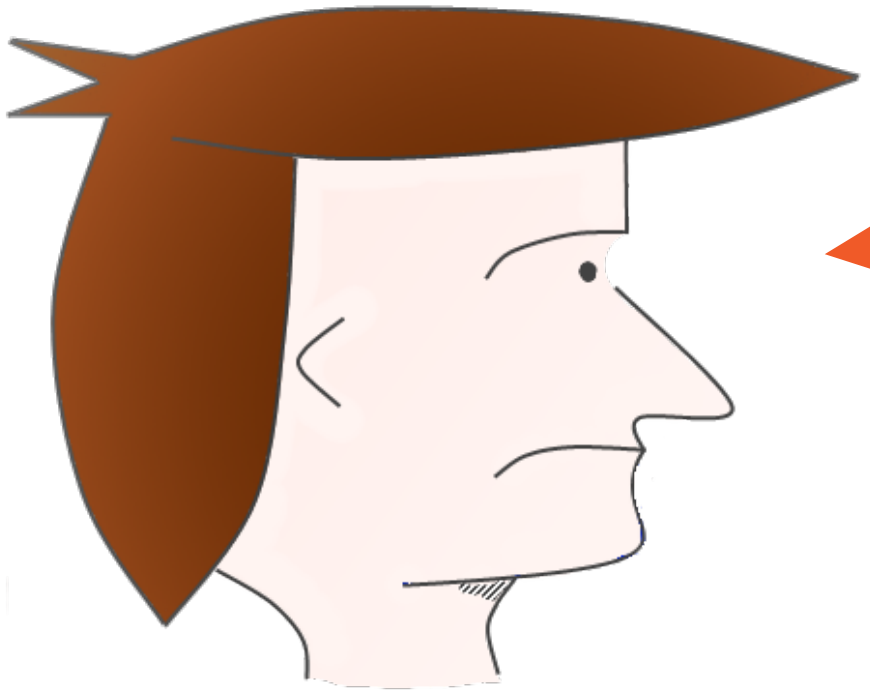
Checks presence of “init” function

Return value comes from “init” not “exports”

“init” supersedes “export”

“init” eliminates the use of “exports”

The “exports” property will not be used when an “init” function is in place



Why don't the return values match?

Loading Legacy Libraries

RequireJS can load legacy JavaScript

Legacy libraries need not be converted to modules

Legacy dependencies are defined in the “shim” property

RequireJS doesn’t automatically return a value

“shim” Property

Only used for legacy libraries

Used to define dependencies

Define callback reference with “exports”

“exports” Property

Defines portion of library to return to callback function

Usually the root object – but not always

Ignored when “init” is present

“init” Function

Useful for maintenance prior to execution

Return value supersedes “exports” property

Absence of return value still supersedes “exports” property

Covered only a few “confusing” options

Bundling and optimization possible

Learn more

- http://RequireJS.org
- http://Pluralsight.com

Summary

Startup code and HTML page

Better project structure

Loading legacy libraries

“shim” property