Require.js Webinar - Documentation

Table of Contents

1. Introduction

This webinar concentrates on the most popular JavaScript file and module loader, called require.js.
The sample application bases on technologies that were introduced in previous webinars:

- ASP.NET MVC
- ASP.NET WebAPI
- Knockout
- jQuery
- various small libraries like Cufon or Date.js

All these technologies need some glue to fit together. Require.js is exactly that glue. Using a modular script loader like require.js will improve the speed and quality of your code, too.

2. Modules

For our context a modules is structure used to encapsulate methods and attributes to avoid polluting the global namespace.

We will use the revealing module pattern because its widely adopted and very easy to understand. JavaScript doesn't have special syntax for namespaces or packages, but the module pattern allows us to write decoupled
pieces of code, which can be treated as black boxes. The module pattern is a combination of several patterns

- ~~Namespaces (not required here)~~
- Immediate functions
- Private and public members
- Declaring dependencies

During the webinar Johannes will explain all of these topics in detail, but for now this snipped shows a typical module.

```
var myModule = function () {

    var _name = "Johannes";

    function greetings() {
        console.log("Hello " + _name);
    }

    function setName(name) {
        _name = name;
    }

    return {
        setName: setName,
        greetings: greetings
    };
}();
```

3. Dependencies

For our context a dependency is a piece of JavaScript code on which another piece of JavaScript somehow depends on. The existence of such a dependency is therefore required to get the complete system running. In most cases a method or value of such a dependency is going to be accessed. In other cases there might be no direct access to the required piece of software, but there could be a hidden dependency where the systems state is changed as required.

The above module might have a dependency to jQuery. A simple way to make this dependency visible is the following construct.

```
var myModule = function ($) {

    var _name = "Johannes";

    function greetings() {
        $('#output').text("Hello " + _name);
    }

    return {
        greetings: greetings
    };
}(window.jQuery);
```

4. Dependencies Management

We have seen a naive way to handle dependencies. But things are getting complicated in practice.
Working with dependencies means loading of JavaScript code...

- ... in the right order.
- ... at the right time.
- ... at the right speed.

These requirements can't be resolved with simple script includes like:
`<script src="myscript.js"></script>`
It is not clear which other files need to be loaded in before. It's also not clear when this file would be required. It might be totally useless, nobody knows. And last but not least, this file will slow down the applications first run. Usually an application is ready to start, when the browser fires `DOMContentLoaded`. But DOMContentLoaded waits for the full HTML and scripts, and then triggers. If the file has no interaction with the DOM, this waiting time is useless.

Require.js takes a different approach to script loading than traditional <script> tags. Its goal is to encourage modular code. Require.js is a JavaScript file and module loader, capable of loading modules defined in the AMD (Asynchronous Module Definition) format and their dependencies. The asynchronous design makes AMD well suited for a browser environment. Today it's embraced by projects including Dojo, MooTools, Firebug or jQuery. The format is easy to learn since it only has to keywords: `define` and `require`.

Therefore require.js is easy to introduce to a project that is already using a module pattern. Only small changes on our module are necessary:

```
define(['jquery'], function ($) {

    var _name = "Johannes";

    function greetings() {
        $('#output').text("Hello " + _name);
    }

    return {
        greetings: greetings
    };
});
```

The file should get the file name "myModule.js".

5. Loading of dependencies

It is considered as best-practice to place to the bottom of a page. But just JavaScript code at the bottom of the page won't make things that much better. That’s why we will should load our dependencies with require.js.

The module is defined but nobody uses it. Let's create a counterpart. The following snipped loads the module "myModule". By convention, the name of the module (the so-called **module ID**) should be equal to the file without the .js file ending.

```
require(['myModule'], function(m) {
    m.greetings();
});
```

Here we load the module "myModule" and decide to name the corresponding parameter in the anonymous function to a shorter version "m". We then call the method "greetings" of the "myModule" module.

6. Configuration

In a perfect world wouldn't need setup files. But often we have to deal with version numbers in files, obscured paths or files that don’t implement the AMD pattern. For all these proposes require.js can be configured. It’s a good idea to put that configuration into a new file, e.g. called require.config.js:

```
requirejs.config({
    baseUrl: "Scripts",
    paths: {
        'jquery': 'jquery-2.0.3',
        'knockout': 'knockout-2.3.0'
    },
    shim: {
        'knockout': { deps: ['jquery'] }
    }
});
```

Here we define paths and file names if the file does not match the convention. There is also a "shim" defined, that indicates that the module "knockout" is dependent upon "jquery". Require.js will make sure, that jQuery will be loaded before Knockout. Shims are also often used to point to a global variable that and AMD-ignorant script will create. (via exports)

7. More

There are several more things to discover.
You should start by downloading the sources of the demo app.

» Download Demo-Code (.zip)

A great book is "JavaScript Patterns" from Stoyan Stefanov. It explains in detail the required patterns for a solid JavaScript-driven website!

Mit Coding Patterns und Entwurfsmustern zu besseren JavaScript-Anwendungen
JavaScript Patterns
Stoyan Stefanov
Deutsche Übersetzung von Thomas Demmig
O'REILLY
YAHOO! PRESS