
text

A RequireJS/AMD loader plugin for loading text resources.

Known to work in RequireJS, but should work in other AMD loaders that support the same loader plugin API.

Docs

See the RequireJS API text section.

Latest release

The latest release is always available from the “latest” tag.

It can also be installed using volo:

```
1 volo add requirejs/text
```

If using npm:

```
1 npm install requirejs/text
```

Usage

It is nice to build HTML using regular HTML tags, instead of building up DOM structures in script. However, there is no good way to embed HTML in a JavaScript file. The best that can be done is using a string of HTML, but that can be hard to manage, particularly for multi-line HTML.

The text.js AMD loader plugin can help with this issue. It will automatically be loaded if the text! prefix is used for a dependency. Download the plugin and put it in the app’s baseUrl directory (or use the paths config to place it in other areas).

You can specify a text file resource as a dependency like so:

```
1 require(["some/module", "text!some/module.html", "text!some/module.css"  
2         ],  
3         function(module, html, css) {  
4             //the html variable will be the text  
5             //of the some/module.html file  
6             //the css variable will be the text  
7             //of the some/module.css file.  
8         }  
9     );
```

Notice the .html and .css suffixes to specify the extension of the file. The “some/module” part of the path will be resolved according to normal module name resolution: it will use the **baseUrl** and **paths** configuration options to map that name to a path.

For HTML/XML/SVG files, there is another option. You can pass !strip, which strips XML declarations so that external SVG and XML documents can be added to a document without worry. Also, if the string is an HTML document, only the part inside the body tag is returned. Example:

```
1 require(["text!some/module.html!strip"],
2     function(html) {
3         //the html variable will be the text of the
4         //some/module.html file, but only the part
5         //inside the body tag.
6     }
7 );
```

The text files are loaded via asynchronous XMLHttpRequest (XHR) calls, so you can only fetch files from the same domain as the web page (see **XHR restrictions** below).

However, the RequireJS optimizer will inline any text! references with the actual text file contents into the modules, so after a build, the modules that have text! dependencies can be used from other domains.

Configuration

XHR restrictions

The text plugin works by using XMLHttpRequest (XHR) to fetch the text for the resources it handles.

However, XHR calls have some restrictions, due to browser/web security policies:

- 1) Many browsers do not allow file:// access to just any file. You are better off serving the application from a local web server than using local file:// URLs. You will likely run into trouble otherwise.
- 2) There are restrictions for using XHR to access files on another web domain. While CORS can help enable the server for cross-domain access, doing so must be done with care (in particular if you also host an API from that domain), and not all browsers support CORS.

So if the text plugin determines that the request for the resource is on another domain, it will try to access a “.js” version of the resource by using a script tag. Script tag GET requests are allowed across domains. The .js version of the resource should just be a script with a define() call in it that returns a string for the module value.

Example: if the resource is 'text!example.html' and that resolves to a path on another web domain, the text plugin will do a script tag load for 'example.html.js'.

The requirejs optimizer will generate these '.js' versions of the text resources if you set this in the build profile:

```
1 optimizeAllPluginResources: true
```

In some cases, you may want the text plugin to not try the .js resource, maybe because you have configured CORS on the other server, and you know that only browsers that support CORS will be used. In that case you can use the module config (requires RequireJS 2+) to override some of the basic logic the plugin uses to determine if the .js file should be requested:

```
1 requirejs.config({
2   config: {
3     text: {
4       useXhr: function (url, protocol, hostname, port) {
5         //Override function for determining if XHR should be
6           used.
7         //url: the URL being requested
8         //protocol: protocol of page text.js is running on
9         //hostname: hostname of page text.js is running on
10        //port: port of page text.js is running on
11        //Use protocol, hostname, and port to compare against
12        the url
13        //being requested.
14        //Return true or false. true means "use xhr", false
15        means
16        // "fetch the .js version of this resource".
17      }
18    }
19  });
```

Custom XHR hooks

There may be cases where you might want to provide the XHR object to use in the request, or you may just want to add some custom headers to the XHR object used to make the request. You can use the following hooks:

```
1 requirejs.config({
2   config: {
3     text: {
4       onXhr: function (xhr, url) {
5         //Called after the XHR has been created and after the
6         //xhr.open() call, but before the xhr.send() call.
7         //Useful time to set headers.
```

```
8         //xhr: the xhr object
9         //url: the url that is being used with the xhr object.
10    },
11    createXhr: function () {
12        //Overrides the creation of the XHR object. Return an
13        //XHR
14        //object from this function.
15        //Available in text.js 2.0.1 or later.
16    },
17    onXhrComplete: function (xhr, url) {
18        //Called whenever an XHR has completed its work. Useful
19        //if browser-specific xhr cleanup needs to be done.
20    }
21 }
22 });
```

Forcing the environment implementation

The text plugin tries to detect what environment it is available for loading text resources, Node, XMLHttpRequest (XHR) or Rhino, but sometimes the Node or Rhino environment may have loaded a library that introduces an XHR implementation. You can force the environment implementation to use by passing an “env” module config to the plugin:

```
1 requirejs.config({
2     config: {
3         text: {
4             //Valid values are 'node', 'xhr', or 'rhino'
5             env: 'rhino'
6         }
7     }
8 });
```

License

MIT

Code of Conduct

jQuery Foundation Code of Conduct.

Where are the tests?

They are in the requirejs and r.js repos.

History

This plugin was in the requirejs repo up until the requirejs 2.0 release.