LIFERAY AMD MODULE LOADER

WHAT IS THE LIFERAY AMD MODULE LOADER?

- The `Liferay AMD Module Loader` is a JavaScript file and module loader.
- It can be found here: *https://www.npmjs.com/package/liferay-amd-loader*

WHAT IS A JAVASCRIPT MODULE?

- JavaScript modules are a way to encapsulate a piece of code into a useful unit that exports its capability/value.
- This makes it easy for other modules to explicitly require this piece of code.
- Structuring an application this way makes it easier to see the broader scope, easier to find what you're looking for, and keeps things related.

WHAT'S THE PURPOSE OF A MODULE LOADER?

- A normal webpage usually loads JavaScript files via HTML `script` tags.
- That's fine for small websites, but when developing large-scale web applications, a better way to organize and load files is needed.
- A module loader allows an application to load dependencies easily by just specifying a string that identifies the module name.

HOW DO YOU DEFINE A MODULE?

- The Liferay AMD Module loader works with JavaScript modules that are in the AMD format. Here is a basic example of the definition of an AMD module:

```
define('aui-dialog', ['aui-node', 'aui-plugin-base'], function(node,
pluginBase) {
    return {
        log: function(text) {
            console.log('module aui-dialog: ' + text);
        }
    };
});
```

- You may specify that the module should be loaded on triggering some other module and only if some condition is being met.

CONDITIONAL LOADING

- This module should be loaded automatically if you request `aui-test` module, but only if some condition is being met.

```
define('aui-dialog', ['aui-node', 'aui-plugin-base'], function(node,
pluginBase) {
    return {
        log: function(text) {
            console.log('module aui-dialog: ' + text);
        }
    };
}, {
    condition: {
        trigger: 'aui-test',
        test: function() {
            var el = document.createElement('input');

            return ('placeholder' in el);
        }
    },
    path: 'aui-dialog.js'
});
```

HOW DO YOU LOAD A MODULE?

- Loading a module is as easy as passing the module name to the `require` method.

```
require('aui-dialog', function(base, test) {
    // your code here
}, function(error) {
    console.error(error);
});
```

MAPPING MODULE NAMES

- You can map module names to specific versions or other naming conventions.

```
__CONFIG__.maps = {
    'liferay': 'liferay@1.0.0',
    'liferay2': 'liferay@1.0.0'
};
```

- Mapping a module will change its name in order to match the value, specified in the map.

```
require('liferay/html/js/autocomplete'...)
```

- Under the hood, this will be the same as:

```
require('liferay@1.0.0/html/js/autocomplete'...)
```

HOW IS THE LOADER USED IN LIFERAY 7?

- Tools, like the `Liferay AMD Module Config Generator`, have been integrated into the platform to make it easy for developers to create and load modules.
- An outline of the process is as follows:
    1. The tool scans your code and looks for amd modules `define(...)` statements.
    2. It will then name the module, if it is not named already.
    3. It takes note of that information, as well as the listed dependencies, and also any other configurations specified.

CONFIG.JSON

- Next, the tool creates a config.json file that may look something like this:

```
{
    "frontend-js-web@1.0.0/html/js/parser": {
        "dependencies": []
    },
    "frontend-js-web@1.0.0/html/js/list-display": {
        "dependencies": ["exports"]
    },
    "frontend-js-web@1.0.0/html/js/autocomplete": {
        "dependencies": ["exports", "./parser", "./list-display"]
    }
}
```

- You can see another example here: `modules/apps/foundation/frontend-js/frontend-js-metal-web/.task-cache/config.json`.
- This configuration object tells the loader which modules are available, where they are, and what dependencies they will require.

Notes: