Adapt Authoring Tool

Module Loader Changes 01/20

03.01.2020

prepared by Tom Taylor

Contents

Main headlines

- Multi-stage boot process simplified
- Module/utility abstractions unified
- Utility abstraction replaced with `coreApiName` feature
- Adapt config moved to separate file

Multi-stage boot process simplified

Old process: developer must split initialisation code into the appropriate Module functions (constructor, preload, boot).

New process: any initialisation code is called from the constructor, and the developer must simply call setReady to signify that the Module has initialised.

Module/utility abstractions unified

AbstractUtility class has been removed completely, with all modules now inheriting from AbstractModule. Utility functionality has been replaced with a '**core API**' feature (see below).

Utility abstraction replaced with `coreApiName` feature

In some cases, you may have a module which should be considered 'core'. Examples of these are `logger`, `jsonschema` and `config`. The two key differences between core and non-core modules are:

- Core modules are accessible directly from the app object (e.g. `app.config`)
- All core modules are initialised prior to non-core modules

Should my module be core?

- Is my module used by many other modules?
- Is there a need to access my module directly from the app object?
- Does my module need to initialise before non-core modules?

If the answer is yes to any of the above, then your module should probably be a core module. To mark your module as a 'core' module, you simply need to add the following to the `adapt.json` file (note: the value you use will become the variable name on the `app` object):

```
“coreApiName”: “myModule”
```

It's important to avoid circular dependencies between core modules during the initialisation process, as this will stop the app from starting. The preferable solution is to *not* rely on any other core modules during the initialisation of your own module. In the cases that this isn't possible, you must make sure to call `setReady` at a time to avoid any circular 'waiting' for other modules.

Example:

- `config` requires `jsonschema` to validate all config schemas
- `jsonschema` requires `config` to enable user-specified overrides

Solution:

***Note**: not necessarily the best solution*

`config` requires `jsonschema` to start correctly, as the module has been designed to fail and stop app execution early if any config is invalid. User-specified overrides for `jsonschema` module (set in config file) are not necessary for initialisation of the module/app, so this code in `jsonschema` has been moved to after `setReady` is called.

Adapt config moved to separate file

All Adapt-specific configuration was previously set in each module's `package.json` file. To make it easier to programmatically determine Adapt modules from non-Adapt modules, this has now been moved into a separate `adapt.json` file.

Resources

- **Working prototype**: https://github.com/taylortom/moduleloadtest