What is Validatem?

> This article is derived from the documentation at https://www.npmjs.com/package/@validatem/core.

The last validation library you'll ever need.

- Does **every kind of validation**, and does it well: it doesn't matter whether you're validating function arguments, form data, JSON request bodies, configuration files, or whatever else. As long as it's structured data of some sort, Validatem can deal with it.
- Supports the notion of **virtual properties** in validation errors, which means that even if your data *isn't* already structured data (eg. an encoded string of some sort), you can bring your own parser, and have it integrate cleanly.
- **Easy to read**; both the code that *uses* Validatem, and the validation error messages that it produces! Your validation code doubles as in-code format documentation, and users get clear feedback about what's wrong.
- Fully **composable**: it's trivial to use third-party validators, or to write your own (reusable!) validators, whether fully custom or made up of a few other validators chained together.
- Supports **value transformation**, which means that you can even encode things like "this value defaults to X" or "when this value is a number, it will be wrapped like so" in your validation code; this can save you a bunch of boilerplate, and makes your validation code *even more complete* as format documentation.
- Validatem has a **small and modular core**, and combined with its composability, this means you won't pull any more code into your project than is strictly necessary to make your validators work! This is also an important part of making Validatem suitable for use in libraries, eg. for argument validation.
- Many **off-the-shelf validators** are already available! You can find the full list here.
- Extensively **documented**, with clear documentation on what is considered valid, and what is not. Likewise, the plumbing libraries that you can use to write your own validators and combinators, are also well-documented.

While Validatem is suitable for any sort of validation, this unique combination of features and design choices makes it *especially* useful for validating arguments in the public API of libraries, unlike other validation libraries!

For example, you might write something like the following (from the `icssify` library):

```
module.exports = function (browserify, options) {
	validateArguments(arguments, {
		browserify: required,
		options: allowExtraProperties({
			mode: oneOf([ "local", "global" ]),
			before: arrayOf([ required, isPostcssPlugin ]),
			after: arrayOf([ required, isPostcssPlugin ]),
			extensions: arrayOf([ required, isString ])
		})
	});

	// Implementation code goes here ...
};
```

And calling it like so:

```
icssify(undefined, {
	mode: "nonExistentMode",
	before: [ NaN ],
	unspecifiedButAllowedOption: true
})
```

... would then produce an error like this:

```
ValidationError: One or more validation errors occurred:
 - At browserify: Required value is missing
 - At options -> mode: Must be one of: 'local', 'global'
 - At options -> before -> 0: Must be a PostCSS plugin
```

Revision #2
Created 10 December 2024 23:35:13 by joepie91
Updated 10 December 2024 23:38:24 by joepie91