NAME
>   validate.inplace – validate elements in a data structure during assignment

SYNOPSIS
```
-- load packages
va = require('validate.args')
vi = require('validate.inplace' )

-- define a validation specification (see docs for validate.args)
spec = { foo = { type = 'posint' } }

-- construct the inplace validation object
vio = vi:new( 'test', spec, va:new() )

-- create a more user friendly table to play with
test = vio:proxy()

-- assign something that's legal.
test.foo = 3

-- assign something that's not legal; this'll throw an error
test.foo = -3

-- use the value
if 3 == test.foo then print ("3!" ) end

-- create an independent standard Lua data structure
copy = vio:copy()
```

DESCRIPTION
>   **validate.inplace** makes it easy to provide instantaneous feedback to users when they assign an incorrect value to an element in a data structure.

>   Traditional validation of data occurs after a data structure has been created and passed into a validation routine. At that point the user can only be notified of which element in the data structure is invalid. It'd be much more useful if the user was notified of where in their *code* the error was made.

>   This class creates a proxy data structure which allows the validation of elements upon assignment to them.

>   Validation is performed by **validate.args**.

>   **Usage**
>   1.   Create a validation specification table (see the **validate.args** docs). As this is a validation of a table, only named elements can be validated, so the specification table will look like

>            ```
>            spec = { name_of_elem1 = { spec for elem1 },
>                     name_of_elem2 = { sepc for elem2 } }
>            ```

>   2.   Create a **validate.args** validation object and customize it as necessary:

>            ```
>            vao = require( 'validate.args'):new()
>            ```

>   3.   Create the **validate.inplace** object:

>            ```
>            vio = vi:new( 'config', spec, vao )
>            ```

>   4.   Retrieve and expose the proxy data structure:

>            ```
>            config = vio:proxy();
>            ```

>   At this point `config` is a Lua table which can be assigned to and read from. However, none of the Lua functions used to traverse a table (such as **pair**, **next**, etc) will be of much use. To use those you'll need to

make a copy of the table (using the *copy()* method) and traverse the copy.

Methods

new

```
obj = vi:new( name, spec, vao )
```

Create a **validate.inplace** object. This object administers the storage and validation of the data, as well as the proxy data structure presented to the user.

The following parameters are required:

name

The name assigned to the root of the data structure, used in error messages. Typically this is the name of the table to which the top level proxy table will be assigned (via the *proxy()* method):

```
vio = vi:new( 'config', spec, va:new() )
config = vio:proxy()
```

spec

This is a **validate.args** validation specification table. Its structure must follow the "named argument layout" structure as documented in the **validate.args** docs.

vao This is a **validate.args** object. It is used along with the specification to validate elements.

proxy

```
table = vio:proxy()
```

This returns a table to which data may be written to and read from. As it is a proxy, attempts to traverse it will be unfulfilling.

copy

```
copy = vio:copy()
```

Create a copy of the data structure managed by the **validate.inplace** object. Unlike the structure returned by the *proxy()* method, the structure returned by *copy()* may be traversed using the standard Lua functions. Changes to it will not be reflected in the data stored in the **validate.inplace** object (nor will changes in the object change this data structure).

LIMITATIONS

Unsupported validate.args features

validate.inplace cannot handle specifications which use **vtable** functions or mutating validation specifications.

Performance

This class is not built for speed. It imposes significant overhead when both setting and retrieving elements within a data structure. It was designed for validation of configuration data which is typically done once.

Some of things that happen under the hood:

1. **validate.args** is invoked whenever a value is set

2. The data structure uses proxy tables to track accesses to the data.

3. Assignment of a nested data structure to an element requires a recursive descent through the nested table if some or all levels of the nested structure are to be validated.

To improve read performance, use the *copy()* method to create a non-proxied version of the data structure. It will be detached from the proxied version (so changes to either will not be reflected in the other).

AUTHOR

Diab Jerius, <djerius@cfa.harvard.edu>

COPYRIGHT AND LICENSE

Copyright (C) 2011 by the Smithsonian Astrophysical Observatory

This software is released under the GNU General Public License. You may find a copy at <http://www.fsf.org/copyleft/gpl.html>.