class Tinky::Hash

Hash configuration for use with Tinky

Table of Contents


```
class Hash::Tinky { ... }
```

Synopsis

```
use Timky::Hash;

# define a class to be able to define methods for the transitions
class MyStateEngine is Tinky::Hash {

  # initialize state engine uning from-hash method
  submethod BUILD ( ) {

    self.from-hash(
      :config( {
          :states([< a z>]),
          :transitions( {
              :az( { :from<a>, :to<z>}),
              :za( { :from<z>, :to<a>}),
            }
          ),
          :workflow( { :name<wf5>, :initial-state<a>}),
```

```
            :taps( {
                :states( {
                    :a( { :leave<leave-a>}),
                    :z( { :enter<enter-z>})
                  }
                ),
              }
            ),
          }
        )
      );
    }
    # call when leaving state a
    method leave-a ( $object ) {
      say "Tr 2 left  a in '$object.^name()'";
    }

    # call when entering state z
    method enter-z ( $object ) {
      say "Tr 2 enter z in '$object.^name()'";
    }
}

# instantiate
my MyStateEngine $th .= new;

# use workflow
$th.workflow('wf5');

# go to state z. this runs the methods leave-a and enter-z.
$th.go-state('z');
```

Description

To understand this module it is wise to also read the documentation about Tinky and day 18 2016 of the perl6 advent calendar.

I was triggered writing Tinky::Hash by the Tinky::JSON module to define a data structure instead of using the commands directly. It makes for a cleaner setup.

A few things are added here compared to the JSON implementation. Using a class which inherits the Tinky::Hash class it is possible to call methods defined by their name in the config. Furthermore, besides that a method can be called upon all transition events, it is possible to call a method on one specific transition.

Methods

new

from-hash

workflow

go-state