🦋 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 all from a single class where it is needed. Also it can be stored in other formats besides JSON, Examples are YAML and TOML.

A few things are added here compared to the Tinky::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.

Because of the way this class stores its data, the workflows are still usable from other classes which inherit the Tinky::Hash.

Methods

new

```
submethod BUILD ( Hash :$config )
```

Instantiate class. When config is given, it will call from-hash with it.

from-hash

```
method from-hash ( Hash:D :$config )
```

Reads the configuration and uses the methods from Tinky to define states, transitions, workflow and also defines the taps for the events of transitions, leaving or entering a state.

Configuration structure

The top level looks like the following;

```
:config( {
    :states( … ),
    :transitions( … ),
    :workflow( … ),
    :taps( … )
  }
)
```

- states

States is used to specify all the states used in the workflow. It is an array of names for the states. These are used to refer to in transitions, workflow and taps.

```
:config( { :states([<locked opened>]), … })
```

- transitions

Transitions describe the connections between the states. The names of defined transitions are used in taps. With the states mentioned above;

```
:config( {
    …
    :transitions( {
        :openit( { :from<locked>, :to<opened>}),
        :lockit( { :from<opened>, :to<locked>})
      }
    ),
    …
  }
)
```

- workflow

A workflow binds everything together specifying a name for the flow and an initial state. E.g. using the defined config above

```
:config( {
    …
    :workflow( { :name(resource-lock), :initial-state<locked>}),
    …
  }
)
```

- taps

Supplies are used to handle events such as entering or leaving a state or on transitions. By creating a tap on a supply the object gets informed by these events. Tinky allows you to specify two types of taps. These are the taps on events on entering and leaving a state and a tap to get all transition events. With Tinky::Hash you can also specify taps on a specific transition. E.g.

```
:config( {
    …
    :taps( {
        :states( {
            :locked( { :leave<make-log>}),
          }
        ),
      }
    )
  }
)
```

Here when a transition is made from *locked* to *opened*, the method make-log is called upon leaving the state *locked*. Methods used like this must be defined as follows;

```
method make-log ( Str :$state, EventType :$event )
```

workflow

```
method workflow ( Str:D $workflow-name )
```

go-state

```
method go-state ( Str:D $state-name )
```