package com.twitter.tweetypie

package hydrator

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.EditState

/\*\*

\* An EditHydrator hydrates a value of type `A`, with a hydration context of type `C`,

\* and produces a function that takes a value and context and returns an EditState[A, C]

\* (an EditState encapsulates a function that takes a value and returns a new ValueState).

\*

\* A series of EditHydrators of the same type may be run in parallel via

\* `EditHydrator.inParallel`.

\*/

class EditHydrator[A, C] private (val run: (A, C) => Stitch[EditState[A]]) {

/\*\*

\* Apply this hydrator to a value, producing an EditState.

\*/

def apply(a: A, ctx: C): Stitch[EditState[A]] = run(a, ctx)

/\*\*

\* Convert this EditHydrator to the equivalent ValueHydrator.

\*/

def toValueHydrator: ValueHydrator[A, C] =

ValueHydrator[A, C] { (a, ctx) => this.run(a, ctx).map(editState => editState.run(a)) }

/\*\*

\* Runs two EditHydrators in parallel.

\*/

def inParallelWith(next: EditHydrator[A, C]): EditHydrator[A, C] =

EditHydrator[A, C] { (x0, ctx) =>

Stitch.joinMap(run(x0, ctx), next.run(x0, ctx)) {

case (r1, r2) => r1.andThen(r2)

}

}

}

object EditHydrator {

/\*\*

\* Create an EditHydrator from a function that returns Stitch[EditState[A]].

\*/

def apply[A, C](f: (A, C) => Stitch[EditState[A]]): EditHydrator[A, C] =

new EditHydrator[A, C](f)

/\*\*

\* Creates a "passthrough" Edit:

\* Leaves A unchanged and produces empty HydrationState.

\*/

def unit[A, C]: EditHydrator[A, C] =

EditHydrator { (\_, \_) => Stitch.value(EditState.unit[A]) }

/\*\*

\* Runs several EditHydrators in parallel.

\*/

def inParallel[A, C](bs: EditHydrator[A, C]\*): EditHydrator[A, C] =

bs match {

case Seq(b) => b

case Seq(b1, b2) => b1.inParallelWith(b2)

case \_ => bs.reduceLeft(\_.inParallelWith(\_))

}

}