package com.twitter.servo.util

/\*\*

\* A collection of FunctionArrow factory functions.

\*/

object FunctionArrow {

def apply[A, B](f: A => B): FunctionArrow[A, B] = fromFunction(f)

/\*\*

\* Produce an FunctionArrow from a function `A => B`.

\*/

def fromFunction[A, B](f: A => B): FunctionArrow[A, B] =

new FunctionArrow[A, B] {

def apply(a: A): B = f(a)

}

/\*\*

\* Produces a FunctionArrow with no side-effects that simply returns its argument.

\*/

def identity[A]: FunctionArrow[A, A] = apply(Predef.identity[A])

/\*\*

\* Appends two FunctionArrows together.

\*

\* This forms a monoid with 'identity'.

\*/

def append[A, B, C](a: FunctionArrow[A, B], b: FunctionArrow[B, C]): FunctionArrow[A, C] =

a.andThen(b)

/\*\*

\* Produce an FunctionArrow that applies an Effect, returning the argument

\* value as-is.

\*/

def effect[A](effect: Effect[A]): FunctionArrow[A, A] = apply { a =>

effect(a); a

}

/\*\*

\* Produces an FunctionArrow that proxies to one of two others, depending on a

\* predicate.

\*/

def choose[A, B](

predicate: A => Boolean,

ifTrue: FunctionArrow[A, B],

ifFalse: FunctionArrow[A, B]

): FunctionArrow[A, B] =

apply { a: A =>

if (predicate(a)) ifTrue(a) else ifFalse(a)

}

/\*\*

\* Produces an FunctionArrow whose application is guarded by a predicate. `f` is

\* applied if the predicate returns true, otherwise the argument is simply

\* returned.

\*/

def onlyIf[A](predicate: A => Boolean, f: FunctionArrow[A, A]): FunctionArrow[A, A] =

choose(predicate, f, identity[A])

}

/\*\*

\* A function encapsulating a computation.

\*

\* Background on the Arrow abstraction:

\* http://en.wikipedia.org/wiki/Arrow\_(computer\_science)

\*/

trait FunctionArrow[-A, +B] extends (A => B) { self =>

/\*\*

\* Composes two FunctionArrows. Produces a new FunctionArrow that performs both in series.

\*/

def andThen[C](next: FunctionArrow[B, C]): FunctionArrow[A, C] =

new FunctionArrow[A, C] {

override def apply(a: A) = next.apply(self(a))

}

}