package com.twitter.servo.util

object Effect {

// a no-op effect

private[this] val \_unit = Effect[Any] { \_ =>

()

}

/\*\*

\* A "no-op" Effect. For any effect E, (E also unit) == (unit also E) == E.

\* Forms a monoid with `also`.

\*/

def unit[A]: Effect[A] = \_unit.asInstanceOf[Effect[A]]

/\*\*

\* Package a function as an Effect.

\*/

def apply[A](f: A => Unit): Effect[A] =

new Effect[A] {

override def apply(value: A) = f(value)

}

/\*\*

\* An effect that only applies to some values.

\*/

def fromPartial[A](f: PartialFunction[A, Unit]): Effect[A] =

Effect[A] { x =>

if (f.isDefinedAt(x)) f(x)

}

}

/\*\*

\* Perform an effect with the given value, without altering the result.

\*

\* Forms a monoid with Effect.unit as unit and `also` as the combining operation.

\*/

trait Effect[A] extends (A => Unit) { self =>

/\*\*

\* An identity function that executes this effect as a side-effect.

\*/

lazy val identity: A => A = { value =>

self(value); value

}

/\*\*

\* Combine effects, so that both effects are performed.

\* Forms a monoid with Effect.unit.

\*/

def also(next: Effect[A]): Effect[A] =

Effect[A](identity andThen next)

/\*\*

\* Convert an effect to an effect of a more general type by way

\* of an extraction function. (contravariant map)

\*/

def contramap[B](extract: B => A): Effect[B] =

Effect[B](extract andThen self)

/\*\*

\* Perform this effect only if the provided gate returns true.

\*/

@deprecated("Use enabledBy(() => Boolean)", "2.5.1")

def enabledBy(enabled: Gate[Unit]): Effect[A] =

enabledBy(() => enabled())

/\*\*

\* Perform this effect only if the provided gate returns true.

\*/

def enabledBy(enabled: () => Boolean): Effect[A] =

onlyIf { \_ =>

enabled()

}

/\*\*

\* Perform this effect only if the provided predicate returns true

\* for the input.

\*/

def onlyIf(predicate: A => Boolean) =

Effect[A] { x =>

if (predicate(x)) this(x) else ()

}

}