ToAttr Pattern Documentation

This documents the `ToAttr` typeclass pattern that was used in PSD3 for polymorphic attribute setters. This pattern was superseded by the finally-tagless expression system but is documented here for reference.

The Pattern

The `ToAttr` typeclass enabled a single function to accept static values, datum-driven functions, or indexed functions:

```
-- The typeclass with functional dependency
class ToAttr :: Type -> Type -> Type -> Constraint
class ToAttr to from datum | from -> to where
  toAttr :: from -> AttributeName -> Attribute datum
```

Instances

The typeclass had instances for three usage patterns, for each supported type (String, Number, Boolean):

```
-- String instances
instance ToAttr String String datum where
  toAttr value name = StaticAttr name (StringValue value)

instance ToAttr String (datum -> String) datum where
  toAttr fn name = DataAttr name (StringValue <<< fn)

instance ToAttr String (datum -> Int -> String) datum where
  toAttr fn name = IndexedAttr name (\d i -> StringValue (fn d i))

-- Number instances
instance ToAttr Number Number datum where
  toAttr value name = StaticAttr name (NumberValue value)

instance ToAttr Number (datum -> Number) datum where
  toAttr fn name = DataAttr name (NumberValue <<< fn)

instance ToAttr Number (datum -> Int -> Number) datum where
  toAttr fn name = IndexedAttr name (\d i -> NumberValue (fn d i))

-- Boolean instances
instance ToAttr Boolean Boolean datum where
  toAttr value name = StaticAttr name (BooleanValue value)

instance ToAttr Boolean (datum -> Boolean) datum where
  toAttr fn name = DataAttr name (BooleanValue <<< fn)
```

```
instance ToAttr Boolean (datum -> Int -> Boolean) datum where
  toAttr fn name = IndexedAttr name (\d i -> BooleanValue (fn d i))
```

Smart Constructors

The typeclass enabled polymorphic smart constructors:

```
-- Fill accepts: "red", (\d -> d.color), or (\d i -> ...)
fill :: forall datum a. ToAttr String a datum => a -> Attribute datum
fill value = toAttr value (AttributeName "fill")

-- cx accepts: 50.0, (\d -> d.x), or (\d i -> toNumber i * 100.0)
cx :: forall datum a. ToAttr Number a datum => a -> Attribute datum
cx value = toAttr value (AttributeName "cx")

-- Similar for: cy, radius, x, y, width, height, stroke, opacity, etc.
```

Usage Examples

```
-- Static value
fill "steelblue"
cx 50.0

-- Datum-driven
fill (\d -> d.color)
cx (\d -> d.x * 20.0 + 50.0)

-- Indexed (useful for staggered animations)
fill (\d i -> if i == 0 then "red" else "blue")
cx (\d i -> toNumber i * 100.0 + 50.0)
```

Why It Was Superseded

The `ToAttr` pattern worked well for simple cases but had limitations:

1. **No multi-interpretation**: Functions are opaque - you can only evaluate them, not inspect or transform them.
2. **No code generation**: Cannot generate JavaScript or documentation from `\d -> d.x * 20.0`.
3. **No AST manipulation**: Cannot write `f :: AST -> AST` transformations.
4. **No type-safe field access**: Field access like `d.x` is stringly-typed at the PureScript level.

The finally-tagless expression system addresses all these:

```
-- Finally-tagless: same expression, multiple interpretations
letterX :: forall repr. NumExpr repr => DatumExpr repr LetterRow => repr
Number
letterX = field @"index" `timesN` 40.0 `plusN` 50.0

-- Evaluate to a value
runEvalD letterX datum 0  -- => 90.0

-- Generate code
runCodeGen letterX  -- => "(d.index * 40) + 50"

-- (Future) Transform the AST
optimize letterX  -- => simplified expression
```

The Underlying Types (Still Used)

The core types from this module remain in use:

```
-- The Attribute ADT
data Attribute datum
  = StaticAttr AttributeName AttributeValue
  | DataAttr AttributeName (datum -> AttributeValue)
  | IndexedAttr AttributeName (datum -> Int -> AttributeValue)

-- Attribute names
newtype AttributeName = AttributeName String

-- Attribute values with type information
data AttributeValue
  = StringValue String
  | NumberValue Number
  | BooleanValue Boolean
```

These types are the target of the finally-tagless interpreters - expressions compile down to `Attribute datum` values.

Design Inspiration

The `ToAttr` pattern was inspired by Ian Ross's Haskell D3 library, which used a similar approach for polymorphic attribute setters.