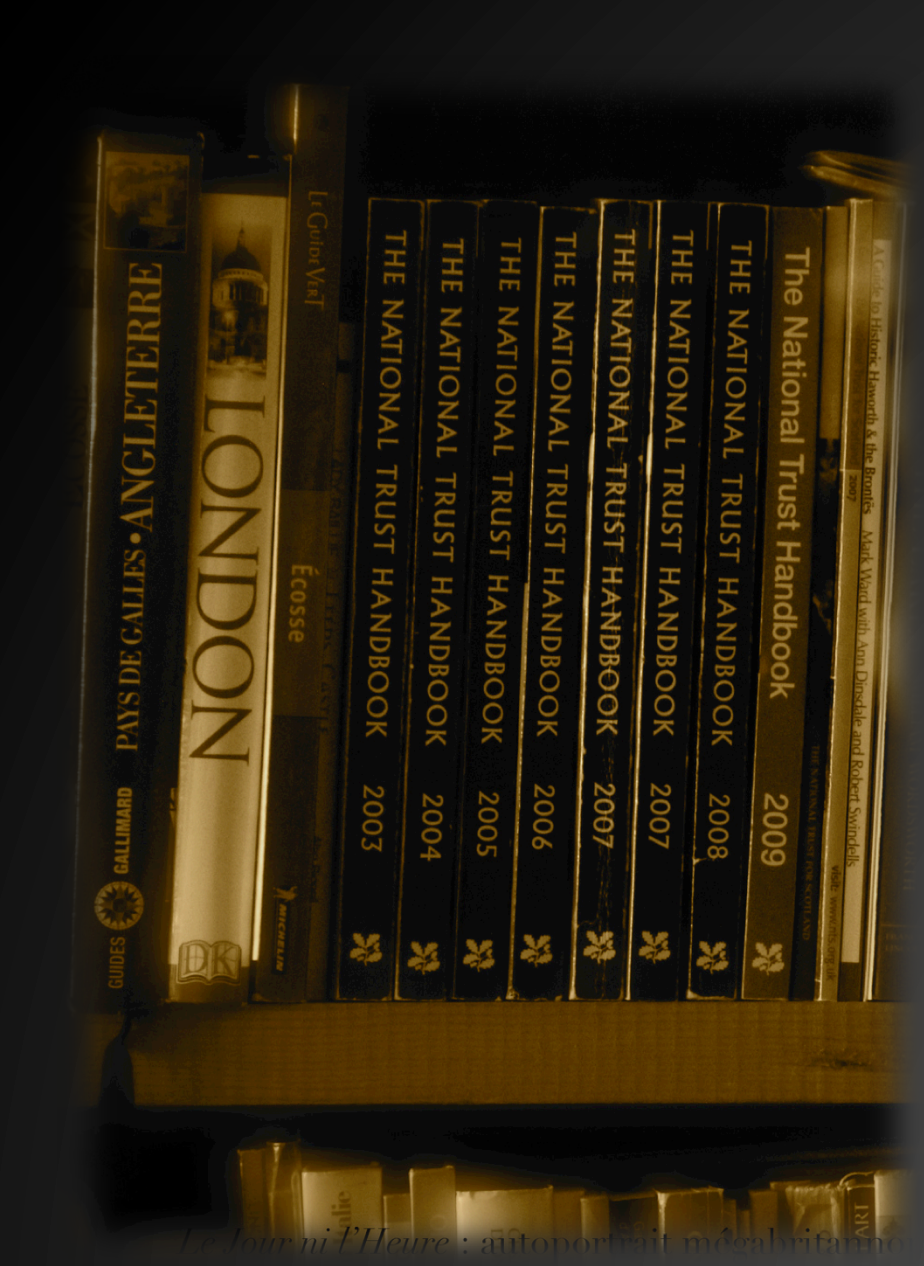


A Field Guide to DSL Design

Tomer Gabel, Wix

Jfokus, January 2015

WIX

DSLs

- Designed for specific *application domains*
- Designers can make more *assumptions*
- This enables *optimized syntax* for:
  - Conciseness
  - Correctness
  - Readability

WIX

External DSL

- Standalone specification and grammar
- *Explicit* library/tool support (lexer, parser)
- Explicit *lifecycle* (build phase or runtime)

WIX

Examples

- Data querying with *SQL*:

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2
  ON s1.article = s2.article
  AND s1.price < s2.price
WHERE s2.article IS NULL;
```

WiX

Examples

- Text formatting with *Markdown*:

```
Download
--------

[Markdown 1.0.1][dl] (18 KB) -- 17 Dec
2004

[dl]: http://daringfireball.net/projects/
downloads/Markdown_1.0.1.zip
```

Internal DSL

- Extends some host language (Scala)
- Exploits the host to add new syntax
  - *Within the confines* of the host
  - Cannot add new *syntactic constructs*
  - Usage is *valid code* in the host language
- Lifecycle is managed by the host

WIX

Examples

- Testing with *ScalaTest*:

```
"Empty combinator" should {
  "successfully validate an empty sequence" in {
    val left = Seq.empty[ String ]
    val validator = new Empty[ Seq[ String ] ]
    validator( left ) should be( aSuccess )
  }
  "render a correct rule violation" in {
    val left = Some( "content" )
    val validator = new Empty[ Option[ String ] ]
    validator( left ) should failWith( "must be empty" )
  }
}
```

WIX

Examples

- Request routing with *Scalatra*:

```
get("/guess/:who") {
   params("who") match {
     case "Frank" => "You got me!"
     case _ => pass()
   }
}
```

WIX

Examples

- JSON AST construction with *Play*:

```
Json.obj( "users" -> Json.arr(
  Json.obj( "name"  -> "bob",
            "age"   -> 31,
            "email" -> "bob@gmail.com" ),
  Json.obj( "name"  -> "kiki",
            "age"   -> 25,
            "email" -> JsNull )
) )
```

WIX


STEP 1: KNOW YOUR DOMAIN
WIX

Specificity

- Identify your *actors*
- What *actions* can they take?
- What *objects* do they act on?
- What are the *consequences* of these actions?

WIX

Example: Assertions

- *Caller* has a piece of *data*
- And an *assumption* on its shape
- Caller *asserts* that the assumption holds
- If not, an *exception* is thrown

Actor

Object

Object

Action

Consequence

WIX

Audience

- Know your *users*
- Which *actor* do they represent?
- What is the desired *consequence*?
- How would they *express* this desire?

WIX

Example: Assertions

- Only one actor and consequence
- But how to best *express* this desire?
- Start with the *user's* perspective:
  - We already have some data
  - We need a way to define *assumptions*
  - And concise syntax for *assertions*

WIX

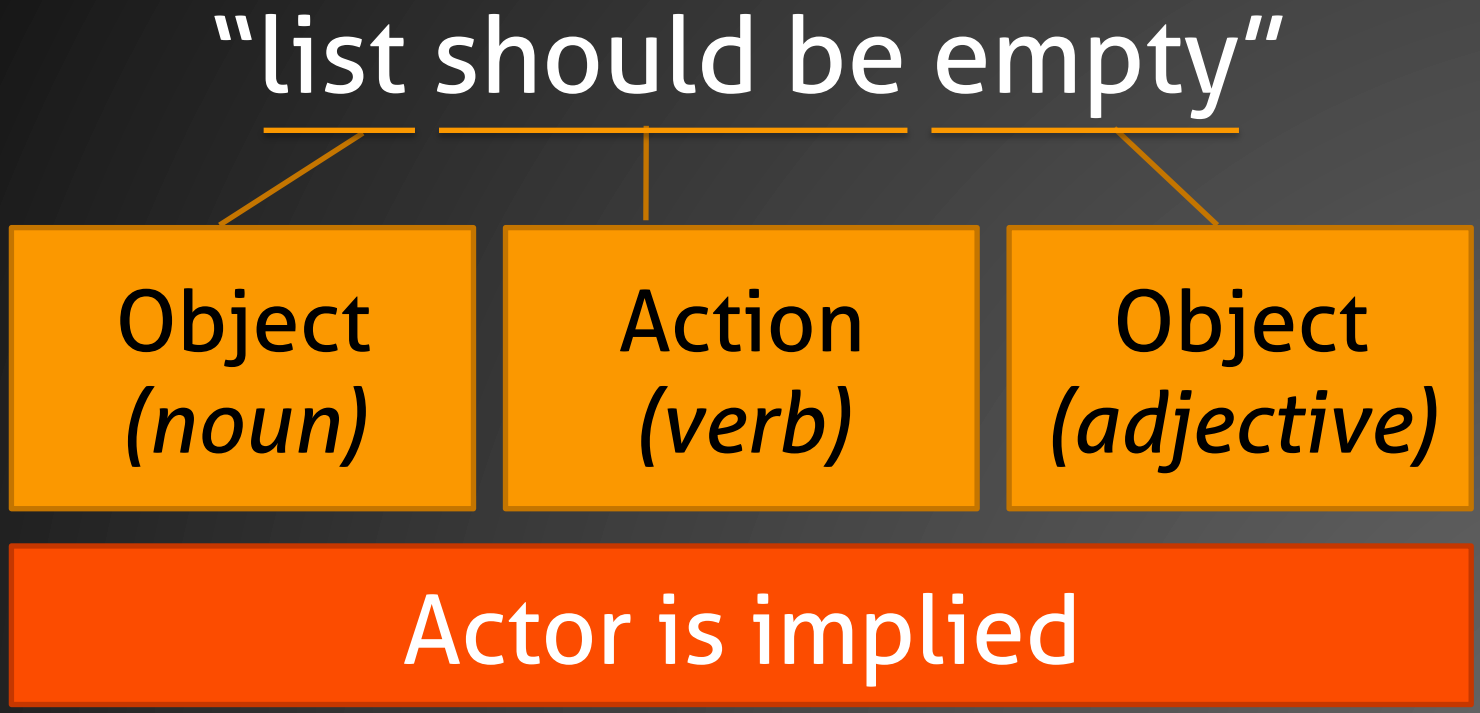
Articulation

- Choose a vocabulary:
  - Nouns describe *objects*
  - Adjectives *qualify* objects
  - Verbs describe *actions*
  - Adverbs *qualify* actions

WIX

Example: Assertions

- An assertion is a *sentence:*



WIX

Example: Assertions

- *Assumptions* (simple)
  - empty
  - null
  - true
  - false
  - left
  - right
  - defined
  - completed
  - ...
- *Assumptions* (parameterized)
  - equalTo
  - startWith
  - endWith
  - contain
  - matchRegex
- *Modifiers* (adverbs)
  - not

WIX

Mode

- *Imperative DSLs* evaluate eagerly
- DSL expressions have *effects*
- Sometimes called "*shallow embedding*"
- Examples:
  - Ant tasks
  - Assertions

WIX

Mode

- *Prescriptive DSLs* are pure and lazy
- Results are *execution plans*
- Prescribe behavior to subsequent logic
- Also called "*deep embedding*"
- Examples:
  - Request routing
  - Query languages

WIX

Mode

- Prescriptive DSLs are *more powerful*
  - Result and evaluation can be optimized
  - Evaluation can be deferred freely
- But are more complex to get right
  - Results require a *domain model*
  - Evaluation requires separate logic

WIX


STEP 2: KNOW YOUR LANGUAGE
WIX

Infix Notation

- Also known as "*dot free syntax*"
- Basic building block in fluent DSLs
- Applies to arity-1 *function applications*

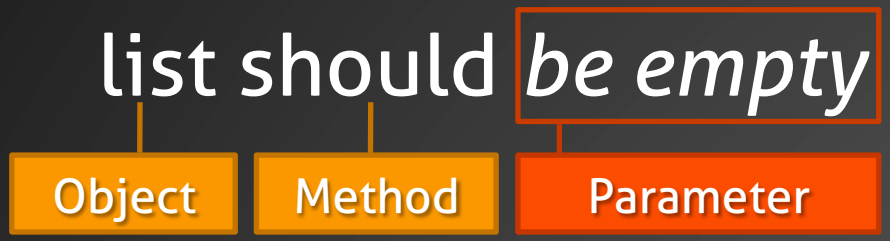
```
object.method(parameter)
```

```
object method parameter
```

WIX

Infix Notation: Caveats

- Chaining must be done with care

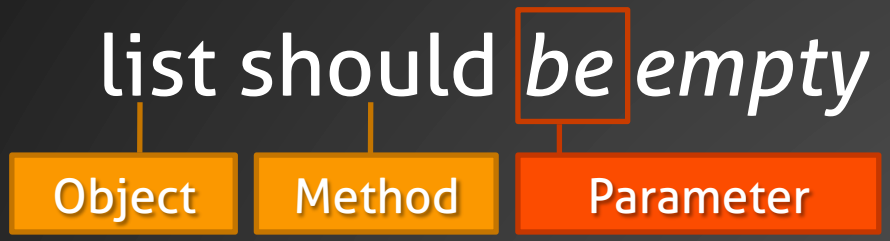


- This is what we expect.

WIX

Infix Notation: Caveats

- Chaining must be done with care



- Infix notation is bloody literal!
- What is the type of *be*?
  - Probably not what you meant

WIX

Infix Notation: Caveats

- Workaround 1: Contraction
  - ➔ list *shouldBe* empty
- Workaround 2: Parentheses
  - ➔ list should be *(*empty*)*
- There is no "right" answer
  - It's an aesthetic preference

WIX

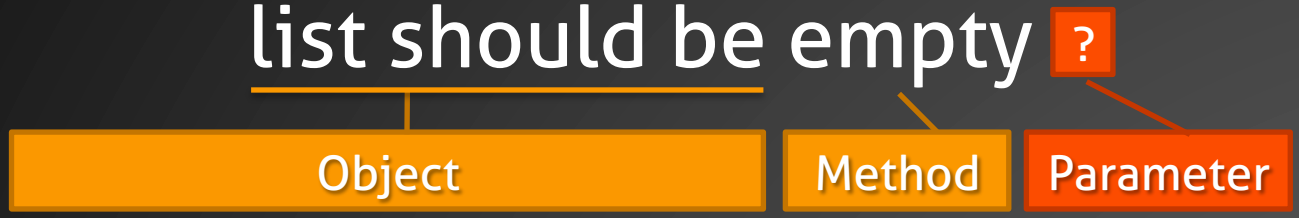
Infix Notation: Caveats

- Chaining must be done with care

WIX

Infix Notation: Caveats

- Chaining must be done with care



- Must have odd number of participants
- This expression is *illegal* in Scala!
  - (Unless you use postfix operators. Don't.)

WIX

Implicit Classes

- The entry point into your DSL

list *shouldBe* empty

Known a-priori

Extension Method

- The extended type is *domain-specific*
- Prefer value classes for performance
- Also used to *lift* values into your domain

WIX

STEP 3:
PUTTING IT TOGETHER

WIX

Foundation

- An assertion is a sentence with the shape: *data shouldBe predicate*
- We first need a *predicate* definition:

```
trait Predicate[-T] {
  def test: T ⇒ Boolean
  def failure: String
}
```

WIX

Foundation

- We next need an *entry point* into our DSL
- Data is the only known entity
- We'll need to extend it:

```
implicit class EntryPoint[T](data: T) {
  def shouldBe(predicate: Predicate[T]) = ???
}
```

WIX

Foundation

- This is an *imperative* DSL
- Assertion failure has *consequences*
- In our case, an exception is thrown*:*

```
implicit class EntryPoint[T](data: T) {
  def shouldBe(predicate: Predicate[T]) =
    require(predicate test data,
            s"Value $data ${predicate.failure}")
}
```

WIX

Foundation

- We can start implementing *predicates*:

```
Nil shouldBe empty
```

- A generic solution is readily apparent:

```
def empty[T <: GenTraversableOnce[_]] =
  new Predicate[T] {
    def test: T ⇒ Boolean = _.isEmpty
    def failure = "is not empty"
  }
```

So far,
so good.

NOW THINGS GET HAIRY

Keywords

- What about *booleans*?

```
(3*4>10) shouldBe true
```

- Booleans are reserved *keywords*
  - Can't provide a def
  - Can't provide an object
- Can we support this syntax?

WIX

Keywords

- Workaround: *Lift* via an implicit class

```
implicit class BooleanPredicate(b: Boolean)
  extends Predicate[Boolean] {

  def test: Boolean ⇒ Boolean = _ == b
  def failure = s"is not $b"
}
```

WIX

Keywords

- We have a similar issue with *null*:

```
val ref: String = null
ref shouldBe null
```

- We can't lift null implicitly, because:

```
def shouldBe(predicate: Predicate[T]): Unit
```

- Null is *bottom type*, extends Predicate[T]
- Implicit search does not take place!

WIX

Keywords

- Workaround: Specific *method overload*
  - Should only apply when T is a reference type

```
implicit class EntryPoint[T](data: T) {
  // ...

  def shouldBe(n: Null)
              (implicit ev: T <:< AnyRef): Unit =
    require(data == null,
            s"Value $data is not null")
}
```

WIX

Parameterization

- What about *parameterized* predicates?

```
3*4 shouldBe equalTo 12
```

- The equality predicate is simple enough:

```
def equalTo[T](rhs: T) = new Predicate[T] {
  def test: T ⇒ Boolean = _ == rhs
  def failure = s"is not equal to $rhs"
}
```

- But we have an *even* number of parts!

WIX

Parameterization

- Workaround: *Parentheses*

```
3*4 shouldBe equalTo(12)
```

- There is *no way** to avoid this entirely
  - Some sentences are shorter
  - Impossible to guarantee the odd part rule

** ... that I know of*

WIX

Grammar Variance

Assumption	Example
startWith	`"Jfokus" should startWith("J")`
endWith	`"Jfokus" should endWith("fokus")`
contain	`List(1, 2, 3) should contain(2)`
matchRegex	`"Jfokus" should matchRegex("Jf.*")`

Can you spot the difference?

WIX

Grammar Variance

- We must support predicate families:
  - Simple modal form:

```
List(1, 2, 3) should contain(2)
```

  - Compound subjunctive form:

```
3*4 shouldBe equalTo(12)
```

- In other words, we need another verb

WIX

Grammar Variance

- A simple solution:

```
implicit class EntryPoint[T](data: T) {
  private def test(predicate: Predicate[T]): Unit =
    require(predicate test data,
            s"Value $data ${predicate.failure}")

  def shouldBe(predicate: Predicate[T]): Unit =
    test(predicate)
  def should(predicate: Predicate[T]): Unit =
    test(predicate)
}
```

WIX

Grammar Variance

- A simple solution:

```
implicit class EntryPoint[T](data: T)
  private def test(predicate: Predicate[T]): Unit =
    require(predicate(data),
            s"value $data failed predicate: $failure}")

  def should(predicate: Predicate[T]): Unit =
    test(predicate)
  def should(predicate: Predicate[T]): Unit =
    test(predicate)
}
```

BAD

Grammar Variance

- Incorrect grammar is legal:

```
List(1, 2, 3) shouldBe contain(2)
```

- We lack differentiation between families
- First, define adequate base traits:

```
trait ModalPredicate[-T] extends Predicate[T]
trait CompoundPredicate[-T] extends Predicate[T]
```

WIX

Grammar Variance

- Next, modify the verbs accordingly:

```
def should(predicate: ModalPredicate[T])…
def shouldBe(predicate: CompoundPredicate[T])…
```

- We must also enforce the decision:
  - Make the base trait Predicate[T] *sealed*
  - Move it to a separate compilation unit
- Finally, modify all predicates to comply

WIX

Negation

- Negation ("*not*" adverb) is not just syntax
  - Predicate must support negation
  - Negative messages must be available
- We must extend the model
- But first we must decide on the grammar

WIX

Negation

- Compound?
  - `List(1, 2, 3) shouldNotBe empty`
  - `List(1, 2, 3) shouldBe not(empty)`
  - `List(1, 2, 3) shouldNot be(empty)`
- Modal?
  - `"Programmers" shouldNot startWith("Java")`
  - `"Programmers" should not(startWith("Java"))`
- Again, an aesthetic choice

WIX

Negation

- Predicate[T] extended with negation:

```
sealed trait Predicate[-T] {
  def test: T ⇒ Boolean
  def failure: String
  def failureNeg: String

  type Self[-T] <: Predicate[T]
  def negate: Self[T]
}
```

Negative messages

Generic negation

WIX

Negation

- Adding negation support to each family:

```
trait ModalPredicate[-T]
  extends Predicate[T] { self ⇒

  type Self[-T] = ModalPredicate[T]
  def negate = new ModalPredicate[T] {
    def test = self.test andThen { !_ }
    def failure = self.failureNeg
    def failureNeg = self.failure
  }
}
```

WIX

Negation

- Finally, add the *not* modifier (adverb):

```
def not[T](pred: Predicate[T]): pred.Self[T] =
  pred.negate
```

- Et voilà:

```
List(1, 2, 3) shouldBe not(empty)
"Programmers" should not(startWith("Java"))
```

WIX


STEP 4: PROFIT
WIX

tomer@tomergabel.com

@tomerg

http://il.linkedin.com/in/tomergabel

Sample code at:

http://tinyurl.com/scala-dsl-guide

Thank you for listening

WE'RE DONE HERE!

WIX