E Fundamentals… with Donuts

Marc Stiegler
Visiting Scholar, HP

Introduction

- When should we meet again?
- Installing E
- 15 Minutes of *E* Syntax
- makeRevokerPair
- Basic Sealer
- Ping Sealer
- Shared Variable Sealer

Tell Me If I Go
Too Fast Or
Too Slow!

Quick-Ref Card…Objects

```
? pragma.disable("explicit-result-guard")
? pragma.enable("easy-return")

? def origin {
>     to getX() {return 0}
>     to getY() {return 0}
> }
# value: <origin>

? origin.getY()
# value: 0

? def add(a, b) {return a + b}
# value: <add>

? add(2,3)
# value: 5
```

```
? println("hello")
hello
? println.run("hello")
hello

? def makePoint(x,y) {
>     def point {
>         to getX() {return x}
>         to getY() {return y}
>     }
>     return point
> }
# value: <makePoint>

? def  p1 := makePoint(1,2)
# value: <point>

? p1.getY()
# value: 2
```

MakeRevokerPair

```
? def stockPredictor() {return "up 10%"}
# value: <stockPredictor>

? def makeRevokerPair(var target) {
>     def revoker {to revoke() {target := null}}
>     def forwarder {
>         match [verb, args] {E.call(target, verb, args)}
>     }
>     return [forwarder, revoker]
> }
# value: <makeRevokerPair>

? def [predictor, revoker] := makeRevokerPair(stockPredictor)
# value: [<forwarder>, <revoker>]

? predictor()
# value: "up 10%"
```

```
? revoker.revoke()
? predictor()
# problem:
<NoSuchMethodException:
<null>.run/0>
```

Sealer / Unsealers

```
? def makeSealerPair := <elib:sealing.Brand>
# value: <import:org.erights.e.elib.sealing.Brand>

? def [sealer, unsealer] :=
makeSealerPair("marcsBrand")
# value: [<marcsBrand sealer>, <marcsBrand
unsealer>]

? def box := sealer.seal(stockPredictor)
# value: <sealed by marcsBrand>

?  def unsealedPredictor := unsealer.unseal(box)
# value: <stockPredictor>

? unsealedPredictor()
# value: "up 10%"
```

Like Public / Private Key

Easy

No Fancy Crypto

Can Box Authorities, Not Just Data

Caps: More TTPs

Ping Sealer

```
def makeSealerPair() {
    def boxes := [].asMap().diverge()
    def sealer {
        to seal(obj) {
            def box{}
            boxes[box] := obj
            return box
        }
    }
    def unsealer {
        to unseal(box) {
            return boxes[box]
        }
    }
    return [sealer, unsealer]
}
```

What's the Problem?

Shared Variable Sealer

```
def makeSealerPair() {
    def noObject{}
    var shared := noObject
    def sealer {
        to seal(obj) {
            def box {to share() {shared := obj} }
            return box
    }   }
    def unsealer {
        to unseal(box) {
            shared := noObject
            box.share()
            if (shared == noObject) {throw("bad box")}
            def obj := shared
            shared := noObject
            return obj
    }   }
    return [sealer, unsealer]
}
```

Incompatible With Threads

Good GC

"share" message risk (notary inspector solution)

The Big Difference

- What happens if you wrap a sealed box in a revocable forwarder?

Eventual Sends, Promises, When-Catches

```
? println <- run("hello")
# value: <Promise>

hello
?

? def print2() {
>     println <- ("hello1")
>     println("hello2")
> }
# value: <print2>

? print2()
hello2
hello1
?
```

```
? def showWhen() {
>     def printVow := println <-("Hello")
>     when (printVow) -> done(printed) {
>         println("Beyond Hello")
>     } catch prob {println ("dead: " + prob)}
> }
# value: <showWhen>

? showWhen()
Hello
Beyond Hello
?
? def num
# value: <Resolver>

? num
# value: <Promise>

? bind num := 3
# value: 3
```

Promise Resolvers

```
? def vowDouble(numVow) {
>     def double
>     when (numVow) -> done(num) {
>         bind double := 2*num
>     } catch prob {throw(prob)}
>     return double
> }
# value: <vowDouble>

? def valResolver := (def val)
# value: <Resolver>

? def doubleVal := vowDouble(val)
# value: <Promise>

? valResolver.resolve(3)
? val
# value: 3

? doubleVal
# value: 6
```

```
? def vowDouble(numVow) {
>     return numVow <- multiply(2)
> }
# value: <vowDouble>

? def val
# value: <Resolver>

? def doubleVal := vowDouble(val)
# value: <Promise>

? bind val := 3
# value: 3

? doubleVal
# value: 6
```

Rolling Dice 1

```
def makePlayer(name, winningNumbers) {
    def didIwin(completeRoll, partnerHalfRoll) {
        def valVow := completeRoll.getRollValueVow(partnerHalfRoll)
        when (valVow) ->done(val) {
            if (winningNumbers.contains(val)) {
                println(name + " won on " + val)
            } else {println(name + "lost on " + val)}
        } catch prob {throw(prob)}
    }
    def player {
        to startRoll(player2) {
            println(name + "started roll")
            def [halfRoll, completeRoll] := makeHalfRollPair()
            def otherHalfRoll := player2.receiveHalfRoll(halfRoll)
            println(name + " got half roll from partner")
            didIwin(completeRoll, otherHalfRoll)
        }
        to receiveHalfRoll(partnerHalfRoll) {
            def [halfRoll, completeRoll] := makeHalfRollPair()
            didIwin(completeRoll, partnerHalfRoll)
            return halfRoll
        }
    }
    return player
}
```

Trivially Breached, but Mind-Twisting

Rolling Dice 2

```
def makeHalfRollPair() {
    def [sealer, unsealer] := makeSealerPair()
    def randomContribution := entropy.nextInt()
    def contributionUnsealer

    def halfRoll {
        to getSealedContribution() {
            return sealer.seal(randomContribution)
        }
        to getContributionUnsealerVow() {
            return contributionUnsealer
        }
    }
```

```
    def completeRoll {
        to getRollValueVow(otherHalf) {
            def rollValueVow
            def sealedOtherContribution :=
              otherHalf.getSealedContribution()
            bind contributionUnsealer := unsealer
            when (otherHalf.getContributionUnsealerVow()) ->
              done(otherUnsealer) {
                def otherContribution := otherUnsealer.unseal(
                    sealedOtherContribution)
                bind rollValueVow :=(otherContribution +
                                     randomContribution)%%6 + 1
            } catch prob {throw ("roll failed")}
            return rollValueVow
        }
    }
    return [halfRoll, completeRoll]
}
```