

Embedding Protocol Buffers

Nathan Howell
Alpha Heavy Industries



Protocol Buffers

- Marketing driven: “a way of encoding structured data in an efficient yet extensible format”
- Easy way of describing and encoding simple versioned messages
- We use it for: IPC, RPC, on-disk storage

Why Bother?

- Less complicated than Thrift, DCE-RPC
- But mental switch: Haskell b/w Protobuf
- Super nasty Setup.hs with Cabal
- hprotoc rebuilds are difficult for shake too

- Since we mostly use Haskell anyways...
- Solution: just do it all in Haskell, wtfn!

Implementation Options

- Preprocessor
- Template Haskell
- Data.Data
- Terrible, terrible boilerplate
- GHC.Generics

Improving your Protobufs

Boring Version

```
message Person
{
  required int32 id = 1;
  required string name = 2;
  optional string email = 3;
}
```

Awesome+++

```
data Person = Person
{
  ident :: Required 1 (Value Int32)
  name :: Required 2 (Value Text)
  email :: Optional 3 (Value Text)
}
```

And GHC.Generics...

- “Generic representation types”
 - Duh.
- Uniform, if somewhat strange language
 - Metadata: `data M1 i c f`
 - Sum types: `data (:+:) f g p`
 - Product types: `data (:*:) f g p`
 - Data constructors: `data K1 i c p`
 - Unit type: `data U1 p`
- Supported in GHC 7.2+

Awaiting Massive Insight

```
Prelude> :module + GHC.Generics
```

```
Prelude GHC.Generics> :info Generic
```

```
class Generic a where
```

```
  type family Rep a1 :: * -> *
```

```
  from :: a -> Rep a x
```

```
  to :: Rep a x -> a
```

Thanks GHCi!

Don't Ask How Sausage Is Made

```
Prelude GHC.Generics> :kind! Rep (Int, Double)
```

```
Rep (Int, Double) :: * -> *
```

```
= M1 D GHC.Generics.D1(,)
```

```
  (M1 C GHC.Generics.C1_o(,)
```

```
    (M1 S NoSelector (K1 R Int) :* M1 S NoSelector (K1 R Double)))
```

```
Prelude GHC.Generics> :kind! Rep (Either Int Double)
```

```
Rep (Either Int Double) :: * -> *
```

```
= M1 D GHC.Generics.D1Either
```

```
  (M1 C GHC.Generics.C1_oEither (M1 S NoSelector (K1 R Int))
```

```
    :+: M1 C GHC.Generics.C1_1Either (M1 S NoSelector (K1 R Double)))
```


But It Is Useful

- Pattern matching¹ on data type *shapes*
- Can help reduce shit-tons of instances
- Or where you'd reach for Template Haskell

1: Think of case statements for types... with no overlap, no guards

Embedding...

`newtype Field n (c (f a))`

- `n` = field tag [1..2²⁹)
- `c` = field type (required, optional, packed, ...)
- `f` = traversable container
- `a` = value with encoding selector

`type family Required n (f a) = Field n (R (f a))`

`type family Optional n (f a) = Field n (O (f a))`

`type family Packed n (f a) = Field n (P (f a))`

`type family Repeated n (f a) = Field n (S (f a))`

Wrapped Type Families

Required $n (f a) \cong a$

Optional $n (f a) \cong \text{Maybe } a$

Repeated $n (f a) \cong [a]$

Packed $n (f a) \cong [a]$

- Not injective: phantoms and overlap
- Fine for record fields as types are fixed

Field Accessors

Hint: use getField and putField, or field

```
let foo= Person
{
  ident = putField 42
  name = putField "Arthur Dent"
  email = putField "adent@milliways.com"
}
```

Message Encoding

`(SingI n, Foldable f, WireEnc a) => Field n (c (f a))`

`encode :: WellFormedRecord -> ByteString`

`encode = runPut . traverse_ encoder . fields`

Nice, simple.

(Actual implementation type-checks and is ~20 lines)

Message Decoding

- Not very generalized...
- But it works, trust QuickCheck.

Field Encoding Selectors

- Overlap breakers:
 - newtype Value a
 - newtype Message a
 - newtype Enumeration a
- Means
 - Never: Field ... Int64
 - Always: Field ... (Value Int64)

Value Encoding Selectors

- Done with lots of instances
 - Value {Int16, Word32, Int64, Text, ...}
 - Value (Fixed ...)
 - Value (Signed ...)
- Adding instances is rarely necessary
- Would be nice to generalize this somewhat

Parsing.proto

- gen-hs-proto plugin for protoc
 - Proof-of-concept available today
 - Parses .proto files...
 - Doesn't yet produce working datatypes

Also: Generating .proto

- Purely type-level
- Working with generic type representation
- It can produce almost valid .proto files
... unlike gen-hs-proto

What's really not done

- Default values
 - Semi-easy for generated code (encode default value as type-level string?)
 - Gross data-default hacks ☹
 - Difficult: defaults for a field, not type
- Extensions
- Preserving unknown fields
- Services

Embrace the Protobuf

hackage.haskell.org/package/protobuf

github.com/alphaHeavy/protobuf

breaks.for.alienz.org

alphaheavy.com

[@nathanhowell](#)

