Scodec: How to create a codec for an optional byte

I must create a codec for a message that has the following specification The message length is indicated by a byte of which the least significant bit is an extension flag that, when set indicates that the following (optional) byte must be used as the most significant byte. (Hope it make sense) It can be depicted as follows:

```
+----------------------------------------------------------------------------------
+
|                                    LENGTH
|
|
|
+----------------------------------+-----+-+---------------------------------------
+
|                                  |     | |
|
|               Len1 (LSB)         | Ext | |         Len2 (MSB) - Optional
|
+----+----+----+----+----+----+----+-----+ +----+----+----+----+----+----+----+----
+
|    |    |    |    |    |    |    |     | |    |    |    |    |    |    |    |
|
|    |    |    |    |    |    |    |  +  | |    |    |    |    |    |    |    |
|
+----+----+----+----+----+----+----+--|--+ +----+----+----+----+----+----+----+----
+
                                      |
                                      |
                                      v
                        Boolean: if true then Len2 is used
                                else only Len1
```

The length of the data that will follow is determined by this field(s). I would like to use the codec along with predefined codecs and combinators. I guess it will involve using flatZip but I am not clear on how to incorporate flatZip into an HList combinator. Any pointers to examples or documentation will be much appreciated.

scodec

edited May 15 '15 at 7:07

asked May 14 '15 at 22:19

iandebeer
107 1 7

1 Answer

One way to do this is using the `scodec.codecs.optional` combinator, which returns a `Codec[Option[A]]` given a `Codec[Boolean]` and a `Codec[A]`.

```
val structure: Codec[(Int, Option[Int])] = uint(7) ~ optional(bool, uint8)
```

This gives us a codec of `(Int, Option[Int])` - we need to convert this to a codec of `Int`. To do so, we'll need to provide a conversion from `Int` to `(Int, Option[Int])` and another conversion in the reverse direction. We know the size field is at most 2^15 - 1 (7 LSB bits and 8 MSB bits), so converting from `(Int, Option[Int])` to `Int` is total, whereas converting in the reverse direction could possibly fail -- for example, 2^16 cannot be represented in this structure. Hence, we can use `widen` to do the conversion:

```
val size: Codec[Int] = structure.widen[Int](
  { case (lsb, msb) => lsb + msb.map(_ << 7).getOrElse(0) },
  { sz =>
    val msb = sz >>> 7
    if (msb > 255 || msb < 0) Attempt.failure(Err(s"invalid size $sz"))
    else Attempt.successful((sz & 0x7F, if (msb > 0) Some(msb) else None))
  })
```

Finally, we can use this size codec to encode a variable length structure via `variableSizeBytes`:

```
val str: Codec[String] = variableSizeBytes(size, ascii)
```

This gives us a `Codec[String]` which prefixes the encoded string bytes with the size in bytes, encoded according to the scheme defined above.

answered May 15 '15 at 12:25
mpilquist
3,320 14 20

1 Thank you so much for the comprehensive answer. Not only does it help with specific question, but it peels back another layer of the scodec onion. Regards, Ian – iandebeer May 16 '15 at 11:58