vctrs: Creating custom vector classes with the vctrs package

Jesse Sadler
Loyola Marymount University

@vivalosburros
jessesadler.com
github.com/jessesadler


DANIEL. VANDER.
MEVLEN.
Aº.1583.


HESTER.DELLA.
FAILLE.
Aº.1583.

Anno 1585.

		£	s	d
	Jacques della faille de jonghe. is crediteur Adi 15. februarij aº 1585. £ 26. 4. 0. voor oncosten van onse Bruijloft voor 50½ ell. lywaet a 10s d'ell aen Willem Monninx betaelt ende 19 s. voor vracht van den coffer met 7. packetten compt tsamen. — — — — — — — — £	26	4	—
—	ditto £ 0. 0. 0. voor de voorss. voor schipvracht van Jo: Gysber della faille met diversche packetten int vertreck van Antwerpen. — — — — £	0	—	—
—	ditto £ 5. 1. 10. voor de voorss. voor een silvere schale wegende 11 on 15 eng. aen Jo: Gysber della faille in october overgelaten. — — — £	5	1	10
—	ditto £ 28. 10. 0. voor de voorss. voor 4. amen renschen wijn voor de Bruijloft tot Dort gecoft compt met de vaten. — — — — — — £	28	10	—
—	ditto £ 4. 7. 0. voor de voorss. voor diverse lacken ende carsey aen Jo: Gysber della faille over lange overgelaten — — — — — — £	4	7	0
—	ditto £ 369. 4. 4. voor de voorss. voor een rekeninge van oncosten op de Bruijloft gedaen als breeder bij selve rekening is blijckende. — — £	369	4	4
20	ditto £ 85. 4. 11. voor de voorss. voor een rekeninge van oncosten tot behoeve vande Bruijloft bij Jacques Noirot in Antwerpen betaelt. £	85	4	11
—	ditto £ 15. 18. 10. voor de voorss. voor soo veele mij Jan della faille voor de conduite vant Bruijloft werck. int gaen ende comen betaelt. £	15	18	10
—	ditto £ 2. 18. 6. voor de voorss. voor soo veele aende timmerman voor sparren deelen ende arbeyt betaelt — — — — — — — £	2	18	6
—	ditto £ 0. 3. 4. voor de voorss. voor vracht ende licencie van een ghelvis aen schip comende van Londen — — — — — — — — £	—	3	4
10 Marti	£ 155. 1. 4. voor tarwe in handen van Robert Noirot in Emiclien voor dat ick inde 235. quarters tarwe geladen int schip de roose de reyse genaempt [illegible]			

Why and how to use vctrs

Examples with debvctrs

- debvctrs on GitHub:
  - github.com/jessesadler/debvctrs
- Simplified version of debkeepr:
  - jessesadler.github.io/debkeepr
- Step-by-step guide to building S3-vector classes with vctrs
  - Use in tandem with vctrs S3 vignette

vctrs

https://vctrs.r-lib.org

Goals of vctrs

- Type stability
  - Predict output type knowing input types
  - Order of arguments in … does not affect output type
- Size stability
  - Predict output size knowing input sizes or with single numeric input that specifies size
- Make it easier to build new S3-vector classes

What do you get by using vctrs?

- Clear development path for creating an S3 class
- Consistency with base R functionality
- Integration with the tidyverse
  - Work in progress: tibble and dplyr
- Hitch yourself to tidyverse implementations and philosophies

debvctrs: Tutorial for building S3 vectors with vctrs

Based on debkeepr
(double-entry bookkeeper)
jessesadler.github.io/debkeepr

debvctrs scripts

See package README

- 01.1-decimal-class.R
- 01.2-lsd-class.R
- 01.3-checks.R
- 02-coercion.R
- 03-casting.R
- 04-comparison-lsd.R
- 05-mathematical-funcs.R
- 06-arithmetic-ops.R
- attr-conversion.R
- lsd-normalize.R
- utils.R
- debvctrs-package.R

Non-decimal currency nomenclature lsd

libra	solidus	denarius
£	s.	d.
Pound	shilling	penny (pence)

Problem space

Compound unit arithmetic

	£	s.	d.
	28	15	8
	32	8	11
	54	18	7
	18	12	9
Answer	**£134**	**15s.**	**11d.**
Unit total	132	53	35
Divide by base	-	53 / 20	35 / 12
Carried forward	2	2	-
Remainder	-	13	11

- Three separate units make up one value
- The units have non-decimal bases
- Need to use compound-unit arithmetic to normalize values
- The non-decimal bases differed by currency

Design principles

- A class that maintains the tripartite structure of non-decimal currencies
- Decimalized class as fall back
- Track the bases of shillings and pence units
- Vectors with different bases cannot be combined
- Choose and track unit represented by decimalized class
- Vectors with different units can be combined but need coercion path

Structure of the classes

deb_lsd

record-style vector

```
deb_lsd(l = c(17, 32, 18),
        s = c(16, 7, 12),
        d = c(6, 9, 3))
```

```
#> <deb_lsd[3]>
#> [1] 17:16s:6d 32:7s:9d
#> [3] 18:12s:3d
#> # Bases: 20s 12d
```

deb_decimal

double vector

```
deb_decimal(x = c(17.8250,
                  32.3875,
                  18.6125))
```

```
#> <deb_decimal[3]>
#> [1] 17.8250 32.3875
#> [3] 18.6125
#> # Unit: pounds
#> # Bases: 20s 12d
```

Structure of the classes

deb_lsd

```
deb_lsd(l = c(17, 32, 18),
        s = c(16, 7, 12),
        d = c(6, 9, 3))
```

```
#> <deb_lsd[3]>
#> [1] 17:16s:6d 32:7s:9d
#> [3] 18:12s:3d
#> # Bases: 20s 12d
```

deb_decimal

```
deb_decimal(x = c(17.8250,
                  32.3875,
                  18.6125))
```

```
#> <deb_decimal[3]>
#> [1] 17.8250 32.3875
#> [3] 18.6125
#> # Unit: pounds
#> # Bases: 20s 12d
```

Printing methods

Structure of the classes

deb_lsd

```
deb_lsd(l = c(17, 32, 18),
        s = c(16, 7, 12),
        d = c(6, 9, 3))
```

```
#> <deb_lsd[3]>
#> [1] 17:16s:6d 32:7s:9d
#> [3] 18:12s:3d
#> # Bases: 20s 12d
```

Bases attribute

deb_decimal

```
deb_decimal(x = c(17.8250,
                  32.3875,
                  18.6125))
```

```
#> <deb_decimal[3]>
#> [1] 17.8250 32.3875
#> [3] 18.6125
#> # Unit: pounds
#> # Bases: 20s 12d
```

Structure of the classes

deb_lsd

```
deb_lsd(l = c(17, 32, 18),
        s = c(16, 7, 12),
        d = c(6, 9, 3))

#> <deb_lsd[3]>
#> [1] 17:16s:6d 32:7s:9d
#> [3] 18:12s:3d
#> # Bases: 20s 12d
```

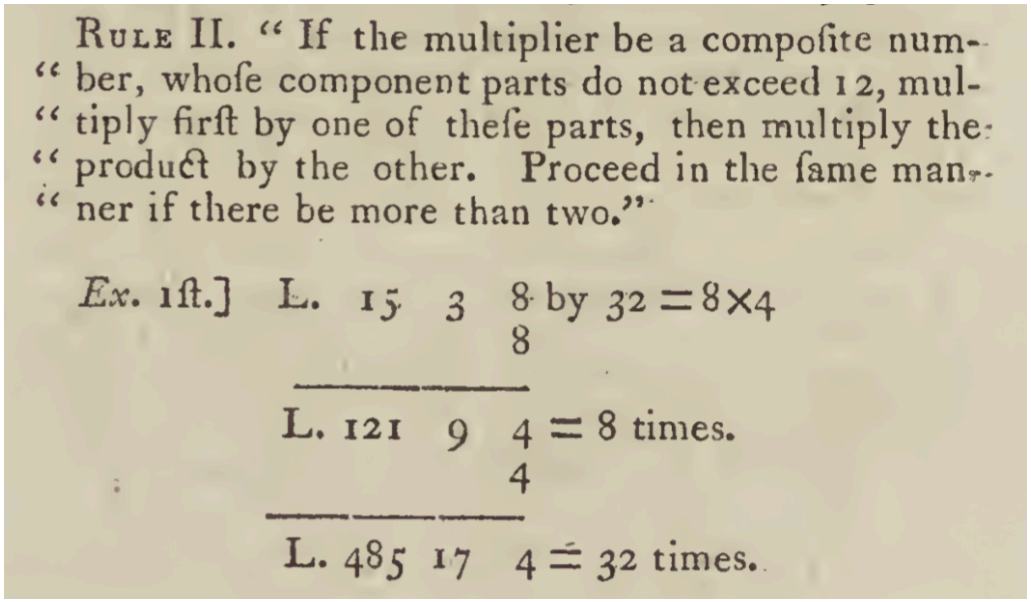
deb_decimal

```
deb_decimal(x = c(17.8250,
                  32.3875,
                  18.6125))

#> <deb_decimal[3]>
#> [1] 17.8250 32.3875
#> [3] 18.6125
#> # Unit: pounds
#> # Bases: 20s 12d
```

Unit attribute

Multiplication



RULE II. "If the multiplier be a compoſite number, whoſe component parts do not exceed 12, multiply firſt by one of theſe parts, then multiply the product by the other. Proceed in the ſame manner if there be more than two."

Ex. 1ſt.] L. 15 3 8 by 32 = 8×4
8

L. 121 9 4 = 8 times.
4

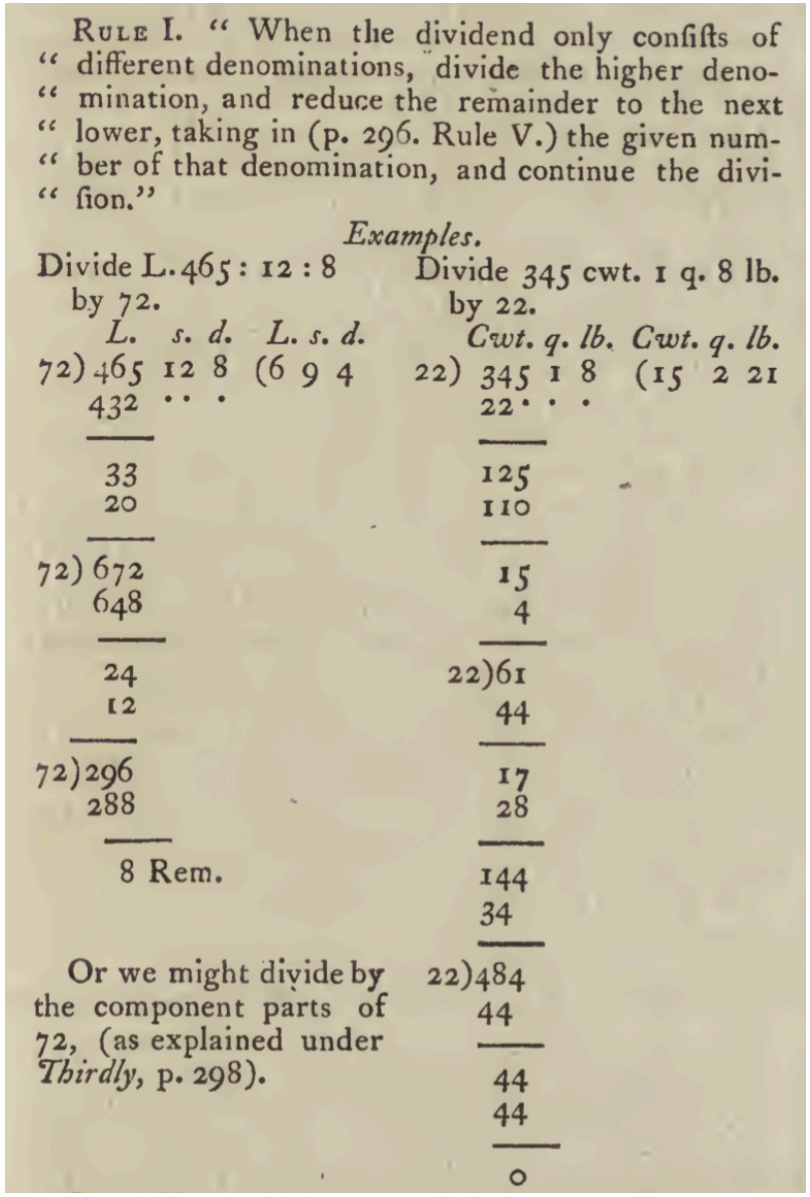
L. 485 17 4 = 32 times.

```
# Multiply £15 3s. 8d.
  sterling by 32

#> deb_lsd(15, 3, 8) * 32

#> <deb_lsd[1]>
#> [1] 485:17s:4d
#> # Bases: 20s 12d
```

Division



RULE I. "When the dividend only consists of "different denominations, divide the higher denomination, and reduce the remainder to the next "lower, taking in (p. 296. Rule V.) the given number of that denomination, and continue the division."

Examples.

Divide L.465 : 12 : 8 by 72.

L.	s.	d.	L.	s.	d.
72) 465	12	8	(6	9	4
432	• •	•			
33					
20					
72) 672					
648					
24					
12					
72) 296					
288					
8 Rem.					

Or we might divide by the component parts of 72, (as explained under *Thirdly*, p. 298).

Divide 345 cwt. 1 q. 8 lb. by 22.

Cwt.	q.	lb.	Cwt.	q.	lb.
22) 345	1	8	(15	2	21
22 •	•	•			
125					
110					
15					
4					
22)61					
44					
17					
28					
144					
34					
22)484					
44					
44					
44					
0					

```
# Divide 345cwt. 1q. 8lbs.
  by 22

#> x <- deb_lsd(345, 1, 8,
                bases = c(4, 28))
#> x / 22

#> <deb_lsd[1]>
#> [1] 15:2s:22d
#> # Bases: 4s 28d
```

Creating S3 vector classes

1. Creation
2. Coercion
3. Casting
4. Comparison
5. Mathematical functions
6. Arithmetic operations

Creating an S3 vector class based on double vector

1. Creation
2. Coercion
3. Casting
4. ~~Comparison~~
5. ~~Mathematical functions~~
6. Arithmetic operations

Nomenclature

- **S3**: A base type with at least a `class` attribute
- **Generic function**: function whose implementation changes depending on the type of vector – `print()` or `summary()`
- **Method**: implementation of a generic function for a specific type – print.factor() or print.numeric()
- **Coercion**: implicit transformation of class – `c()`
- **Casting**: explicit transformation of class – `as_character()`
- **Mathematical functions**: `sum()`, `mean()`, `round()`
- **Arithmetic operations**: `+`, `-`, `*`, `/`

1. Creation

01.1-decimal-class.R, 01.2-lsd-class.r, and 01.3-check.R

1. Constructor: `new_lsd()`
2. Helper: `deb_lsd()`
3. Formally declare S3 class: `setOldClass()`
4. Attribute access: `deb_bases()`
5. Class check: `deb_is_lsd()`
6. Format method
7. Abbreviated name type

1. Constructor: deb_decimal

```
# 1. Define arguments
new_decimal <- function(x = double(),
                        unit = c("l", "s", "d"),
                        bases = c(20L, 12L)) {

# 2. Ensure proper types and sizes for arguments
  vctrs::vec_assert(x, ptype = double())
  unit <- rlang::arg_match(unit)
  vctrs::vec_assert(bases, ptype = integer(),
                    size = 2)

# 3. Create vector class
  vctrs::new_vctr(x,
                  unit = unit,
                  bases = bases,
                  class = "deb_decimal")

}
```

1. Constructor: deb_lsd

```
# 1. Define arguments
new_lsd <- function(l = double(),
                    s = double(),
                    d = double(),
                    bases = c(20L, 12L)) {

# 2. Ensure proper types and sizes for arguments
  vctrs::vec_assert(l, ptype = double())
  vctrs::vec_assert(s, ptype = double())
  vctrs::vec_assert(d, ptype = double())
  vctrs::vec_assert(bases, ptype = integer(), size = 2)

# 3. Create record-style vector class
  vctrs::new_rcrd(list(l = l, s = s, d = d),
                  bases = bases,
                  class = "deb_lsd")
}
```

2. Helper: deb_decimal

```
#1. Define function
deb_decimal <- function(x = double(),
                        unit = c("l", "s", "d"),
                        bases = c(20, 12)) {

# 2. Checks: see 01.3-check.R
  unit <- rlang::arg_match(unit)
  bases_check(bases)

# 3. Cast to allow compatible types for each argument
  x <- vctrs::vec_cast(x, to = double())
  bases <- vctrs::vec_cast(bases, to = integer())

# 4. Use new_decimal() to do actual creation of the vector
  new_decimal(x = x, unit = unit, bases = bases)
}
```

Coercion and casting workflow

- Boilerplate
  - Define method for class
  - Default method for class for incompatible inputs
- Methods within the class
- Methods with compatible classes

2. Coercion: `vec_ptype2()`

02-coercion.R

1. Coercion boilerplate
2. Coercion within the class
3. Coercion with compatible types

3. Casting: `vec_cast()`

03-casting.R

1. Casting boilerplate
2. Casting within the class
3. Casting to and from compatible types
4. Casting function: Casting generic and methods if necessary

Coercion and casting

- Coercion looks for the common type
- Casting does the actual transformation
- Casting makes comparison between classes possible

Allow coercion between deb_decimal vectors with the same bases but different units

```
# Hierarchy: d -> s -> l
unit_hierarchy <- function(x, y) {
  if (identical(deb_unit(x), deb_unit(y))) {
    deb_unit(x)
  } else if (any(c(deb_unit(x), deb_unit(y)) == "l")) {
    "l"
  } else {
    "s"
  }
}

# Actual coercion
vec_ptype2.deb_decimal.deb_decimal <- function(x, y, ...) {
  bases_equal(x, y)
  unit <- unit_hierarchy(x, y)

  new_decimal(bases = deb_bases(x), unit = unit)
}
```

Allow coercion between deb_decimal vectors with the same bases but different units

```
# if else logic within vec_cast() deb_decimal to deb_decimal
vec_cast.deb_decimal.deb_decimal <- function(x, to, ...) {

  # Change value of x from the from_unit to to_unit
  if (from_unit == "l" && to_unit == "s") {
    converted <- x * bases[[1]]
  } else if (from_unit == "l" && to_unit == "d") {
    converted <- x * prod(bases)
  } else if (from_unit == "s" && to_unit == "d") {
    converted <- x * bases[[2]]
  } else if (from_unit == "s" && to_unit == "l") {
    converted <- x / bases[[1]]
  } else if (from_unit == "d" && to_unit == "l") {
    converted <- x / prod(bases)
  } else if (from_unit == "d" && to_unit == "s") {
    converted <- x / bases[[2]]
  }
}
```

Allow coercion between deb_decimal vectors with the same bases but different units

```
# Now you can combine vectors with different units
#> c(deb_decimal(15.5),
     deb_decimal(3255, unit = "d"))

#> <deb_decimal[2]>
#> [1] 15.5000 13.5625
#> # Unit: pounds
#> # Bases: 20s 12d

# Or compare vectors with different units
#> deb_decimal(15.5) < deb_decimal(3255, unit = "d")

#> [1] FALSE
```

4. Comparison

04-comparison-lsd.R

Only necessary for record-style vectors

1. Equality: `vec_proxy_equal()`
2. Comparison: `vec_proxy_compare()`

5. Mathematical functions

05-mathematical-funcs.R

Only necessary for record-style vectors

1. Implemented mathematical methods
2. Unimplemented methods

6. Arithmetic ops: `vec_arith()`

06-arithmetic-ops.R

1. Arithmetic operations boilerplate
2. Arithmetic operations within the class
3. Arithmetic operations with numeric vectors
4. Unary operations

Resources on vctrs

- vctrs websitesite: vctrs.r-lib.org
- The S3 vectors vignette: is particularly important for building an S3-vector class.
- Hadley Wickham, *Advanced R* - Chapter 13: S3
- Hadley Wickham, vctrs: Tools for making size and type consistent functions at RStudio::conf2019