Package 'decimal'

November 14, 2026

Title Exact Arbitrary-Precision Decimal Vectors

Version 0.1.2

Description Arbitrary-precision vectors with an exact decimal representation, avoiding the rounding surprises of binary floating point. Built on the 'mpdecimal' C library, arithmetic is governed by an explicit decimal context controlling precision, rounding, and signaling, and vectors integrate with 'vctrs' for use in data frames, 'tibble' objects, summaries, and common numeric workflows. Missing values, signed zeros, infinities, and not-a-number values are supported throughout. The arithmetic model follows Cowlishaw (2009) ``General Decimal Arithmetic'' <https://speleotrove.com/decimal/decarith.html>.

License MIT + file LICENSE

Language en-US

URL https://github.com/pedrobtz/decimal, https://pedrobtz.github.io/decimal/

BugReports https://github.com/pedrobtz/decimal/issues

Depends R (>= 4.2.0)

Imports methods, rlang, vctrs, withr

Suggests arrow, covr, data.table, dplyr, knitr, pillar, R6, rmarkdown, stringi, tibble, testthat (>= 3.0.0), tidyr

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/documentation roxygen2

Encoding UTF-8

NeedsCompilation yes

Config/roxygen2/version 8.0.0

Author Pedro Baltazar [aut, cre, cph], Stefan Krah [ctb, cph] (Vendored 'mpdecimal' library in src/mpdecimal; see inst/COPYRIGHTS.)

Maintainer Pedro Baltazar <pedrobtz@gmail.com>

Repository CRAN

Date/Publication 2026-11-14 14:00:10 UTC

Contents

adjusted . . . 2
arrow_as_data_frame . . . 3
arrow_decimal_type . . . 4
as_decimal.Array . . . 5
clear_decimal_flags . . . 6
decimal . . . 7
decimal-summaries . . . 8
decimal_arrow . . . 9
decimal_context . . . 11
decimal_flags . . . 12
decimal_from_double . . . 13
fma . . . 13
get_decimal_context . . . 14
is_decimal . . . 14
is_normal . . . 15
is_qnan . . . 15
is_signed . . . 16
is_snan . . . 16
is_subnormal . . . 17
is_zero . . . 17
local_decimal_context . . . 18
mpdecimal_version . . . 18
NA_decimal_ . . . 19
normalize . . . 19
number_class . . . 20
quantize . . . 20
same_quantum . . . 21
set_decimal_context . . . 22
summary.decimal . . . 22
with_decimal_context . . . 23

Index **25**

`adjusted` *Compute the adjusted exponent*

Description

`adjusted()` returns the position of the most significant digit after accounting for the stored exponent. For a finite nonzero value, this is equivalent to the base-10 order of magnitude. The representation of zero retains its exponent, so differently scaled zeros can have different adjusted exponents.

Usage

```
adjusted(x)
```

Arguments

x	A decimal-compatible vector.

Value

An integer vector with the same length as `x`. Infinities, NaNs, and R missing values produce `NA_integer_`.

Examples

```
adjusted(decimal(c("123", "0.01")))
```

`arrow_as_data_frame` *Convert an Arrow table to a data frame, keeping decimal columns exact*

Description

`as.data.frame()` on an Arrow `Table` or `RecordBatch` converts a plain decimal field through `double`, which silently drops digits beyond the seventeenth. `arrow_as_data_frame()` converts those fields with `as_decimal()` instead, so they arrive as `decimal` vectors with the scale declared by their Arrow type. Every other column, including decimal columns this package wrote as its extension type, is converted by arrow in the usual way.

Usage

```
arrow_as_data_frame(x)
```

Arguments

x	An Arrow `Table` or `RecordBatch`.

Value

A data frame whose decimal columns are `decimal` vectors.

See Also

decimal_arrow for when a decimal field is plain and when it is the extension type.

Examples

```
if (requireNamespace("arrow", quietly = TRUE)) {
  tab <- arrow::arrow_table(
    id = 1:2,
    amount = arrow::Array$create(
      c("100.05", "99999999999999999999.99")
    )$cast(arrow::decimal128(25, 2))
  )
  arrow_as_data_frame(tab)
}
```

`arrow_decimal_type` *Arrow type for a decimal column*

Description

Builds the Arrow extension type this package writes `decimal` vectors as, with a chosen precision and scale rather than ones inferred from the values. Use it to pin the type of a column that will be appended to, for example through the `schema` argument of `arrow::arrow_table()` or the `type` argument of `arrow::as_arrow_array()`. The storage is `arrow::decimal128()` up to 38 digits of precision and `arrow::decimal256()` above that.

Usage

```
arrow_decimal_type(precision, scale = 0L)
```

Arguments

`precision`	Total number of digits, from 1 to 76.
`scale`	Number of fractional digits, at most `precision`. A negative scale is accepted for in-memory use, but Parquet cannot store it.

Value

An Arrow extension type.

See Also

decimal_arrow for how the type behaves.

Examples

```
if (requireNamespace("arrow", quietly = TRUE)) {
  arrow_decimal_type(20, 2)
  arrow::as_arrow_array(decimal("1.25"), type = arrow_decimal_type(20, 2))
}
```

`as_decimal.Array` *Convert a vector to decimal*

Description

`as_decimal()` is the conversion generic for decimal vectors. Character values are parsed exactly, integer values are converted exactly, and existing decimal vectors are returned unchanged, or rescaled when `scale` is given. Promotion to a finer shared scale is exact and context-free; reduction to a coarser scale is quantized. Double values are decoded from their exact IEEE 754 representation and therefore require an explicit or globally configured scale before quantization.

Usage

```
## S3 method for class 'Array'
as_decimal(x, scale = NULL)

## S3 method for class 'ChunkedArray'
as_decimal(x, scale = NULL)

as_decimal(x, scale = NULL)

## S3 method for class 'decimal'
as_decimal(x, scale = NULL)

## S3 method for class 'character'
as_decimal(x, scale = NULL)

## S3 method for class 'integer'
as_decimal(x, scale = NULL)

## S3 method for class 'numeric'
as_decimal(x, scale = NULL)

## Default S3 method:
as_decimal(x, scale = NULL)
```

Arguments

`x`	A decimal, character, integer, or double vector, or an Arrow `Array` or `ChunkedArray`.
`scale`	An integer scalar giving the number of fractional digits to store, or `NULL` to use the global default or input-specific inference.

Details

When `scale` is `NULL`, `getOption("decimal.default_scale")` is used for character, integer and double input when set. Without that option, character input uses the largest number of fractional

digits found in the input, integer input uses scale zero, and double input raises an error. A decimal vector or an Arrow decimal array already has a scale and keeps it; the option does not apply. Quantization uses the active `decimal_context()`.

An Arrow `Array` or `ChunkedArray` of any decimal type converts exactly, taking its scale from the Arrow type rather than from the values, so a chunk holding only whole numbers keeps its declared fractional digits. An Arrow integer array converts exactly at every width. Any other Arrow type converts to an R vector first and then follows the rules above. See decimal_arrow and `vignette("arrow-decimal-types")`.

Value

A `decimal` vector.

Examples

```
as_decimal(c("1.2", "3.45"))
```

`clear_decimal_flags` *Clear sticky decimal flags*

Description

Clear sticky decimal flags

Usage

```
clear_decimal_flags()
```

Value

The previously active sticky flags, invisibly.

Examples

```
clear_decimal_flags()
```

`decimal` *Construct decimal vectors*

Description

`decimal()` creates an immutable decimal vector backed by exact strings, at a single shared scale (number of fractional digits) for the whole vector. Character and integer inputs are parsed exactly. Promotion to a finer shared scale only appends zeros and is context-free; a requested coarser scale quantizes using the active context. Double inputs are decoded from their exact IEEE 754 binary value and require either an explicit `scale` or the `decimal.default_scale` option before quantization.

Usage

```
decimal(x = character(), scale = NULL)
```

Arguments

`x`	A decimal, character, integer, or double vector.
`scale`	An integer scalar giving the number of fractional digits to store. `NULL` uses the `decimal.default_scale` option when set, then falls back to input-specific inference. Negative values round into the integer part (e.g. `scale = -2` rounds to the nearest hundred).

Details

When `scale` is `NULL`, `getOption("decimal.default_scale")` is used when set. Otherwise, character input infers the largest number of fractional digits present in `x`, and integer input uses scale zero. A decimal `x` is returned unchanged. The option is a default for these conversions only: operations such as arithmetic with an integer, `round()`, `quantize()` and `summary()` never consult it. Because scale is a property of the vector, not the element, `decimal("1.2") == decimal("1.20")`, and combining them yields one uniformly scaled vector.

Value

A `decimal` vector.

Examples

```
decimal(c("1.20", "2.30"))
```

`decimal-summaries` *Summaries of decimal vectors*

Description

`sum()`, `prod()`, `mean()`, `min()`, `max()` and `range()` compute in decimal arithmetic and return a `decimal` vector. Integer vectors can be mixed in; doubles and strings are refused, as they are in arithmetic.

Usage

```
## S3 method for class 'decimal'
mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'decimal'
Summary(..., na.rm = FALSE)

## S3 method for class 'decimal'
range(x, ..., na.rm = FALSE, finite = FALSE)

## S3 method for class 'decimal'
min(x, ..., na.rm = FALSE)

## S3 method for class 'decimal'
max(x, ..., na.rm = FALSE)
```

Arguments

`x, ...`	`decimal` or integer vectors. A decimal must come first; see "Put a decimal first".
`trim`	The fraction of values to drop from each end before averaging.
`na.rm`	Whether to remove missing values and NaNs first.
`finite`	Not supported; `TRUE` is an error.

Details

A missing value gives `NA` unless `na.rm = TRUE`. A `NaN` gives `NaN`, and a signaling NaN raises `invalid_operation`. `na.rm = TRUE` removes both missing values and NaNs, because `is.na()` is true for a NaN.

`mean()` sums the values, then divides under the active context. A `trim` between 0 and 0.5 drops `floor(n * trim)` values from each end of the sorted vector before averaging, and a `trim` of 0.5 or more gives the median, as in `mean.default()`.

Value

A `decimal` vector: of length two for `range()`, otherwise of length one.

Put a decimal first

R picks the method for `sum()`, `prod()`, `min()`, `max()` and `range()` from the first argument alone. When that argument is not a decimal, R never calls these methods and treats a decimal as the text it stores. The package has no way to intercept this, nor to make it an error. In each case, put a decimal first or convert the first argument with `as_decimal()`:

- `min(10L, x)` compares text, so `min(10L, decimal("9"))` is `"10"`. Write `min(x, 10L)` or `min(as_decimal(10L), x)`.
- `max(2L, x)` and `range(2L, x)` return a character vector. Write `max(x, 2L)` or `range(as_decimal(2L), x)`.
- `sum(1L, x)` and `prod(1L, x)` fail with "invalid 'type' (character) of argument". Write `sum(x, 1L)` or `prod(as_decimal(1L), x)`.
- `pmax(2L, x)` and `pmin(2L, x)` return a character vector. Write `pmax(x, 2L)` or `pmax(as_decimal(2L), x)`.

`pmax()` and `pmin()` are not generic. With a decimal first they compare values, and stop with an error when a later argument has a finer scale than the first, because base R would give the result the first argument's scale: `pmax(decimal("10"), decimal("9.5"))` fails, while `pmax(as_decimal(decimal("10"), scale = 1), decimal("9.5"))` and `pmax(decimal("9.5"), decimal("10"))` work.

`any()` and `all()` are not supported for decimal vectors.

Examples

```
x <- decimal(c("9.50", "10", "2.25"))
sum(x)
max(x, 12L)
range(x)
mean(decimal(c("1", "2", "3", "100")), trim = 0.25)
```

---

decimal_arrow *Arrow interoperability*

Description

`decimal` vectors convert to and from Arrow decimal arrays exactly, in both directions, when the arrow package is installed.

Arrow to decimal. `as_decimal()` accepts an Arrow `Array` or `ChunkedArray`. A `decimal32()`, `decimal64()`, `decimal128()` or `decimal256()` array converts exactly, taking its scale from the Arrow type. An integer array of any width converts exactly too. Other Arrow types convert to the equivalent R vector first and follow the ordinary `as_decimal()` rules. `arrow_as_data_frame()` applies the same conversion to every decimal field of a `Table` or `RecordBatch`.

Decimal to Arrow. A `decimal` vector becomes a decimal field wherever arrow infers types: `arrow::as_arrow_array()`, `arrow::arrow_table()`, `arrow::write_parquet()` and `arrow::write_dataset()`. By default the field is an Arrow *extension type* whose storage is a real `decimal128()` or `decimal256()` with the vector's scale and a precision inferred from the values. Other readers see the storage, a plain decimal column. In R the column returns as a `decimal` vector on every read path, `as.data.frame()`,

`arrow::read_parquet()` and `dplyr::collect()` included. `arrow_decimal_type()` pins a precision and scale.

Arrow's compute engine does not operate on extension columns, so arrow-side arithmetic or filtering on such a column fails. To write a plain field instead, pass a plain Arrow decimal type as `type` to `arrow::as_arrow_array()`, or set the option below. A plain field comes back from arrow's own conversion as a `double` wearing the `decimal` class, because arrow reapplies the column's recorded R attributes; the package refuses to format such an object. Read those tables with `arrow_as_data_frame()`.

Infinities and NaNs have no Arrow decimal representation and raise an error on conversion to Arrow. A value wider than a pinned type raises an error of class `decimal_arrow_precision_error`, never a rounded or wrapped value. Negative zero has no Arrow representation either and becomes zero: `-0.00` reads back as `0.00`.

Arrow accepts a negative scale but Parquet does not, so a vector with a negative scale is written at scale 0, which holds the same values exactly, and reads back with scale 0. A negative scale pinned through `type` is kept in memory, but `arrow::write_parquet()` rejects such a column only after it has opened, and so emptied, the destination file; the same holds for a table already carrying a decimal field whose scale exceeds its precision.

Known issue

combining columns of different types: Combining decimal extension columns that arrow typed differently, with different scales or storage widths, can silently misread values (#37). This includes `arrow::concat_arrays()`, `arrow::concat_tables()` and `arrow::open_dataset()` over parts written separately, followed by `collect()` or `arrange()`. arrow compares two of this package's extension types by name alone, so it can reinterpret one part's stored integers at the other part's scale: 3.125 reads back as 31.25.

Workaround: pin one type with `arrow_decimal_type()` for every part that will be appended or combined.

The package checks every chunk it converts and raises an error of class `decimal_arrow_chunk_error` for a column whose chunks differ in scale or width. Chunks that differ only in precision are combined exactly. A misread that arrow's own kernels have already produced, as in `arrow::concat_arrays()`, cannot be detected.

Options

`decimal.arrow_extension`: `TRUE` (the default) writes the extension type; `FALSE` writes plain decimal fields everywhere.

See Also

`vignette("arrow-decimal-types")`.

`decimal_context` *Create a decimal arithmetic context*

Description

Constructs a validated arithmetic context for native `mpdecimal` operations. The context controls precision, rounding, exponent limits, traps, sticky flags, and classification of normal versus subnormal values.

Usage

```
decimal_context(
  precision = 28L,
  rounding = "half_even",
  emax = 999999L,
  emin = -999999L,
  traps = decimal_default_traps(),
  flags = character(),
  clamp = FALSE
)
```

Arguments

`precision`	Integer scalar precision.
`rounding`	One of `"up"`, `"down"`, `"ceiling"`, `"floor"`, `"half_up"`, `"half_down"`, `"half_even"`, or `"05up"`.
`emax`	Integer scalar maximum exponent.
`emin`	Integer scalar minimum exponent.
`traps`	Character vector of trapped signals. Trapped signals raise an error; all other raised signals are recorded as flags and, unless `options(decimal.report_flags = FALSE)`, reported as warnings.
`flags`	Character vector of sticky signal flags.
`clamp`	Logical scalar clamp mode.

Details

Each operation may raise one or more signals (see `decimal_flags()`). Their disposition depends on `traps`:

- A raised signal that is in `traps` stops the operation with an error.
- Any other raised signal is recorded as a sticky flag and, by default, also surfaced as a warning of class `decimal_flags_warning`.

The warnings are purely informational; sticky flags accumulate either way. Set `options(decimal.report_flags = FALSE)` to silence them and rely on `decimal_flags()` alone.

Public signal names are `"clamped"`, `"division_by_zero"`, `"inexact"`, `"invalid_operation"`, `"overflow"`, `"rounded"`, `"subnormal"`, and `"underflow"`. The standard `invalid_operation` condition groups lower-level invalid subconditions such as undefined division (`0 / 0`).

A few operations are exempt from the warning because `inexact`/`rounded` is their guaranteed, expected outcome rather than a surprise: `quantize()` (and `round()`/`signif()`, built on it), and `sqrt()`, `exp()`, `log()`, and `log10()`, which are irrational for nearly every input. These still accumulate sticky flags as usual.

Value

A `decimal_context` object.

Examples

```
decimal_context(precision = 10L)
decimal_context(precision = 3L, rounding = "floor", traps = character())
```

`decimal_flags` *Read sticky decimal flags*

Description

Sticky flags accumulate the signals raised by operations that were not trapped (see `decimal_context()`). They persist until `clear_decimal_flags()` is called. By default the same non-trapped signals are also reported as warnings as they occur; set `options(decimal.report_flags = FALSE)` to silence the warnings and inspect flags only through this function.

Usage

```
decimal_flags()
```

Value

A character vector of active sticky flags.

Examples

```
decimal_flags()
```

decimal_from_double *Controlled conversion from double*

Description

Decodes each IEEE 754 double to its exact decimal value, then quantizes that value to the requested scale. This differs from parsing a character literal such as `"0.1"`. Because the exact binary value of a double can require dozens of fractional digits, a scale must be given explicitly or configured with `options(decimal.default_scale = )`. Quantization uses the active context's rounding and trap settings.

Usage

```
decimal_from_double(x, scale = NULL)
```

Arguments

`x`	A double vector.
`scale`	An integer scalar giving the number of fractional digits to store, or `NULL` to use `getOption("decimal.default_scale")`.

Value

A `decimal` vector.

Examples

```
decimal_from_double(0.1, scale = 20)
decimal_from_double(c(0.5, 0.25), scale = 2)
```

fma *Fused multiply-add*

Description

`fma()` computes `x * y + z` with a single final rounding step. This can be more accurate than evaluating multiplication and addition separately under a limited-precision context. The arguments follow normal vctrs recycling rules.

Usage

```
fma(x, y, z)
```

Arguments

`x, y, z`	Decimal-compatible vectors.

Value

A `decimal` vector.

Examples

```
fma(decimal("2"), decimal("3"), decimal("4"))
```

---

get_decimal_context *Get the active decimal arithmetic context*

Description

Get the active decimal arithmetic context

Usage

```
get_decimal_context()
```

Value

A `decimal_context` object.

Examples

```
get_decimal_context()
```

---

is_decimal *Test whether an object is a decimal vector*

Description

`is_decimal()` reports whether `x` inherits from the `decimal` vector class. It does not attempt to parse or convert other objects.

Usage

```
is_decimal(x)
```

Arguments

`x`	An object to test.

Value

A single logical value.

Examples

```
is_decimal(decimal("1.5"))
```

is_normal *Identify normal decimal values*

Description

`is_normal()` reports whether each finite, nonzero value is normal under the active decimal context. Normality depends on the context's exponent limits and precision.

Usage

```
is_normal(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A logical vector with the same length as `x`.

Examples

```
is_normal(decimal(c("1", "0", "Infinity")))
```

is_qnan *Identify quiet NaN values*

Description

`is_qnan()` identifies quiet not-a-number values. Signaling NaNs and R missing values are not quiet NaNs.

Usage

```
is_qnan(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A logical vector with the same length as `x`.

Examples

```
is_qnan(decimal(c("NaN", "sNaN", "1")))
```

is_signed *Identify values with a negative sign*

Description

`is_signed()` inspects the stored sign bit rather than comparing with zero. It therefore identifies negative zero as signed.

Usage

```
is_signed(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A logical vector with the same length as `x`.

Examples

```
is_signed(decimal(c("-2", "2", "-0", "0")))
```

is_snan *Identify signaling NaN values*

Description

`is_snan()` identifies signaling not-a-number values without performing an arithmetic operation or raising the `invalid_operation` signal.

Usage

```
is_snan(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A logical vector with the same length as `x`.

Examples

```
is_snan(decimal(c("sNaN", "NaN", "1")))
```

is_subnormal *Identify subnormal decimal values*

Description

`is_subnormal()` reports whether each finite, nonzero value is subnormal under the active decimal context. A value can therefore be subnormal in one context and normal in another.

Usage

```
is_subnormal(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A logical vector with the same length as `x`.

Examples

```
x <- decimal("0.001")
with_decimal_context(
  decimal_context(precision = 3L, emin = -2L),
  is_subnormal(x)
)
```

is_zero *Identify decimal zeros*

Description

`is_zero()` identifies both positive and negative zero, regardless of the vector's scale.

Usage

```
is_zero(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A logical vector with the same length as `x`.

Examples

```
is_zero(decimal(c("0.00", "-0", "1")))
```

`local_decimal_context` *Install a decimal context for the current scope*

Description

Install a decimal context for the current scope

Usage

```
local_decimal_context(x, .local_envir = parent.frame())
```

Arguments

`x`	A `decimal_context` object or compatible list.
`.local_envir`	Environment whose scope should control restoration.

Value

`x`, invisibly.

Examples

```
f <- function() {
  local_decimal_context(decimal_context(precision = 2L, traps = character()))
  decimal("1.234") + decimal("0")
}
f()
```

`mpdecimal_version` *Report the bundled mpdecimal runtime version*

Description

Returns the version string reported by the bundled `mpdecimal` library that was loaded with the package DLL.

Usage

```
mpdecimal_version()
```

Value

A length-one character vector.

Examples

```
mpdecimal_version()
```

`NA_decimal_` *Missing decimal scalar*

Description

A length-one missing decimal value.

Usage

```
NA_decimal_
```

Value

A length-one `decimal` vector containing the typed R missing value.

Examples

```
c(decimal("1"), NA_decimal_)
```

`normalize` *Remove unnecessary trailing zeros*

Description

`normalize()` reduces each finite value to its shortest equivalent decimal representation, then chooses the finest scale required by any element so the result remains a valid shared-scale decimal vector. Special values pass through unchanged.

Usage

```
normalize(x)
```

Arguments

`x`	A decimal-compatible vector.

Value

A normalized `decimal` vector.

Examples

```
normalize(decimal(c("1.2300", "1.2")))
```

number_class *Classify decimal values*

Description

number_class() returns the General Decimal Arithmetic class of each value. Possible finite classes include "+Normal", "-Normal", "+Subnormal", "-Subnormal", "+Zero", and "-Zero"; infinities and NaNs have their corresponding class names. Normal and subnormal classes depend on the active decimal context.

Usage

```
number_class(x)
```

Arguments

x	A decimal-compatible vector.

Value

A character vector with the same length as x. R missing values produce NA_character_.

Examples

```
number_class(decimal(c("1", "-0", "Infinity", "NaN")))
```

quantize *Quantize decimal values to a scale*

Description

quantize() rounds each value in x to the scale declared by quantum. The operation uses the active context's rounding mode, updates sticky flags, and raises any enabled traps. The arguments follow normal vctrs recycling rules.

Usage

```
quantize(x, quantum)
```

Arguments

x	A decimal-compatible vector to quantize.
quantum	A decimal-compatible vector whose shared scale determines the result scale.

Details

Reducing the scale is the explicit purpose of `quantize()` (and of `round()` and `signif()`, both implemented on top of it), so the `inexact` and `rounded` signals this commonly raises are not reported as warnings the way other operations' signals are (see `decimal_context()`); they still accumulate as sticky flags.

Value

A `decimal` vector with the shared scale of `quantum`.

Examples

```
quantize(decimal("1.23456"), decimal("0.01"))
```

`same_quantum` *Compare decimal vector scales*

Description

`same_quantum()` tests whether `x` and `y` have the same shared vector scale. Scale is a vector-level property in this package, so every recycled element comparison receives the same result.

Usage

```
same_quantum(x, y)
```

Arguments

`x, y`	Decimal-compatible vectors.

Value

A logical vector with the common recycled size of `x` and `y`.

Examples

```
same_quantum(decimal("1.00"), decimal("2.0"))
```

set_decimal_context *Set the active decimal arithmetic context*

Description

Set the active decimal arithmetic context

Usage

```
set_decimal_context(x)
```

Arguments

x	A decimal_context object or compatible list.

Value

The previously active decimal_context, invisibly.

Examples

```
old <- set_decimal_context(decimal_context(precision = 5L))
set_decimal_context(old)
```

summary.decimal *Summarize a decimal vector*

Description

The six-number summary that summary() gives for a numeric vector, computed in exact decimal arithmetic and returned as a decimal vector rather than a vector of doubles.

Usage

```
## S3 method for class 'decimal'
summary(object, ..., maxsum = 100L, digits = NULL)
```

Arguments

object	A decimal vector.
...	These dots must be empty.
maxsum, digits	Accepted for compatibility with summary.data.frame(), which passes them to every column, and ignored. digits in particular is not honored: rounding an exact decimal for display is the surprise this package exists to avoid.

Details

Missing values are removed before the statistics are computed and reported as an `NA's` entry, matching `summary.default()`. As in base R, `NaN` counts as missing here, because `is.na()` is true for it.

The quartiles use the type 7 definition, the default of `stats::quantile()`. A quantile that falls between two elements is interpolated, and the mean divides by the number of elements, so both run under the active decimal context and may raise `inexact` and `rounded` signals like any other division. The minimum, the maximum, and any quantile that lands exactly on an element are always exact.

Because a `decimal` vector carries one shared scale, every entry is padded to the widest one present – including the `NA's` count, which is a count rather than a measured value. Interpolating a quartile can need more digits than the input carries, which widens that shared scale.

Interpolating between `-Infinity` and `Infinity` is an invalid operation, and the default context traps it, so summarizing a vector that spans both signed infinities raises an error rather than returning `NaN` quartiles. That is the same error `decimal("Infinity") - decimal("Infinity")` raises. Clear the trap with `with_decimal_context()` to get base R's `NaN` instead.

Value

A named `decimal` vector holding `Min.`, `1st Qu.`, `Median`, `Mean`, `3rd Qu.` and `Max.`, followed by `NA's` when the input contains missing values.

Examples

```
summary(decimal(c("1.25", "2.50", "3.75", "10.00")))

# Missing values are counted, not propagated.
summary(decimal(c("1.5", NA, "2.5")))
```

---

with_decimal_context *Use a decimal context within a block*

Description

Use a decimal context within a block

Usage

```
with_decimal_context(x, code)
```

Arguments

`x`	A `decimal_context` object or compatible list.
`code`	Code evaluated with `x` installed as the active context.

Value

The result of `code`.

Examples

```
with_decimal_context(
  decimal_context(precision = 3L, traps = character()),
  decimal("1.25") + decimal("0")
)
```

Index

adjusted, 2
arrow_as_data_frame, 3
arrow_as_data_frame(), *9, 10*
arrow_decimal_type, 4
arrow_decimal_type(), *10*
as_decimal *(*as_decimal.Array*)*, 5
as_decimal(), *3*, *9*
as_decimal.Array, 5

clear_decimal_flags, 6
clear_decimal_flags(), *12*

decimal, 7
decimal-summaries, 8
decimal_arrow, *3*, *4*, *6*, 9
decimal_context, 11
decimal_context(), *6*, *12*, *21*
decimal_flags, 12
decimal_flags(), *11*, *12*
decimal_from_double, 13

fma, 13

get_decimal_context, 14

is.na(), *8*, *23*
is_decimal, 14
is_normal, 15
is_qnan, 15
is_signed, 16
is_snan, 16
is_subnormal, 17
is_zero, 17

local_decimal_context, 18

max.decimal *(*decimal-summaries*)*, 8
mean.decimal *(*decimal-summaries*)*, 8
mean.default(), *8*
min.decimal *(*decimal-summaries*)*, 8
mpdecimal_version, 18

NA_decimal_, 19
normalize, 19
number_class, 20

pmax(), *9*
pmin(), *9*

quantize, 20
quantize(), *7*, *12*

range.decimal *(*decimal-summaries*)*, 8
round(), *7*

same_quantum, 21
set_decimal_context, 22
stats::quantile(), *23*
summary(), *7*, *22*
summary.data.frame(), *22*
Summary.decimal *(*decimal-summaries*)*, 8
summary.decimal, 22
summary.default(), *23*

with_decimal_context, 23
with_decimal_context(), *23*