

# Package ‘wkutils’

January 11, 2021

**Title** Utilities for Well-Known Geometry Vectors

**Version** 0.1.1

**Description** Provides extra utilities for well-known formats in the 'wk' package that are outside the scope of that package. Utilities to parse coordinates from data frames, plot well-known geometry vectors, extract meta information from well-known geometry vectors, and calculate bounding boxes are provided.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**Imports** wk (>= 0.3.1), Rcpp, tibble, vctrs

**LinkingTo** wk, Rcpp

**Suggests** testthat

**URL** <https://paleolimbot.github.io/wkutils/>,  
<https://github.com/paleolimbot/wkutils>

**BugReports** <https://github.com/paleolimbot/wkutils/issues>

**NeedsCompilation** yes

**Author** Dewey Dunnington [aut, cre] (<<https://orcid.org/0000-0002-9415-4582>>)

**Maintainer** Dewey Dunnington <[dewey@fishandwhistle.net](mailto:dewey@fishandwhistle.net)>

**Repository** CRAN

**Date/Publication** 2021-01-11 15:10:02 UTC

## R topics documented:

coords_point_translate_wkt . . . . .	2
wkb_coords . . . . .	4
wkb_debug . . . . .	5
wkb_draw_points . . . . .	5

wkb_meta . . . . .	7
wkb_ranges . . . . .	8
wkt_grob . . . . .	9
wkt_has_missing . . . . .	10
wkt_plot . . . . .	11
wkt_set_srid . . . . .	12
wkt_unnest . . . . .	13

<b>Index</b>	<b>14</b>
--------------	-----------

---

coords\_point\_translate\_wkt

*Parse coordinates into well-known formats*

---

### Description

These functions provide the reverse function of `wkt_coords()` and company: they parse vectors of coordinate values into well-known formats. Polygon rings are automatically closed, as closed rings are assumed or required by many parsers.

### Usage

```
coords_point_translate_wkt(x, y, z = NA, m = NA, precision = 16, trim = TRUE)
```

```
coords_point_translate_wkb(
  x,
  y,
  z = NA,
  m = NA,
  endian = wk::wk_platform_endian(),
  buffer_size = 2048
)
```

```
coords_linestring_translate_wkt(
  x,
  y,
  z = NA,
  m = NA,
  feature_id = 1L,
  precision = 16,
  trim = TRUE
)
```

```
coords_linestring_translate_wkb(
  x,
  y,
  z = NA,
  m = NA,

```

```

    feature_id = 1L,
    endian = wk::wk_platform_endian(),
    buffer_size = 2048
  )

coords_polygon_translate_wkt(
  x,
  y,
  z = NA,
  m = NA,
  feature_id = 1L,
  ring_id = 1L,
  precision = 16,
  trim = TRUE
)

coords_polygon_translate_wkb(
  x,
  y,
  z = NA,
  m = NA,
  feature_id = 1L,
  ring_id = 1L,
  endian = wk::wk_platform_endian(),
  buffer_size = 2048
)

```

### Arguments

<code>x, y, z, m</code>	Vectors of coordinate values
<code>precision</code>	The rounding precision to use when writing (number of decimal places).
<code>trim</code>	Trim unnecessary zeroes in the output?
<code>endian</code>	For WKB writing, 0 for big endian, 1 for little endian. Defaults to <code>wk_platform_endian()</code> (slightly faster).
<code>buffer_size</code>	For WKB writing, the initial buffer size to use for each feature, in bytes. This will be extended when needed, but if you are calling this repeatedly with huge geometries, setting this value to a larger number may result in less copying.
<code>feature_id, ring_id</code>	Vectors for which a change in sequential values indicates a new feature or ring. Use <code>factor()</code> to convert from a character vector.

### Value

`*_translate_wkt()` returns a character vector of well-known text; `*_translate_wkb()` returns a list of raw vectors.

### Examples

```
coords_point_translate_wkt(1:3, 2:4)
```

```
coords_linestring_translate_wkt(1:5, 2:6, feature_id = c(1, 1, 1, 2, 2))
coords_polygon_translate_wkt(c(0, 10, 0), c(0, 0, 10))
```

---

wkb\_coords

*Extract coordinates from well-known geometries*


---

## Description

These functions are optimised for graphics output, which in R require flat coordinate structures. See [graphics::points\(\)](#), [graphics::lines\(\)](#), and [graphics::polypath\(\)](#) for how to send these to a graphics device, or [grid::pointsGrob\(\)](#), [grid::linesGrob\(\)](#), and [grid::pathGrob\(\)](#) for how to create graphical objects using this output.

## Usage

```
wkb_coords(wkb, sep_na = FALSE)
```

```
wkt_coords(wkt, sep_na = FALSE)
```

## Arguments

wkb	A <code>list()</code> of <code>raw()</code> vectors, such as that returned by <code>sf::st_as_binary()</code> .
sep_na	Use TRUE to separate geometries and linear rings with a row of NAs. This is useful for generating output that can be fed directly to <a href="#">graphics::polypath()</a> or <a href="#">graphics::lines()</a> without modification.
wkt	A character vector containing well-known text.

## Value

A data.frame with columns:

- `feature_id`: The index of the top-level feature
- `part_id`: The part identifier, guaranteed to be unique for every simple geometry (including those contained within a multi-geometry or collection)
- `ring_id`: The ring identifier, guaranteed to be unique for every ring.
- `x, y, z, m`: Coordinate values (both absence and nan are recorded as NA)

## Examples

```
text <- c("LINESTRING (0 1, 19 27)", "LINESTRING (-1 -1, 4 10)")
wkt_coords(text)
wkt_coords(text, sep_na = TRUE)
```

---

wkb_debug	<i>Debug well-known geometry</i>
-----------	----------------------------------

---

**Description**

Prints the raw calls to the `WKBGeometryHandler()`. Useful for writing custom C++ handlers and debugging read problems.

**Usage**

```
wkb_debug(wkb)
```

```
wkt_debug(wkt)
```

```
wkt_streamer_debug(wkt)
```

**Arguments**

`wkb` A `list()` of `raw()` vectors, such as that returned by `sf::st_as_binary()`.

`wkt` A character vector containing well-known text.

**Value**

The input, invisibly

**Examples**

```
wkt_debug("MULTIPOLYGON (((0 0, 10 0, 0 10, 0 0)))")
wkt_streamer_debug("MULTIPOLYGON (((0 0, 10 0, 0 10, 0 0)))")
wkb_debug(
  wk::wkt_translate_wkb(
    "MULTIPOLYGON (((0 0, 10 0, 0 10, 0 0)))"
  )
)
```

---

wkb_draw_points	<i>Draw well-known geometries</i>
-----------------	-----------------------------------

---

**Description**

These functions send well-known geometry vectors to a graphics device using `graphics::points()`, `graphics::lines()`, and `graphics::polypath()`. These are minimal wrappers aimed at developers who need to visualize test data: they do not check geometry type and are unlikely to work with vectorized graphical parameters in . . . . Use the `wk*_plot_new()` functions to initialize a plot using the extent of all coordinates in the vector.

**Usage**

```

wkb_draw_points(wkb, ...)

wkt_draw_points(wkt, ...)

wkb_draw_lines(wkb, ...)

wkt_draw_lines(wkt, ...)

wkb_draw_polypath(wkb, ..., rule = "evenodd")

wkt_draw_polypath(wkt, ..., rule = "evenodd")

wkb_plot_new(
  wkb,
  ...,
  asp = 1,
  xlab = "",
  ylab = "",
  main = deparse(substitute(wkb))
)

wkt_plot_new(
  wkt,
  ...,
  asp = 1,
  xlab = "",
  ylab = "",
  main = deparse(substitute(wkt))
)

```

**Arguments**

wkb	A list() of <code>raw()</code> vectors, such as that returned by <code>sf::st_as_binary()</code> .
...	Passed to <code>graphics::points()</code> , <code>graphics::lines()</code> , or <code>graphics::polypath()</code>
wkt	A character vector containing well-known text.
rule	Passed to <code>graphics::polypath()</code>
asp, xlab, ylab, main	Passed to <code>graphics::plot()</code> to initialize a new plot.

**Value**

The input, invisibly

**Examples**

```
x <- "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"
```

```
wkt_plot_new(x)
wkt_draw_polypath(x, col = "grey90")
wkt_draw_lines(x, col = "red")
wkt_draw_points(x)
```

---

wkb\_meta

*Extract meta information*


---

## Description

Extract meta information

## Usage

```
wkb_meta(wkb, recursive = FALSE)
wkt_meta(wkt, recursive = FALSE)
wkt_streamer_meta(wkt, recursive = FALSE)
wk_geometry_type(type_id)
wk_geometry_type_id(type)
```

## Arguments

wkb	A <code>list()</code> of <code>raw()</code> vectors, such as that returned by <code>sf::st_as_binary()</code> .
recursive	Pass TRUE to recurse into multi-geometries and collections to extract meta of sub-geometries
wkt	A character vector containing well-known text.
type_id	An integer version of the geometry type
type	A string version of the geometry type (e.g., point, linestring, polygon, multi-point, multilinestring, multipolygon, geometrycollection)

## Value

A `data.frame` with columns:

- `feature_id`: The index of the top-level feature
- `nest_id`: The recursion level (if feature is a geometry collection)
- `part_id`: The part index (if nested within a multi-geometry or collection)
- `type_id`: The type identifier (see `wk_geometry_type()`)
- `size`: For points and linestrings the number of points, for polygons the number of rings, and for mutli-geometries and collection types, the number of child geometries.
- `srid`: The spatial reference identifier as an integer

## Examples

```
wkt_meta("POINT (30 10)")
wkt_meta("GEOMETRYCOLLECTION (POINT (30 10))", recursive = FALSE)
wkt_meta("GEOMETRYCOLLECTION (POINT (30 10))", recursive = TRUE)
```

---

wkb\_ranges

*Extract ranges information*

---

## Description

This is intended to behave the same as `range()`, returning the minimum and maximum x, y, z, and m coordinate values.

## Usage

```
wkb_ranges(wkb, na.rm = FALSE, finite = FALSE)
wkt_ranges(wkt, na.rm = FALSE, finite = FALSE)
wkb_feature_ranges(wkb, na.rm = FALSE, finite = FALSE)
wkt_feature_ranges(wkt, na.rm = FALSE, finite = FALSE)
```

## Arguments

wkb	A <code>list()</code> of <code>raw()</code> vectors, such as that returned by <code>sf::st_as_binary()</code> .
na.rm	Pass TRUE to not consider missing (nan) values
finite	Pass TRUE to only consider finite (non-missing, non-infinite) values.
wkt	A character vector containing well-known text.

## Value

A data.frame with columns:

- xmin, ymin, zmin, and mmin: Minimum coordinate values
- xmax, ymax, zmax, and mmax: Maximum coordinate values

## Examples

```
wkt_ranges("POINT (30 10)")
```



**Description**

Using `wkt_meta()` and `wkt_coords()`, these functions create graphical objects using the grid package. Vectors that contain geometries of a single dimension are efficiently packed into a `grid::pointsGrob()`, `grid::polylineGrob()`, or `grid::pathGrob()`. Vectors with mixed types and nested collections are encoded less efficiently using a `grid::gTree()`.

**Usage**

```
wkt_grob(
  wkt,
  ...,
  rule = "evenodd",
  default.units = "native",
  name = NULL,
  vp = NULL
)
```

```
wkb_grob(
  wkt,
  ...,
  rule = "evenodd",
  default.units = "native",
  name = NULL,
  vp = NULL
)
```

**Arguments**

<code>wkt</code>	A character vector containing well-known text.
<code>...</code>	Graphical parameters passed to <code>grid::gpar()</code> . These are recycled along the input. Dynamic dots (e.g., <code>!!!</code> ) are supported.
<code>rule</code>	Use "winding" if polygon rings are correctly encoded with a winding direction.
<code>default.units</code>	Coordinate units, which may be defined by the viewport (see <code>grid::unit()</code> ). Defaults to native.
<code>name, vp</code>	Passed to <code>grid::pointsGrob()</code> , <code>grid::polylineGrob()</code> , <code>grid::pathGrob()</code> , or <code>grid::gTree()</code> depending on the types of geometries in the input.

**Value**

A [graphical object](#)

## Examples

```
grid::grid.newpage()
grid::grid.draw(wkt_grob("POINT (0.5 0.5)", pch = 16, default.units = "npc"))
```

---

wkt\_has\_missing      *Test well-known geometries for missing and non-finite coordinates*

---

## Description

Note that EMPTY geometries are considered finite and non-missing. Use the size column of [wkt\\_meta\(\)](#) to test for empty geometries.

## Usage

```
wkt_has_missing(wkt)
```

```
wkb_has_missing(wkb)
```

```
wkt_is_finite(wkt)
```

```
wkb_is_finite(wkb)
```

## Arguments

wkt            A character vector containing well-known text.

wkb            A list() of [raw\(\)](#) vectors, such as that returned by `sf::st_as_binary()`.

## Value

A logical vector with the same length as the input.

## Examples

```
wkt_has_missing("POINT (0 1)")
wkt_has_missing("POINT (nan nan)")
wkt_has_missing("POINT (inf inf)")
```

```
wkt_is_finite("POINT (0 1)")
wkt_is_finite("POINT (nan nan)")
wkt_is_finite("POINT (inf inf)")
```

---

wkt\_plot *Plot well-known geometry vectors*

---

### Description

These plot functions are intended to help debug geometry vectors, and are not intended to be high-performance.

### Usage

```
wkt_plot(
  x,
  ...,
  asp = 1,
  bbox = NULL,
  xlab = "",
  ylab = "",
  rule = "evenodd",
  add = FALSE
)
```

```
wkb_plot(
  x,
  ...,
  asp = 1,
  bbox = NULL,
  xlab = "",
  ylab = "",
  rule = "evenodd",
  add = FALSE
)
```

### Arguments

x	A <a href="#">wkt()</a> or <a href="#">wkb()</a> vector.
...	Passed to plotting functions for features: <a href="#">graphics::points()</a> for point and multipoint geometries, <a href="#">graphics::lines()</a> for linestring and multilinestring geometries, and <a href="#">graphics::polypath()</a> for polygon and multipolygon geometries.
asp, xlab, ylab	Passed to <a href="#">graphics::plot()</a>
bbox	The limits of the plot in the form returned by <a href="#">wkt_ranges()</a> .
rule	The rule to use for filling polygons (see <a href="#">graphics::polypath()</a> )
add	Should a new plot be created, or should x be added to the existing plot?

### Value

x, invisibly

**Examples**

```
wkt_plot("POINT (30 10)")
```

---

wkt_set_srid	<i>Modify well-known geometries</i>
--------------	-------------------------------------

---

**Description**

Modify well-known geometries

**Usage**

```
wkt_set_srid(wkt, srid, precision = 16, trim = TRUE)
```

```
wkb_set_srid(wkb, srid)
```

```
wkt_set_z(wkt, z, precision = 16, trim = TRUE)
```

```
wkb_set_z(wkb, z)
```

```
wkt_transform(wkt, trans, precision = 16, trim = TRUE)
```

```
wkb_transform(wkb, trans)
```

**Arguments**

wkt	A character vector containing well-known text.
srid	An integer spatial reference identifier with a user-defined meaning. Use NA to unset this value.
precision	The rounding precision to use when writing (number of decimal places).
trim	Trim unnecessary zeroes in the output?
wkb	A list() of raw() vectors, such as that returned by sf::st_as_binary().
z	A Z value that will be assigned to every coordinate in each feature. Use NA to unset this value.
trans	A 3x3 transformation matrix that will be applied to all coordinates in the input.

**Value**

An unclassed well-known vector with the same type as the input.

**Examples**

```
wkt_set_srid("POINT (30 10)", 1234)
wkt_set_z("POINT (30 10)", 1234)
wkt_transform(
  "POINT (0 0)",
  # translation +12 +13
  matrix(c(1, 0, 0, 0, 1, 0, 12, 13, 1), ncol = 3)
)
```

---

wkt_unnest	<i>Flatten nested geometry structures</i>
------------	-------------------------------------------

---

**Description**

Flatten nested geometry structures

**Usage**

```
wkt_unnest(wkt, keep_empty = FALSE, keep_multi = TRUE, max_depth = 1)
```

```
wkb_unnest(wkb, keep_empty = FALSE, keep_multi = TRUE, max_depth = 1)
```

**Arguments**

wkt	A character vector containing well-known text.
keep_empty	If TRUE, a GEOMETRYCOLLECTION EMPTY is left as-is rather than collapsing to length 0.
keep_multi	If TRUE, MULTI* geometries are not expanded to sub-features.
max_depth	The maximum recursive GEOMETRYCOLLECTION depth to unnest.
wkb	A list() of raw() vectors, such as that returned by sf::st_as_binary().

**Value**

An unclassed vector with attribute lengths, which is an integer vector with the same length as the input denoting the length to which each feature was expanded.

**Examples**

```
wkt_unnest("GEOMETRYCOLLECTION (POINT (1 2), POINT (3 4))")
wkt_unnest("GEOMETRYCOLLECTION EMPTY")
wkt_unnest("GEOMETRYCOLLECTION EMPTY", keep_empty = TRUE)
```

# Index

`coords_linestring_translate_wkb`  
    (`coords_point_translate_wkt`), 2  
`coords_linestring_translate_wkt`  
    (`coords_point_translate_wkt`), 2  
`coords_point_translate_wkb`  
    (`coords_point_translate_wkt`), 2  
`coords_point_translate_wkt`, 2  
`coords_polygon_translate_wkb`  
    (`coords_point_translate_wkt`), 2  
`coords_polygon_translate_wkt`  
    (`coords_point_translate_wkt`), 2

`factor()`, 3

graphical object, 9  
`graphics::lines()`, 4–6, 11  
`graphics::plot()`, 6, 11  
`graphics::points()`, 4–6, 11  
`graphics::polypath()`, 4–6, 11  
`grid::gpar()`, 9  
`grid::gTree()`, 9  
`grid::linesGrob()`, 4  
`grid::pathGrob()`, 4, 9  
`grid::pointsGrob()`, 4, 9  
`grid::polylineGrob()`, 9  
`grid::unit()`, 9

`range()`, 8  
`raw()`, 4–8, 10, 12, 13

`wk_geometry_type` (`wkb_meta`), 7  
`wk_geometry_type()`, 7  
`wk_geometry_type_id` (`wkb_meta`), 7  
`wk_platform_endian()`, 3  
`wkb()`, 11  
`wkb_coords`, 4  
`wkb_debug`, 5  
`wkb_draw_lines` (`wkb_draw_points`), 5  
`wkb_draw_points`, 5  
`wkb_draw_polypath` (`wkb_draw_points`), 5  
`wkb_feature_ranges` (`wkb_ranges`), 8  
`wkb_grob` (`wkt_grob`), 9  
`wkb_has_missing` (`wkt_has_missing`), 10  
`wkb_is_finite` (`wkt_has_missing`), 10  
`wkb_meta`, 7  
`wkb_plot` (`wkt_plot`), 11  
`wkb_plot_new` (`wkb_draw_points`), 5  
`wkb_ranges`, 8  
`wkb_set_srid` (`wkt_set_srid`), 12  
`wkb_set_z` (`wkt_set_srid`), 12  
`wkb_transform` (`wkt_set_srid`), 12  
`wkb_unnest` (`wkt_unnest`), 13  
`wkt()`, 11  
`wkt_coords` (`wkb_coords`), 4  
`wkt_coords()`, 2, 9  
`wkt_debug` (`wkb_debug`), 5  
`wkt_draw_lines` (`wkb_draw_points`), 5  
`wkt_draw_points` (`wkb_draw_points`), 5  
`wkt_draw_polypath` (`wkb_draw_points`), 5  
`wkt_feature_ranges` (`wkb_ranges`), 8  
`wkt_grob`, 9  
`wkt_has_missing`, 10  
`wkt_is_finite` (`wkt_has_missing`), 10  
`wkt_meta` (`wkb_meta`), 7  
`wkt_meta()`, 9, 10  
`wkt_plot`, 11  
`wkt_plot_new` (`wkb_draw_points`), 5  
`wkt_ranges` (`wkb_ranges`), 8  
`wkt_ranges()`, 11  
`wkt_set_srid`, 12  
`wkt_set_z` (`wkt_set_srid`), 12  
`wkt_streamer_debug` (`wkb_debug`), 5  
`wkt_streamer_meta` (`wkb_meta`), 7  
`wkt_transform` (`wkt_set_srid`), 12  
`wkt_unnest`, 13