

# Package: wk (via r-universe)

October 11, 2024

**Title** Lightweight Well-Known Geometry Parsing

**Version** 0.9.3.9000

**Maintainer** Dewey Dunnington <dewey@fishandwhistle.net>

**Description** Provides a minimal R and C++ API for parsing well-known binary and well-known text representation of geometries to and from R-native formats. Well-known binary is compact and fast to parse; well-known text is human-readable and is useful for writing tests. These formats are useful in R only if the information they contain can be accessed in R, for which high-performance functions are provided here.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.2.3

**Suggests** testthat (>= 3.0.0), vctrs (>= 0.3.0), sf, tibble, readr

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

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

**Config/testthat/edition** 3

**Depends** R (>= 2.10)

**LazyData** true

**Repository** <https://paleolimbot.r-universe.dev>

**RemoteUrl** <https://github.com/paleolimbot/wk>

**RemoteRef** HEAD

**RemoteSha** 96c013260801dc52b2ed0766467bfdc77f0e207b

## Contents

|                                |    |
|--------------------------------|----|
| crc                            | 3  |
| crc_x                          | 4  |
| grd                            | 5  |
| grd_cell                       | 7  |
| grd_extract                    | 8  |
| grd_snap_next                  | 9  |
| grd_subset                     | 9  |
| grd_summary                    | 11 |
| grd_tile                       | 11 |
| grd_tile_template              | 12 |
| handle_wkt_without_vector_size | 13 |
| new_wk_crc                     | 14 |
| new_wk_grd                     | 14 |
| new_wk_rct                     | 15 |
| new_wk_wkb                     | 15 |
| new_wk_wkt                     | 16 |
| new_wk_xy                      | 16 |
| plot.wk_grd_xy                 | 17 |
| rct                            | 18 |
| rct_xmin                       | 19 |
| vctrs-methods                  | 20 |
| wkb                            | 21 |
| wkb_to_hex                     | 22 |
| wkb_translate_wkt              | 23 |
| wkt                            | 23 |
| wk_bbox                        | 24 |
| wk_chunk_strategy_single       | 25 |
| wk_count                       | 26 |
| wk_crs                         | 27 |
| wk_crs_equal                   | 28 |
| wk_crs_inherit                 | 28 |
| wk_crs_proj_definition         | 29 |
| wk_debug                       | 30 |
| wk_example                     | 31 |
| wk_flatten                     | 32 |
| wk_format                      | 33 |
| wk_handle.data.frame           | 34 |
| wk_handle.wk_crc               | 35 |
| wk_handle.wk_grd_xy            | 36 |
| wk_handle_slice.data.frame     | 37 |
| wk_identity                    | 38 |
| wk_is_geodesic                 | 39 |
| wk_linestring                  | 40 |
| wk_meta                        | 41 |
| wk_orient                      | 42 |
| wk_plot                        | 43 |

wk\_problems . . . . . 45  
 wk\_proj\_crs\_view . . . . . 46  
 wk\_set\_z . . . . . 46  
 wk\_transform . . . . . 47  
 wk\_translate.sfc . . . . . 48  
 wk\_trans\_affine . . . . . 48  
 wk\_trans\_explicit . . . . . 49  
 wk\_trans\_inverse . . . . . 50  
 wk\_vertices . . . . . 50  
 wk\_void . . . . . 51  
 wk\_writer.sfc . . . . . 52  
 xy . . . . . 53  
 xy\_x . . . . . 55

**Index** **56**

crc *2D Circle Vectors*

**Description**

2D Circle Vectors

**Usage**

```
crc(x = double(), y = double(), r = double(), crs = wk_crs_auto())

as_crc(x, ...)

## S3 method for class 'wk_crc'
as_crc(x, ...)

## S3 method for class 'matrix'
as_crc(x, ..., crs = NULL)

## S3 method for class 'data.frame'
as_crc(x, ..., crs = NULL)
```

**Arguments**

- x, y           Coordinates of the center
- r             Circle radius
- crs           A value to be propagated as the CRS for this vector.
- ...           Extra arguments passed to as\_crc().

**Value**

A vector along the recycled length of bounds.

**Examples**

```
crc(1, 2, 3)
```

---

crc\_x

*Circle accessors*

---

**Description**

Circle accessors

**Usage**

```
crc_x(x)
```

```
crc_y(x)
```

```
crc_center(x)
```

```
crc_r(x)
```

**Arguments**

x                    A `crc()` vector

**Value**

Components of the `crc()` vector

**Examples**

```
x <- crc(1, 2, r = 3)
crc_x(x)
crc_y(x)
crc_r(x)
crc_center(x)
```

grd

*Raster-like objects***Description**

`grd()` objects are just an array (any object with more than two `dim()`s) and a bounding box (a `rct()`, which may or may not have a `wk_crs()` attached). The ordering of the dimensions is y (indices increasing downwards), x (indices increasing to the right). This follows the ordering of `as.raster()/rasterImage()` and aligns with the printing of matrices.

**Usage**

```
grd(
  bbox = NULL,
  nx = NULL,
  ny = NULL,
  dx = NULL,
  dy = NULL,
  type = c("polygons", "corners", "centers")
)

grd_rct(data, bbox = rct(0, 0, dim(data)[2], dim(data)[1]))

grd_xy(data, bbox = rct(0, 0, dim(data)[2] - 1, dim(data)[1] - 1))

as_grd_rct(x, ...)

## S3 method for class 'wk_grd_rct'
as_grd_rct(x, ...)

## S3 method for class 'wk_grd_xy'
as_grd_rct(x, ...)

as_grd_xy(x, ...)

## S3 method for class 'wk_grd_xy'
as_grd_xy(x, ...)

## S3 method for class 'wk_grd_rct'
as_grd_xy(x, ...)
```

**Arguments**

`bbox` A `rct()` containing the bounds and CRS of the object. You can specify a `rct()` with `xmin > xmax` or `ymin > ymax` which will flip the underlying data and return an object with a normalized bounding box and data.

|                             |  |
|-----------------------------|--|
| <code>nx, ny, dx, dy</code> | Either a number of cells in the x- and y- directions or delta in the x- and y- directions (in which case <code>bbox</code> must be specified).   |
| <code>type</code>           | Use "polygons" to return a grid whose objects can be represented using an <code>rct()</code> ; use "centers" to return a grid whose objects are the center of the <code>rct()</code> grid; use "corners" to return a grid along the corners of <code>bbox</code> . |
| <code>data</code>           | An object with two or more dimensions. Most usefully, a matrix.  |
| <code>x</code>              | An object to convert to a grid   |
| <code>...</code>            | Passed to S3 methods   |

### Value

- `grd()` returns a `grd_rct()` for `type == "polygons"` or a `grd_xy()` otherwise.
- `grd_rct()` returns an object of class "wk\_grd\_rct".
- `grd_xy()` returns an object of class "wk\_grd\_xy".

### Examples

```
# create a grid with no data (just for coordinates)
(grid <- grd(nx = 2, ny = 2))
as_rct(grid)
as_xy(grid)
plot(grid, border = "black")

# more usefully, wraps a matrix or nd array + bbox
# approx volcano in New Zealand Transverse Mercator
bbox <- rct(
  5917000,      1757000 + 870,
  5917000 + 610, 1757000,
  crs = "EPSG:2193"
)
(grid <- grd_rct(volcano, bbox))

# these come with a reasonable default plot method for matrix data
plot(grid)

# you can set the data or the bounding box after creation
grid$bbox <- rct(0, 0, 1, 1)

# subset by indices or rct
plot(grid[1:2, 1:2])
plot(grid[c(start = NA, stop = NA, step = 2), c(start = NA, stop = NA, step = 2)])
plot(grid[rct(0, 0, 0.5, 0.5)])
```

grd\_cell

*Grid cell operators***Description**

Grid cell operators

**Usage**

```

grd_cell(grid, point, ..., snap = grd_snap_next)

grd_cell_range(
  grid,
  bbox = wk_bbox(grid),
  ...,
  step = 1L,
  snap = grd_snap_next
)

grd_cell_rct(grid, i, j = NULL, ...)

## S3 method for class 'wk_grd_rct'
grd_cell_rct(grid, i, j = NULL, ..., out_of_bounds = "keep")

## S3 method for class 'wk_grd_xy'
grd_cell_rct(grid, i, j = NULL, ..., out_of_bounds = "keep")

grd_cell_xy(grid, i, j = NULL, ...)

## S3 method for class 'wk_grd_rct'
grd_cell_xy(grid, i, j = NULL, ..., out_of_bounds = "keep")

## S3 method for class 'wk_grd_xy'
grd_cell_xy(grid, i, j = NULL, ..., out_of_bounds = "keep")

```

**Arguments**

|       |  |
|-------|--|
| grid  | A <a href="#">grd_xy()</a> , <a href="#">grd_rct()</a> , or other object implementing <code>grd_*</code> methods.  |
| point | A <a href="#">handleable</a> of points.  |
| ...   | Unused   |
| snap  | A function that transforms real-valued indices to integer indices (e.g., <a href="#">floor()</a> , <a href="#">ceiling()</a> , or <a href="#">round()</a> ). For <a href="#">grd_cell_range()</a> , a <code>list()</code> with exactly two elements to be called for the minimum and maximum index values, respectively. |
| bbox  | An <a href="#">rct()</a> object.   |
| step  | The difference between adjacent indices in the output  |

`i, j` 1-based index values. `i` indices correspond to decreasing `y` values; `j` indices correspond to increasing `x` values. Values outside the range `1:nrow|ncol(data)` will be censored to NA including 0 and negative values.

`out_of_bounds` One of 'keep', 'censor', 'discard', or 'squish'

### Value

- `grd_cell()`: returns a `list(i, j)` of index values corresponding to the input points and adjusted according to `snap`. Index values will be outside `dim(grid)` for points outside `wk_bbox(grid)` including negative values.
- `grd_cell_range()` returns a slice describing the range of indices in the `i` and `j` directions.
- `grd_cell_rct()` returns a `rct()` of the cell extent at `i, j`.
- `grd_cell_xy()` returns a `xy()` of the cell center at `i, j`.

### Examples

```
grid <- grd(nx = 3, ny = 2)
grd_cell(grid, xy(0.5, 0.5))
grd_cell_range(grid, grid$bbox)
grd_cell_rct(grid, 1, 1)
grd_cell_xy(grid, 1, 1)
```

---

grd\_extract

*Extract values from a grid*

---

### Description

Unlike `grd_subset()`, which subsets like a matrix, `grd_extract()` returns values.

### Usage

```
grd_extract(grid, i = NULL, j = NULL)
```

```
grd_extract_nearest(grid, point, out_of_bounds = c("censor", "squish"))
```

```
grd_data_extract(grid_data, i = NULL, j = NULL)
```

### Arguments

`grid` A `grd_xy()`, `grd_rct()`, or other object implementing `grd_*` methods.

`i, j` Index values as in `grd_subset()` except recycled to a common size.

`point` A `handleable` of points.

`out_of_bounds` One of 'keep', 'censor', 'discard', or 'squish'

`grid_data` The data member of a `grd()`. This is typically an array but can also be an S3 object with an array-like subset method. The `native raster` is special-cased as its subset method requires non-standard handling.



**Value**

A matrix or vector with two fewer dimensions than the input.

---

|               |                             |
|---------------|-----------------------------|
| grd_snap_next | <i>Index snap functions</i> |
|---------------|-----------------------------|

---

**Description**

These functions can be used in `grd_cell()` and `grd_cell_range()`. These functions differ in the way they round 0.5: `grd_snap_next()` always rounds up and `grd_snap_previous()` always rounds down. You can also use `floor()` and `ceiling()` as index snap functions.

**Usage**

```
grd_snap_next(x)
```

```
grd_snap_previous(x)
```

**Arguments**

`x`                      A vector of rescaled but non-integer indices

**Value**

A vector of integer indices

**Examples**

```
grd_snap_next(seq(0, 2, 0.25))
grd_snap_previous(seq(0, 2, 0.25))
```

---

|            |                            |
|------------|----------------------------|
| grd_subset | <i>Subset grid objects</i> |
|------------|----------------------------|

---

**Description**

The `grd_subset()` method handles the subsetting of a `grd()` in x-y space. Ordering of indices is not considered and logical indices are recycled silently along dimensions. The result of a `grd_subset()` is always a `grd()` of the same type whose relationship to x-y space has not changed.

**Usage**

```

grd_subset(grid, i = NULL, j = NULL, ...)

grd_crop(grid, bbox, ..., step = 1L, snap = NULL)

grd_extend(grid, bbox, ..., step = 1L, snap = NULL)

## S3 method for class 'wk_grd_rct'
grd_crop(grid, bbox, ..., step = 1L, snap = NULL)

## S3 method for class 'wk_grd_xy'
grd_crop(grid, bbox, ..., step = 1L, snap = NULL)

## S3 method for class 'wk_grd_rct'
grd_extend(grid, bbox, ..., step = 1L, snap = NULL)

## S3 method for class 'wk_grd_xy'
grd_extend(grid, bbox, ..., step = 1L, snap = NULL)

grd_data_subset(grid_data, i = NULL, j = NULL)

```

**Arguments**

|           |  |
|-----------|--|
| grid      | A <a href="#">grd_xy()</a> , <a href="#">grd_rct()</a> , or other object implementing <code>grd_*()</code> methods.  |
| i, j      | 1-based index values. i indices correspond to decreasing y values; j indices correspond to increasing x values. Values outside the range <code>1:nrow ncol(data)</code> will be censored to NA including 0 and negative values.  |
| ...       | Passed to subset methods   |
| bbox      | An <a href="#">rct()</a> object.   |
| step      | The difference between adjacent indices in the output  |
| snap      | A function that transforms real-valued indices to integer indices (e.g., <a href="#">floor()</a> , <a href="#">ceiling()</a> , or <a href="#">round()</a> ). For <a href="#">grd_cell_range()</a> , a <code>list()</code> with exactly two elements to be called for the minimum and maximum index values, respectively. |
| grid_data | The data member of a <a href="#">grd()</a> . This is typically an array but can also be an S3 object with an array-like subset method. The <a href="#">native raster</a> is special-cased as its subset method requires non-standard handling.   |

**Value**

A modified grid whose cell centres have not changed location as a result of the subset.

**Examples**

```

grid <- grd_rct(volcano)
grd_subset(grid, 1:20, 1:30)
grd_crop(grid, rct(-10, -10, 10, 10))
grd_extend(grid, rct(-10, -10, 10, 10))

```

---

|             |                         |
|-------------|-------------------------|
| grd_summary | <i>Grid information</i> |
|-------------|-------------------------|

---

**Description**

Grid information

**Usage**

```
grd_summary(grid)
```

**Arguments**

grid            A [grd\\_xy\(\)](#), [grd\\_rct\(\)](#), or other object implementing `grd_*()` methods.

**Value**

- `grd_summary()` returns a `list()` with components `xmin`, `ymin`, `xmax`, `ymax`, `nx`, `ny`, `dx`, `dy`, `width`, and `height`.

**Examples**

```
grd_summary(grd(nx = 3, ny = 2))
```

---

|          |                                      |
|----------|--------------------------------------|
| grd_tile | <i>Extract normalized grid tiles</i> |
|----------|--------------------------------------|

---

**Description**

Unlike [grd\\_tile\\_template\(\)](#), which returns a [grd\(\)](#) whose elements are the boundaries of the specified tiles with no data attached, [grd\\_tile\(\)](#) returns the actual tile with the data.

**Usage**

```
grd_tile(grid, level, i, j = NULL)

## S3 method for class 'wk_grd_rct'
grd_tile(grid, level, i, j = NULL)

## S3 method for class 'wk_grd_xy'
grd_tile(grid, level, i, j = NULL)
```

**Arguments**

|       |   |
|-------|---|
| grid  | A <a href="#">grd_xy()</a> , <a href="#">grd_rct()</a> , or other object implementing <code>grd_*()</code> methods.   |
| level | An integer describing the overview level. This is related to the step value by a power of 2 (i.e., a level of 1 indicates a step of 2, a level of 2 indicates a step of 4, etc.).   |
| i, j  | 1-based index values. i indices correspond to decreasing y values; j indices correspond to increasing x values. Values outside the range <code>1:nrow ncol(data)</code> will be censored to NA including 0 and negative values. |

**Value**

A [grd\\_subset\(\)](#)ed version

**Examples**

```
grid <- grd_rct(volcano)
plot(grd_tile(grid, 4, 1, 1))

plot(grd_tile(grid, 3, 1, 1), add = TRUE)
plot(grd_tile(grid, 3, 1, 2), add = TRUE)
plot(grd_tile(grid, 3, 2, 1), add = TRUE)
plot(grd_tile(grid, 3, 2, 2), add = TRUE)

grid <- as_grd_xy(grd_tile(grid, 4, 1, 1))
plot(grid, add = TRUE, pch = ".")
plot(grd_tile(grid, 3, 1, 1), add = TRUE, col = "green", pch = ".")
plot(grd_tile(grid, 3, 1, 2), add = TRUE, col = "red", pch = ".")
plot(grd_tile(grid, 3, 2, 1), add = TRUE, col = "blue", pch = ".")
plot(grd_tile(grid, 3, 2, 2), add = TRUE, col = "magenta", pch = ".")
```

---

grd\_tile\_template      *Compute overview grid tile*

---

**Description**

A useful workflow for raster data in a memory bounded environment is to chunk a grid into sections or tiles. These functions compute tiles suitable for such processing. Use [grd\\_tile\\_summary\(\)](#) to generate statistics for level values to choose for your application.

**Usage**

```
grd_tile_template(grid, level)

grd_tile_summary(grid, levels = NULL)
```

**Arguments**

|        |   |
|--------|---|
| grid   | A <a href="#">grd_xy()</a> , <a href="#">grd_rct()</a> , or other object implementing <code>grd_*</code> () methods.  |
| level  | An integer describing the overview level. This is related to the step value by a power of 2 (i.e., a level of 1 indicates a step of 2, a level of 2 indicates a step of 4, etc.). |
| levels | A vector of level values or NULL to use a sequence from 0 to the level that would result in a 1 x 1 grid.   |

**Value**

A [grd\(\)](#)

**Examples**

```
grid <- grd_rct(volcano)
grd_tile_summary(grid)
grd_tile_template(grid, 3)
```

---

handle\_wkt\_without\_vector\_size

*Test handlers for handling of unknown size vectors*

---

**Description**

Test handlers for handling of unknown size vectors

**Usage**

```
handle_wkt_without_vector_size(handleable, handler)
```

**Arguments**

|            |   |
|------------|---|
| handleable | A geometry vector (e.g., <a href="#">wkb()</a> , <a href="#">wkt()</a> , <a href="#">xy()</a> , <a href="#">rct()</a> , or <a href="#">sf::st_sfc()</a> ) for which <a href="#">wk_handle()</a> is defined. |
| handler    | A <a href="#">wk_handler</a> object.  |

**Examples**

```
handle_wkt_without_vector_size(wkt(), wk_vector_meta_handler())
```

---

new\_wk\_crc                      *S3 details for crc objects*

---

**Description**

S3 details for crc objects

**Usage**

```
new_wk_crc(x = list(x = double(), y = double(), r = double()), crs = NULL)
```

**Arguments**

|     |  |
|-----|--|
| x   | A <a href="#">crc()</a>                              |
| crs | A value to be propagated as the CRS for this vector. |

---

new\_wk\_grd                      *S3 details for grid objects*

---

**Description**

S3 details for grid objects

**Usage**

```
new_wk_grd(x, subclass = character())
```

**Arguments**

|          |                         |
|----------|-------------------------|
| x        | A <a href="#">grd()</a> |
| subclass | An optional subclass.   |

**Value**

An object inheriting from 'grd'

---

|            |                                   |
|------------|-----------------------------------|
| new_wk_rct | <i>S3 details for rct objects</i> |
|------------|-----------------------------------|

---

**Description**

S3 details for rct objects

**Usage**

```
new_wk_rct(
  x = list(xmin = double(), ymin = double(), xmax = double(), ymax = double()),
  crs = NULL
)
```

**Arguments**

|     |  |
|-----|--|
| x   | A <a href="#">rct()</a>                              |
| crs | A value to be propagated as the CRS for this vector. |

---

|            |                              |
|------------|------------------------------|
| new_wk_wkb | <i>S3 Details for wk_wkb</i> |
|------------|------------------------------|

---

**Description**

S3 Details for wk\_wkb

**Usage**

```
new_wk_wkb(x = list(), crs = NULL, geodesic = NULL)

validate_wk_wkb(x)

is_wk_wkb(x)
```

**Arguments**

|          |  |
|----------|--|
| x        | A (possibly) <a href="#">wkb()</a> vector  |
| crs      | A value to be propagated as the CRS for this vector.   |
| geodesic | TRUE if edges must be interpolated as geodesics when coordinates are spherical, FALSE otherwise. |

---

new\_wk\_wkt                      *S3 Details for wk\_wkt*

---

**Description**

S3 Details for wk\_wkt

**Usage**

```
new_wk_wkt(x = character(), crs = NULL, geodesic = NULL)
```

```
is_wk_wkt(x)
```

```
validate_wk_wkt(x)
```

**Arguments**

|          |  |
|----------|--|
| x        | A (possibly) <code>wkt()</code> vector   |
| crs      | A value to be propagated as the CRS for this vector.   |
| geodesic | TRUE if edges must be interpolated as geodesics when coordinates are spherical, FALSE otherwise. |

---

new\_wk\_xy                      *S3 details for xy objects*

---

**Description**

S3 details for xy objects

**Usage**

```
new_wk_xy(x = list(x = double(), y = double()), crs = NULL)
```

```
new_wk_xyz(x = list(x = double(), y = double(), z = double()), crs = NULL)
```

```
new_wk_xym(x = list(x = double(), y = double(), m = double()), crs = NULL)
```

```
new_wk_xyzm(
  x = list(x = double(), y = double(), z = double(), m = double()),
  crs = NULL
)
```

```
validate_wk_xy(x)
```

```
validate_wk_xyz(x)
```



```
validate_wk_xym(x)
```

```
validate_wk_xyzm(x)
```

### Arguments

`x` A `xy()` object.

`crs` A value to be propagated as the CRS for this vector.

---

|                |                          |
|----------------|--------------------------|
| plot.wk_grd_xy | <i>Plot grid objects</i> |
|----------------|--------------------------|

---

### Description

Plot grid objects

### Usage

```
## S3 method for class 'wk_grd_xy'
plot(x, ...)

## S3 method for class 'wk_grd_rct'
plot(
  x,
  ...,
  image = NULL,
  interpolate = FALSE,
  oversample = 4,
  border = NA,
  asp = 1,
  bbox = NULL,
  xlab = "",
  ylab = "",
  add = FALSE
)
```

### Arguments

`x` A `wkb()` or `wkt()`

`...` Passed to plotting functions for features: `graphics::points()` for point and multipoint geometries, `graphics::lines()` for linestring and multilinestring geometries, and `graphics::polypath()` for polygon and multipolygon geometries.

`image` A raster or `nativeRaster` to pass to `graphics::rasterImage()`. use `NULL` to do a quick-and-dirty rescale of the data such that the low value is black and the high value is white.

|                 |  |
|-----------------|--|
| interpolate     | Use TRUE to perform interpolation between color values.  |
| oversample      | A scale on the number of pixels on the device to use for sampling estimation of large raster values. Use Inf to disable. |
| border          | Color to use for polygon borders. Use NULL for the default and NA to skip plotting borders.                              |
| asp, xlab, ylab | Passed to <code>graphics::plot()</code>  |
| bbox            | The limits of the plot as a <code>rct()</code> or compatible object  |
| add             | Should a new plot be created, or should <code>handleable</code> be added to the existing plot?                           |

**Value**

x, invisibly.

**Examples**

```
plot(grd_rct(volcano))
plot(grd_xy(volcano))
```

---

rct

*2D rectangle vectors*


---

**Description**

2D rectangle vectors

**Usage**

```
rct(
  xmin = double(),
  ymin = double(),
  xmax = double(),
  ymax = double(),
  crs = wk_crs_auto()
)

as_rct(x, ...)

## S3 method for class 'wk_rct'
as_rct(x, ...)

## S3 method for class 'matrix'
as_rct(x, ..., crs = NULL)

## S3 method for class 'data.frame'
as_rct(x, ..., crs = NULL)
```

**Arguments**

|                        |  |
|------------------------|--|
| xmin, ymin, xmax, ymax | Rectangle bounds.                                    |
| crs                    | A value to be propagated as the CRS for this vector. |
| x                      | An object to be converted to a <code>rct()</code> .  |
| ...                    | Extra arguments passed to <code>as_rct()</code> .    |

**Value**

A vector along the recycled length of bounds.

**Examples**

```
rct(1, 2, 3, 4)
```

---

rct\_xmin                      *Rectangle accessors and operators*

---

**Description**

Rectangle accessors and operators

**Usage**

```
rct_xmin(x)
rct_ymin(x)
rct_xmax(x)
rct_ymax(x)
rct_width(x)
rct_height(x)
rct_intersects(x, y)
rct_contains(x, y)
rct_intersection(x, y)
```

**Arguments**

x, y                      `rct()` vectors

**Value**

- `rct_xmin()`, `rct_xmax()`, `rct_ymin()`, and `rct_ymax()` return the components of the `rct()`.

**Examples**

```
x <- rct(0, 0, 10, 10)
y <- rct(5, 5, 15, 15)

rct_xmin(x)
rct_ymin(x)
rct_xmax(x)
rct_ymax(x)
rct_height(x)
rct_width(x)
rct_intersects(x, y)
rct_intersection(x, y)
rct_contains(x, y)
rct_contains(x, rct(4, 4, 6, 6))
```

---

vctrs-methods

*Vctrs methods*

---

**Description**

Vctrs methods

**Usage**

```
vec_cast.wk_wkb(x, to, ...)
vec_ptype2.wk_wkb(x, y, ...)
vec_cast.wk_wkt(x, to, ...)
vec_ptype2.wk_wkt(x, y, ...)
vec_cast.wk_xy(x, to, ...)
vec_ptype2.wk_xy(x, y, ...)
vec_cast.wk_xyz(x, to, ...)
vec_ptype2.wk_xyz(x, y, ...)
vec_cast.wk_xym(x, to, ...)
vec_ptype2.wk_xym(x, y, ...)
```

```
vec_cast.wk_xyzm(x, to, ...)
```

```
vec_ptype2.wk_xyzm(x, y, ...)
```

```
vec_cast.wk_rct(x, to, ...)
```

```
vec_ptype2.wk_rct(x, y, ...)
```

```
vec_cast.wk_crc(x, to, ...)
```

```
vec_ptype2.wk_crc(x, y, ...)
```

### Arguments

`x, y, to, ...` See [vctrs::vec\\_cast\(\)](#) and [vctrs::vec\\_ptype2\(\)](#).

---

|     |   |
|-----|---|
| wkb | <i>Mark lists of raw vectors as well-known binary</i> |
|-----|---|

---

### Description

Mark lists of raw vectors as well-known binary

### Usage

```
wkb(x = list(), crs = wk_crs_auto(), geodesic = FALSE)
```

```
parse_wkb(x, crs = wk_crs_auto(), geodesic = FALSE)
```

```
wk_platform_endian()
```

```
as_wkb(x, ...)
```

```
## Default S3 method:
```

```
as_wkb(x, ...)
```

```
## S3 method for class 'character'
```

```
as_wkb(x, ..., crs = NULL, geodesic = FALSE)
```

```
## S3 method for class 'wk_wkb'
```

```
as_wkb(x, ...)
```

```
## S3 method for class 'blob'
```

```
as_wkb(x, ..., crs = NULL, geodesic = FALSE)
```

```
## S3 method for class 'WKB'
```

```
as_wkb(x, ..., crs = NULL, geodesic = FALSE)
```

**Arguments**

|          |  |
|----------|--|
| x        | A <code>list()</code> of <code>raw()</code> vectors or NULL.                                     |
| crs      | A value to be propagated as the CRS for this vector.   |
| geodesic | TRUE if edges must be interpolated as geodesics when coordinates are spherical, FALSE otherwise. |
| ...      | Unused   |

**Value**

A `new_wk_wkb()`

**Examples**

```
as_wkb("POINT (20 10)")
```

---

wkb\_to\_hex

*Convert well-known binary to hex*

---

**Description**

Convert well-known binary to hex

**Usage**

```
wkb_to_hex(x)
```

**Arguments**

|   |                             |
|---|-----------------------------|
| x | A <code>wkb()</code> vector |
|---|-----------------------------|

**Value**

A hex encoded `wkb()` vector

**Examples**

```
x <- as_wkb(xyz(1:5, 6:10, 11:15))
wkb_to_hex(x)
```

---

wkb\_translate\_wkt      *Deprecated functions*

---

### Description

These functions are deprecated and will be removed in a future version.

### Usage

```
wkb_translate_wkt(wkb, ..., precision = 16, trim = TRUE)
```

```
wkb_translate_wkb(wkb, ..., endian = NA_integer_)
```

```
wkt_translate_wkt(wkt, ..., precision = 16, trim = TRUE)
```

```
wkt_translate_wkb(wkt, ..., endian = NA_integer_)
```

### Arguments

|           |  |
|-----------|--|
| wkb       | A list() of raw() vectors, such as that returned by sf::st_as_binary().        |
| ...       | Used to keep backward compatibility with previous versions of these functions. |
| precision | The rounding precision to use when writing (number of decimal places).         |
| trim      | Trim unnecessary zeroes in the output?   |
| endian    | Force the endian of the resulting WKB.   |
| wkt       | A character vector containing well-known text.                                 |

---

wkt      *Mark character vectors as well-known text*

---

### Description

Mark character vectors as well-known text

### Usage

```
wkt(x = character(), crs = wk_crs_auto(), geodesic = FALSE)
```

```
parse_wkt(x, crs = wk_crs_auto(), geodesic = FALSE)
```

```
as_wkt(x, ...)
```

```
## Default S3 method:
```

```
as_wkt(x, ...)
```

```
## S3 method for class 'character'
as_wkt(x, ..., crs = NULL, geodesic = FALSE)

## S3 method for class 'wk_wkt'
as_wkt(x, ...)
```

### Arguments

|          |  |
|----------|--|
| x        | A <code>character()</code> vector containing well-known text.                                    |
| crs      | A value to be propagated as the CRS for this vector.   |
| geodesic | TRUE if edges must be interpolated as geodesics when coordinates are spherical, FALSE otherwise. |
| ...      | Unused   |

### Value

A `new_wk_wkt()`

### Examples

```
wkt("POINT (20 10)")
```

---

|         |                               |
|---------|-------------------------------|
| wk_bbox | <i>2D bounding rectangles</i> |
|---------|-------------------------------|

---

### Description

2D bounding rectangles

### Usage

```
wk_bbox(handleable, ...)

wk_envelope(handleable, ...)

## Default S3 method:
wk_bbox(handleable, ...)

## Default S3 method:
wk_envelope(handleable, ...)

## S3 method for class 'wk_rct'
wk_envelope(handleable, ...)

## S3 method for class 'wk_crc'
wk_envelope(handleable, ...)
```



```
## S3 method for class 'wk_xy'
wk_envelope(handleable, ...)

wk_bbox_handler()

wk_envelope_handler()
```

### Arguments

`handleable` A geometry vector (e.g., `wkb()`, `wkt()`, `xy()`, `rct()`, or `sf::st_sfc()`) for which `wk_handle()` is defined.

`...` Passed to the `wk_handle()` method.

### Value

A `rct()` of length 1.

### Examples

```
wk_bbox(wkt("LINESTRING (1 2, 3 5)"))
```

---

wk\_chunk\_strategy\_single  
*Chunking strategies*

---

### Description

It is often impractical, inefficient, or impossible to perform an operation on a vector of geometries with all the geometries loaded into memory at the same time. These functions help generalize the pattern of split-apply-combine to one or more handlers recycled along a common length. These functions are designed for developers rather than users and should be considered experimental.

### Usage

```
wk_chunk_strategy_single()

wk_chunk_strategy_feature(n_chunks = NULL, chunk_size = NULL)

wk_chunk_strategy_coordinates(n_chunks = NULL, chunk_size = NULL, reduce = "*")
```

**Arguments**

- `n_chunks`, `chunk_size` Exactly one of the number of chunks or the chunk size. For `wk_chunk_strategy_feature()` the chunk size refers to the number of features; for `wk_chunk_strategy_coordinates()` this refers to the number of coordinates as calculated from multiple handleables using `reduce`.
- `reduce` For `wk_chunk_strategy_coordinates()` this refers to the function used with `Reduce()` to combine coordinate counts from more than one handleable.

**Value**

A function that returns a data.frame with columns `from` and `to` when called with a handleable and the feature count.

**Examples**

```
feat <- c(as_wkt(xy(1:4, 1:4)), wkt("LINESTRING (1 1, 2 2)"))
wk_chunk_strategy_single()(list(feat), 5)
wk_chunk_strategy_feature(chunk_size = 2)(list(feat), 5)
wk_chunk_strategy_coordinates(chunk_size = 2)(list(feat), 5)
```

---

|          |                                  |
|----------|----------------------------------|
| wk_count | <i>Count geometry components</i> |
|----------|----------------------------------|

---

**Description**

Counts the number of geometries, rings, and coordinates found within each feature. As opposed to `wk_meta()`, this handler will iterate over the entire geometry.

**Usage**

```
wk_count(handleable, ...)
```

```
## Default S3 method:
wk_count(handleable, ...)
```

```
wk_count_handler()
```

**Arguments**

- `handleable` A geometry vector (e.g., `wkb()`, `wkt()`, `xy()`, `rct()`, or `sf::st_sfc()`) for which `wk_handle()` is defined.
- `...` Passed to the `wk_handle()` method.

**Value**

A data.frame with one row for every feature encountered and columns:

- `n_geom`: The number of geometries encountered, including the root geometry. Will be zero for a null feature.
- `n_ring`: The number of rings encountered. Will be zero for a null feature.
- `n_coord`: The number of coordinates encountered. Will be zero for a null feature.

**Examples**

```
wk_count(as_wkt("LINESTRING (0 0, 1 1)"))
wk_count(as_wkb("LINESTRING (0 0, 1 1)"))
```

---

 wk\_crs

*Set and get vector CRS*


---

**Description**

The wk package doesn't operate on CRS objects, but does propagate them through subsetting and concatenation. A CRS object can be any R object, and x can be any object whose 'crs' attribute carries a CRS. These functions are S3 generics to keep them from being used on objects that do not use this system of CRS propagation.

**Usage**

```
wk_crs(x)

## S3 method for class 'wk_vctr'
wk_crs(x)

## S3 method for class 'wk_rcrd'
wk_crs(x)

wk_crs(x) <- value

wk_set_crs(x, crs)

wk_crs_output(...)

wk_is_geodesic_output(...)
```

**Arguments**

|                     |   |
|---------------------|---|
| <code>x, ...</code> | Objects whose "crs" attribute is used to carry a CRS. |
| <code>value</code>  | See <code>crs</code> .                                |
| <code>crs</code>    | An object that can be interpreted as a CRS            |

---

|              |                            |
|--------------|----------------------------|
| wk_crs_equal | <i>Compare CRS objects</i> |
|--------------|----------------------------|

---

### Description

The `wk_crs_equal()` function uses special S3 dispatch on `wk_crs_equal_generic()` to evaluate whether or not two CRS values can be considered equal. When implementing `wk_crs_equal_generic()`, every attempt should be made to make `wk_crs_equal(x, y)` and `wk_crs_equal(y, x)` return identically.

### Usage

```
wk_crs_equal(x, y)

wk_crs_equal_generic(x, y, ...)
```

### Arguments

|                   |   |
|-------------------|---|
| <code>x, y</code> | Objects stored in the <code>crs</code> attribute of a vector. |
| <code>...</code>  | Unused  |

### Value

TRUE if `x` and `y` can be considered equal, FALSE otherwise.

---

|                |                           |
|----------------|---------------------------|
| wk_crs_inherit | <i>Special CRS values</i> |
|----------------|---------------------------|

---

### Description

The CRS handling in the `wk` package requires two sentinel CRS values. The first, `wk_crs_inherit()`, signals that the vector should inherit a CRS of another vector if combined. This is useful for empty, NULL, and/or zero-length geometries. The second, `wk_crs_auto()`, is used as the default argument of `crs` for constructors so that zero-length geometries are assigned a CRS of `wk_crs_inherit()` by default.

### Usage

```
wk_crs_inherit()

wk_crs_longlat(crs = NULL)

wk_crs_auto()

wk_crs_auto_value(x, crs)
```

**Arguments**

|     |   |
|-----|---|
| crs | A value for the coordinate reference system supplied by the user.   |
| x   | A raw input to a constructor whose length and crs attribute is used to determine the default CRS returned by <code>wk_crs_auto()</code> . |

**Examples**

```
wk_crs_auto_value(list(), wk_crs_auto())
wk_crs_auto_value(list(), 1234)
wk_crs_auto_value(list(NULL), wk_crs_auto())
```

---

```
wk_crs_proj_definition
```

*CRS object generic methods*

---

**Description**

CRS object generic methods

**Usage**

```
wk_crs_proj_definition(crs, proj_version = NULL, verbose = FALSE)

wk_crs_projjson(crs)

## S3 method for class '`NULL`'
wk_crs_proj_definition(crs, proj_version = NULL, verbose = FALSE)

## S3 method for class 'wk_crs_inherit'
wk_crs_proj_definition(crs, proj_version = NULL, verbose = FALSE)

## S3 method for class 'character'
wk_crs_proj_definition(crs, proj_version = NULL, verbose = FALSE)

## S3 method for class 'double'
wk_crs_proj_definition(crs, proj_version = NULL, verbose = FALSE)

## S3 method for class 'integer'
wk_crs_proj_definition(crs, proj_version = NULL, verbose = FALSE)
```

**Arguments**

|              |   |
|--------------|---|
| crs          | An arbitrary R object   |
| proj_version | A <code>package_version()</code> of the PROJ version, or NULL if the PROJ version is unknown. |

`verbose` Use TRUE to request a more verbose version of the PROJ definition (e.g., PROJJSON). The default of FALSE should return the most compact version that completely describes the CRS. An authority:code string (e.g., "OGC:CRS84") is the recommended way to represent a CRS when verbose is FALSE, if possible, falling back to the most recent version of WKT2 or PROJJSON.

### Value

- `wk_crs_proj_definition()` Returns a string used to represent the CRS in PROJ. For recent PROJ version you'll want to return PROJJSON; however you should check `proj_version` if you want this to work with older versions of PROJ.
- `wk_crs_projjson()` Returns a PROJJSON string or `NA_character_` if this representation is unknown or can't be calculated.

### Examples

```
wk_crs_proj_definition("EPSG:4326")
```

---

wk\_debug

*Debug filters and handlers*

---

### Description

Debug filters and handlers

### Usage

```
wk_debug(handleable, handler = wk_void_handler(), ...)
```

```
wk_debug_filter(handler = wk_void_handler())
```

### Arguments

`handleable` A geometry vector (e.g., `wkb()`, `wkt()`, `xy()`, `rct()`, or `sf::st_sfc()`) for which `wk_handle()` is defined.

`handler` A `wk_handler` object.

`...` Passed to the `wk_handle()` method.

### Value

The result of the handler.

### Examples

```
wk_debug(wkt("POINT (1 1)"))
wk_handle(wkt("POINT (1 1)"), wk_debug_filter())
```

---

|            |  |
|------------|--|
| wk_example | <i>Create example geometry objects</i> |
|------------|--|

---

**Description**

Create example geometry objects

**Usage**

```
wk_example(which = "nc", crs = NA, geodesic = FALSE)
```

```
wk_example_wkt
```

**Arguments**

|          |   |
|----------|---|
| which    | An example name. Valid example names are <ul style="list-style-type: none"><li>• "nc" (data derived from the sf package)</li><li>• "point", "linestring", "polygon", "multipoint", "multilinestring", "multipolygon", "geometrycollection"</li><li>• One of the above with the "_z", "_m", or "_zm" suffix.</li></ul> |
| crs      | An object that can be interpreted as a CRS  |
| geodesic | TRUE if edges must be interpolated as geodesics when coordinates are spherical, FALSE otherwise.  |

**Format**

An object of class `list` of length 29.

**Value**

A `wkt()` with the specified example.

**Examples**

```
wk_example("polygon")
```

---

|            |                                  |
|------------|----------------------------------|
| wk_flatten | <i>Extract simple geometries</i> |
|------------|----------------------------------|

---

## Description

Extract simple geometries

## Usage

```
wk_flatten(handleable, ..., max_depth = 1)
```

```
wk_flatten_filter(handler, max_depth = 1L, add_details = FALSE)
```

## Arguments

|             |   |
|-------------|---|
| handleable  | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| ...         | Passed to the <code>wk_handle()</code> method.  |
| max_depth   | The maximum (outer) depth to remove.  |
| handler     | A <code>wk_handler</code> object.   |
| add_details | Use TRUE to add a "wk_details" attribute, which contains columns <code>feature_id</code> , <code>part_id</code> , and <code>ring_id</code> .  |

## Value

handleable transformed such that collections have been expanded and only simple geometries (point, linestring, polygon) remain.

## Examples

```
wk_flatten(wkt("MULTIPOINT (1 1, 2 2, 3 3)"))
wk_flatten(
  wkt("GEOMETRYCOLLECTION (GEOMETRYCOLLECTION (GEOMETRYCOLLECTION (POINT (0 1))))"),
  max_depth = 2
)
```



---

|           |  |
|-----------|--|
| wk_format | <i>Format well-known geometry for printing</i> |
|-----------|--|

---

### Description

Provides an abbreviated version of the well-known text representation of a geometry. This returns a constant number of coordinates for each geometry, so is safe to use for geometry vectors with many (potentially large) features. Parse errors are passed on to the format string and do not cause this handler to error.

### Usage

```
wk_format(handleable, precision = 7, trim = TRUE, max_coords = 6, ...)
```

```
wkt_format_handler(precision = 7, trim = TRUE, max_coords = 6)
```

### Arguments

|            |   |
|------------|---|
| handleable | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| precision  | If <code>trim</code> is <code>TRUE</code> , the total number of significant digits to keep for each result or the number of digits after the decimal place otherwise.                     |
| trim       | Use <code>FALSE</code> to keep trailing zeroes after the decimal place.   |
| max_coords | The maximum number of coordinates to include in the output.   |
| ...        | Passed to the <code>wk_handle()</code> method.  |

### Value

A character vector of abbreviated well-known text.

### Examples

```
wk_format(wkt("MULTIPOLYGON (((0 0, 10 0, 0 10, 0 0)))"))
wk_format(new_wk_wkt("POINT ENTPTY"))
wk_handle(
  wkt("MULTIPOLYGON (((0 0, 10 0, 0 10, 0 0)))"),
  wkt_format_handler()
)
```

---

wk\_handle.data.frame *Use data.frame with wk*

---

## Description

Use data.frame with wk

## Usage

```
## S3 method for class 'data.frame'
wk_handle(handleable, handler, ...)

## S3 method for class 'data.frame'
wk_restore(handleable, result, ...)

## S3 method for class 'tbl_df'
wk_restore(handleable, result, ...)

## S3 method for class 'data.frame'
wk_translate(handleable, to, ...)

## S3 method for class 'tbl_df'
wk_translate(handleable, to, ...)

## S3 method for class 'sf'
wk_translate(handleable, to, ...)

## S3 method for class 'sf'
wk_restore(handleable, result, ...)
```

## Arguments

|            |   |
|------------|---|
| handleable | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| handler    | A <code>wk_handler</code> object.   |
| ...        | Passed to the <code>wk_handle()</code> method.  |
| result     | The result of a filter operation intended to be a transformation.   |
| to         | A prototype object.   |

## Examples

```
wk_handle(data.frame(a = wkt("POINT (0 1)")), wkb_writer())
wk_translate(wkt("POINT (0 1)"), data.frame(col_name = wkb()))
wk_translate(data.frame(a = wkt("POINT (0 1)")), data.frame(wkb()))
```

---

 wk\_handle.wk\_crc      *Read geometry vectors*


---

## Description

The handler is the basic building block of the wk package. In particular, the `wk_handle()` generic allows operations written as handlers to "just work" with many different input types. The wk package provides the `wk_void()` handler, the `wk_format()` handler, the `wk_debug()` handler, the `wk_problems()` handler, and `wk_writer()`s for `wkb()`, `wkt()`, `xy()`, and `sf::st_sfc()` vectors.

## Usage

```
## S3 method for class 'wk_crc'
wk_handle(
  handleable,
  handler,
  ...,
  n_segments = getOption("wk_crc_n_segments", NULL),
  resolution = getOption("wk_crc_resolution", NULL)
)

## S3 method for class 'wk_rct'
wk_handle(handleable, handler, ...)

## S3 method for class 'sfc'
wk_handle(handleable, handler, ...)

## S3 method for class 'wk_wkb'
wk_handle(handleable, handler, ...)

## S3 method for class 'wk_wkt'
wk_handle(handleable, handler, ...)

## S3 method for class 'wk_xy'
wk_handle(handleable, handler, ...)

wk_handle(handleable, handler, ...)

is_handleable(handleable)

new_wk_handler(handler_ptr, subclass = character())

is_wk_handler(handler)

as_wk_handler(handler, ...)

## S3 method for class 'sfg'
```

```
wk_handle(handleable, handler, ...)
```

```
## S3 method for class 'sf'
wk_handle(handleable, handler, ...)
```

```
## S3 method for class 'bbox'
wk_handle(handleable, handler, ...)
```

### Arguments

|                        |  |
|------------------------|--|
| handleable             | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined.  |
| handler                | A <code>wk_handler</code> object.  |
| ...                    | Passed to the <code>wk_handle()</code> method.   |
| n_segments, resolution | The number of segments to use when approximating a circle. The default uses <code>getOption("wk.crc_n_segments")</code> so that this value can be set for implicit conversions (e.g., <code>as_wkb()</code> ). Alternatively, set the minimum distance between points on the circle (used to estimate <code>n_segments</code> ). The default is obtained using <code>getOption("wk.crc_resolution")</code> . |
| handler_ptr            | An external pointer to a newly created WK handler  |
| subclass               | The handler subclass   |

### Value

A WK handler.

---

`wk_handle.wk_grd_xy`    *Handler interface for grid objects*

---

### Description

Handler interface for grid objects

### Usage

```
## S3 method for class 'wk_grd_xy'
wk_handle(handleable, handler, ..., data_order = c("y", "x"))
```

```
## S3 method for class 'wk_grd_rct'
wk_handle(handleable, handler, ..., data_order = c("y", "x"))
```

**Arguments**

|            |   |
|------------|---|
| handleable | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined.   |
| handler    | A <code>wk_handler</code> object.   |
| ...        | Passed to the <code>wk_handle()</code> method.  |
| data_order | A vector of length 2 describing the order in which values should appear. The default, <code>c("y", "x")</code> , will output values in the same order as the default matrix storage in R (column-major). You can prefix a dimension with <code>-</code> to reverse the order of a dimension (e.g., <code>c("-y", "x")</code> ). |

**Value**

The result of the handler.

**Examples**

```
wk_handle(grd(nx = 3, ny = 3), wkt_writer())
wk_handle(grd(nx = 3, ny = 3, type = "centers"), wkt_writer())
```

---

wk\_handle\_slice.data.frame

*Handle specific regions of objects*

---

**Description**

Handle specific regions of objects

**Usage**

```
## S3 method for class 'data.frame'
wk_handle_slice(handleable, handler, from = NULL, to = NULL, ...)

wk_handle_slice(
  handleable,
  handler = wk_writer(handleable),
  from = NULL,
  to = NULL,
  ...
)

## Default S3 method:
wk_handle_slice(
  handleable,
  handler = wk_writer(handleable),
  from = NULL,
```

```

    to = NULL,
    ...
)

```

### Arguments

|            |   |
|------------|---|
| handleable | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| handler    | A <code>wk_handler</code> object.   |
| from       | 1-based index of the feature to start from  |
| to         | 1-based index of the feature to end at  |
| ...        | Passed to the <code>wk_handle()</code> method.  |

### Value

A subset of handleable

### Examples

```

wk_handle_slice(xy(1:5, 1:5), wkt_writer(), from = 3, to = 5)
wk_handle_slice(
  data.frame(let = letters[1:5], geom = xy(1:5, 1:5)),
  wkt_writer(),
  from = 3, to = 5
)

```

---

wk\_identity

*Copy a geometry vector*

---

### Description

Copy a geometry vector

### Usage

```
wk_identity(handleable, ...)
```

```
wk_identity_filter(handler)
```

```
wk_restore(handleable, result, ...)
```

```
## Default S3 method:
```

```
wk_restore(handleable, result, ...)
```

**Arguments**

|            |   |
|------------|---|
| handleable | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| ...        | Passed to the <code>wk_handle()</code> method.  |
| handler    | A <code>wk_handler</code> object.   |
| result     | The result of a filter operation intended to be a transformation.   |

**Value**

A copy of handleable.

**Examples**

```
wk_identity(wkt("POINT (1 2)"))
```

---

|                |   |
|----------------|---|
| wk_is_geodesic | <i>Set and get vector geodesic edge interpolation</i> |
|----------------|---|

---

**Description**

Set and get vector geodesic edge interpolation

**Usage**

```
wk_is_geodesic(x)

wk_set_geodesic(x, geodesic)

wk_is_geodesic(x) <- value

wk_geodesic_inherit()
```

**Arguments**

|          |  |
|----------|--|
| x        | An R object that contains edges  |
| geodesic | TRUE if edges must be interpolated as geodesics when coordinates are spherical, FALSE otherwise. |
| value    | See geodesic.  |

**Value**

TRUE if edges must be interpolated as geodesics when coordinates are spherical, FALSE otherwise.

---

|               |  |
|---------------|--|
| wk_linestring | <i>Create lines, polygons, and collections</i> |
|---------------|--|

---

**Description**

Create lines, polygons, and collections

**Usage**

```
wk_linestring(handleable, feature_id = 1L, ..., geodesic = NULL)
```

```
wk_polygon(handleable, feature_id = 1L, ring_id = 1L, ..., geodesic = NULL)
```

```
wk_collection(
  handleable,
  geometry_type = wk_geometry_type("geometrycollection"),
  feature_id = 1L,
  ...
)
```

```
wk_linestring_filter(handler, feature_id = 1L)
```

```
wk_polygon_filter(handler, feature_id = 1L, ring_id = 1L)
```

```
wk_collection_filter(
  handler,
  geometry_type = wk_geometry_type("geometrycollection"),
  feature_id = 1L
)
```

**Arguments**

|               |   |
|---------------|---|
| handleable    | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| feature_id    | An identifier where changes in sequential values indicate a new feature. This is recycled silently as needed.   |
| ...           | Passed to the <code>wk_handle()</code> method.  |
| geodesic      | Use TRUE or FALSE to explicitly force the geodesic-ness of the output.  |
| ring_id       | An identifier where changes in sequential values indicate a new ring. Rings are automatically closed. This is recycled silently as needed.  |
| geometry_type | The collection type to create.  |
| handler       | A <code>wk_handler</code> object.   |

**Value**

An object of the same class as `handleable` with whose coordinates have been assembled into the given type.



**Examples**

```
wk_linestring(xy(c(1, 1), c(2, 3)))
wk_polygon(xy(c(0, 1, 0), c(0, 0, 1)))
wk_collection(xy(c(1, 1), c(2, 3)))
```

---

 wk\_meta

*Extract feature-level meta*


---

**Description**

These functions return the non-coordinate information of a geometry and/or vector. They do not parse an entire geometry/vector and are intended to be very fast even for large vectors.

**Usage**

```
wk_meta(handleable, ...)

## Default S3 method:
wk_meta(handleable, ...)

wk_vector_meta(handleable, ...)

## Default S3 method:
wk_vector_meta(handleable, ...)

wk_meta_handler()

wk_vector_meta_handler()

wk_geometry_type_label(geometry_type)

wk_geometry_type(geometry_type_label)
```

**Arguments**

|                     |   |
|---------------------|---|
| handleable          | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| ...                 | Passed to the <code>wk_handle()</code> method.  |
| geometry_type       | An integer code for the geometry type. These integers follow the WKB specification (e.g., 1 for point, 7 for geometrycollection).   |
| geometry_type_label | A character vector of (lowercase) geometry type labels as would be found in WKT (e.g., point, geometrycollection).  |

**Value**

A data.frame with columns:

- `geometry_type`: An integer identifying the geometry type. A value of 0 indicates that the types of geometry in the vector are not known without parsing the entire vector.
- `size`: For points and linestrings, the number of coordinates; for polygons, the number of rings; for collections, the number of child geometries. A value of zero indicates an EMPTY geometry. A value of NA means this value is unknown without parsing the entire geometry.
- `has_z`: TRUE if coordinates contain a Z value. A value of NA means this value is unknown without parsing the entire vector.
- `has_m`: TRUE if coordinates contain an M value. A value of NA means this value is unknown without parsing the entire vector.
- `srid`: An integer identifying a CRS or NA if this value was not provided.
- `precision`: A grid size or 0.0 if a grid size was not provided. Note that coordinate values may not have been rounded; the grid size only refers to the level of detail with which they should be interpreted.
- `is_empty`: TRUE if there is at least one non-empty coordinate. For the purposes of this value, a non-empty coordinate is one that contains at least one value that is not NA or NaN.

**Examples**

```
wk_vector_meta(as_wkt("LINESTRING (0 0, 1 1)"))
wk_meta(as_wkt("LINESTRING (0 0, 1 1)"))
wk_meta(as_wkb("LINESTRING (0 0, 1 1)"))

wk_geometry_type_label(1:7)
wk_geometry_type(c("point", "geometrycollection"))
```

---

wk\_orient

*Orient polygon coordinates*

---

**Description**

Orient polygon coordinates

**Usage**

```
wk_orient(handleable, ..., direction = wk_counterclockwise())

wk_orient_filter(handler, direction = wk_counterclockwise())

wk_clockwise()

wk_counterclockwise()
```

**Arguments**

|            |   |
|------------|---|
| handleable | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| ...        | Passed to the <code>wk_handle()</code> method.  |
| direction  | The winding polygon winding direction   |
| handler    | A <code>wk_handler</code> object.   |

**Value**

handleable with consistently oriented polygons, in direction winding order.

**Examples**

```
wk_orient(wkt("POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))"))
wk_orient(
  wkt("POLYGON ((0 0, 0 1, 1 1, 1 0, 0 0))"),
  direction = wk_clockwise()
)
```

---

 wk\_plot

*Plot well-known geometry vectors*


---

**Description**

Plot well-known geometry vectors

**Usage**

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

## Default S3 method:
wk_plot(
  handleable,
  ...,
  asp = 1,
  bbox = NULL,
```

```

    xlab = "",
    ylab = "",
    rule = "evenodd",
    add = FALSE
)

## S3 method for class 'wk_wkt'
plot(
  x,
  ...,
  asp = 1,
  bbox = NULL,
  xlab = "",
  ylab = "",
  rule = "evenodd",
  add = FALSE
)

## S3 method for class 'wk_wkb'
plot(
  x,
  ...,
  asp = 1,
  bbox = NULL,
  xlab = "",
  ylab = "",
  rule = "evenodd",
  add = FALSE
)

## S3 method for class 'wk_xy'
plot(x, ..., asp = 1, bbox = NULL, xlab = "", ylab = "", add = FALSE)

## S3 method for class 'wk_rct'
plot(x, ..., asp = 1, bbox = NULL, xlab = "", ylab = "", add = FALSE)

## S3 method for class 'wk_crc'
plot(x, ..., asp = 1, bbox = NULL, xlab = "", ylab = "", add = FALSE)

```

## Arguments

|                 |  |
|-----------------|--|
| handleable      | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined.  |
| ...             | Passed to plotting functions for features: <code>graphics::points()</code> for point and multipoint geometries, <code>graphics::lines()</code> for linestring and multilinestring geometries, and <code>graphics::polypath()</code> for polygon and multipolygon geometries. |
| asp, xlab, ylab | Passed to <code>graphics::plot()</code>  |

|      |  |
|------|--|
| bbox | The limits of the plot as a <code>rct()</code> or compatible object                            |
| rule | The rule to use for filling polygons (see <code>graphics::polypath()</code> )                  |
| add  | Should a new plot be created, or should <code>handleable</code> be added to the existing plot? |
| x    | A <code>wkb()</code> or <code>wkt()</code>   |

**Value**

The input, invisibly.

**Examples**

```
plot(as_wkt("LINESTRING (0 0, 1 1)"))
plot(as_wkb("LINESTRING (0 0, 1 1)"))
```

---

|             |   |
|-------------|---|
| wk_problems | <i>Validate well-known binary and well-known text</i> |
|-------------|---|

---

**Description**

The problems handler returns a character vector of parse errors and can be used to validate input of any type for which `wk_handle()` is defined.

**Usage**

```
wk_problems(handleable, ...)

wk_problems_handler()
```

**Arguments**

|                         |   |
|-------------------------|---|
| <code>handleable</code> | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| <code>...</code>        | Passed to the <code>wk_handle()</code> method.  |

**Value**

A character vector of parsing errors. NA signifies that there was no parsing error.

**Examples**

```
wk_problems(new_wk_wkt(c("POINT EMPTY", "POINT (20 30)")))
wk_handle(
  new_wk_wkt(c("POINT EMPTY", "POINT (20 30)")),
  wk_problems_handler()
)
```

---

|                  |                                   |
|------------------|-----------------------------------|
| wk_proj_crs_view | <i>Common CRS Representations</i> |
|------------------|-----------------------------------|

---

**Description**

These fixtures are calculated from PROJ version 9.1.0 and the database built from its source. They are used internally to transform and inspect coordinate reference systems.

**Usage**

```
wk_proj_crs_view
```

```
wk_proj_crs_json
```

**Format**

An object of class `data.frame` with 13387 rows and 7 columns.

An object of class `data.frame` with 13387 rows and 3 columns.

**Examples**

```
head(wk_proj_crs_view)
colnames(wk_proj_crs_json)
```

---

|          |                              |
|----------|------------------------------|
| wk_set_z | <i>Set coordinate values</i> |
|----------|------------------------------|

---

**Description**

Set coordinate values

**Usage**

```
wk_set_z(handleable, z, ...)
```

```
wk_set_m(handleable, m, ...)
```

```
wk_drop_z(handleable, ...)
```

```
wk_drop_m(handleable, ...)
```

```
wk_trans_set(value, use_z = NA, use_m = NA)
```

**Arguments**

|              |   |
|--------------|---|
| handleable   | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| z, m         | A vector of Z or M values applied feature-wise and recycled along handleable. Use NA to keep the existing value of a given feature.   |
| ...          | Passed to the <code>wk_handle()</code> method.  |
| value        | An <code>xy()</code> , <code>xyz()</code> , <code>xym()</code> , or <code>xyzm()</code> of coordinates used to replace values in the input. Use NA to keep the existing value.            |
| use_z, use_m | Used to declare the output type. Use TRUE to ensure the output has that dimension, FALSE to ensure it does not, and NA to leave the dimension unchanged.                                  |

**Examples**

```
wk_set_z(wkt("POINT (0 1)"), 2)
wk_set_m(wkt("POINT (0 1)"), 2)
wk_drop_z(wkt("POINT ZM (0 1 2 3)"))
wk_drop_m(wkt("POINT ZM (0 1 2 3)"))
```

---

|              |   |
|--------------|---|
| wk_transform | <i>Apply coordinate transformations</i> |
|--------------|---|

---

**Description**

Apply coordinate transformations

**Usage**

```
wk_transform(handleable, trans, ...)
```

```
wk_transform_filter(handler, trans)
```

**Arguments**

|            |   |
|------------|---|
| handleable | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| trans      | An external pointer to a wk_trans object  |
| ...        | Passed to the <code>wk_handle()</code> method.  |
| handler    | A <code>wk_handler</code> object.   |

**Examples**

```
wk_transform(xy(0, 0), wk_affine_translate(2, 3))
```

---

wk\_translate.sfc      *Translate geometry vectors*

---

### Description

Translate geometry vectors

### Usage

```
## S3 method for class 'sfc'
wk_translate(handleable, to, ...)

wk_translate(handleable, to, ...)

## Default S3 method:
wk_translate(handleable, to, ...)
```

### Arguments

|            |   |
|------------|---|
| handleable | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| to         | A prototype object.   |
| ...        | Passed to the <code>wk_handle()</code> method.  |

---

wk\_trans\_affine      *Affine transformer*

---

### Description

Affine transformer

### Usage

```
wk_trans_affine(trans_matrix)

wk_affine_identity()

wk_affine_rotate(rotation_deg)

wk_affine_scale(scale_x = 1, scale_y = 1)

wk_affine_translate(dx = 0, dy = 0)

wk_affine_fit(src, dst)
```



```
wk_affine_rescale(rct_in, rct_out)
```

```
wk_affine_compose(...)
```

```
wk_affine_invert(x)
```

### Arguments

|                  |   |
|------------------|---|
| trans_matrix     | A 3x3 transformation matrix   |
| rotation_deg     | A rotation to apply in degrees counterclockwise.  |
| scale_x, scale_y | Scale factor to apply in the x and y directions, respectively   |
| dx, dy           | Coordinate offsets in the x and y direction   |
| src, dst         | Point vectors of control points used to estimate the affine mapping (using <code>base::qr.solve()</code> ). |
| rct_in, rct_out  | The input and output bounds   |
| ...              | Zero or more transforms in the order they should be applied.  |
| x                | A <code>wk_trans_affine()</code>  |

---

|                   |   |
|-------------------|---|
| wk_trans_explicit | <i>Transform using explicit coordinate values</i> |
|-------------------|---|

---

### Description

A `wk_trans` implementation that replaces coordinate values using a vector of pre-calculated coordinates. This is used to perform generic transforms using R functions and system calls that are impossible or impractical to implement at the C level.

### Usage

```
wk_trans_explicit(value, use_z = NA, use_m = NA)
```

### Arguments

|              |  |
|--------------|--|
| value        | An <code>xy()</code> , <code>xyz()</code> , <code>xym()</code> , or <code>xyzm()</code> of coordinates used to replace values in the input. Use NA to keep the existing value. |
| use_z, use_m | Used to declare the output type. Use TRUE to ensure the output has that dimension, FALSE to ensure it does not, and NA to leave the dimension unchanged.                       |

### See Also

`wk_coords()` which has a replacement version "wk\_coords<-"

### Examples

```
trans <- wk_trans_explicit(xy(1:5, 1:5))
wk_transform(rep(xy(0, 0), 5), trans)
```

---

wk\_trans\_inverse      *Generic transform class*

---

### Description

Generic transform class

### Usage

```
wk_trans_inverse(trans, ...)

as_wk_trans(x, ...)

## S3 method for class 'wk_trans'
as_wk_trans(x, ...)

new_wk_trans(trans_ptr, subclass = character())
```

### Arguments

|           |   |
|-----------|---|
| trans     | An external pointer to a wk_trans object              |
| ...       | Passed to S3 methods                                  |
| x         | An object to be converted to a transform.             |
| trans_ptr | An external pointer to a wk_trans_t transform struct. |
| subclass  | An optional subclass to apply to the pointer          |

---

wk\_vertices      *Extract vertices*

---

### Description

These functions provide ways to extract individual coordinate values. Whereas wk\_vertices() returns a vector of coordinates as in the same format as the input, wk\_coords() returns a data frame with coordinates as columns.

### Usage

```
wk_vertices(handleable, ...)

wk_coords(handleable, ...)

wk_coords(handleable, use_z = NA, use_m = NA) <- value

wk_vertex_filter(handler, add_details = FALSE)
```

**Arguments**

|              |   |
|--------------|---|
| handleable   | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| ...          | Passed to the <code>wk_handle()</code> method.  |
| use_z, use_m | Used to declare the output type. Use TRUE to ensure the output has that dimension, FALSE to ensure it does not, and NA to leave the dimension unchanged.                                  |
| value        | An <code>xy()</code> , <code>xyz()</code> , <code>xym()</code> , or <code>xyzm()</code> of coordinates used to replace values in the input. Use NA to keep the existing value.            |
| handler      | A <code>wk_handler</code> object.   |
| add_details  | Use TRUE to add a "wk_details" attribute, which contains columns <code>feature_id</code> , <code>part_id</code> , and <code>ring_id</code> .  |

**Details**

`wk_coords<-` is the replacement-function version of 'wk\_coords'. Using the engine of `wk_trans_explicit()` the coordinates of an object can be transformed in a generic way using R functions as needed.

**Value**

- `wk_vertices()` extracts vertices and returns the in the same format as the handler
- `wk_coords()` returns a data frame with columns `feature_id` (the index of the feature from whence it came), `part_id` (an arbitrary integer identifying the point, line, or polygon from whence it came), `ring_id` (an arbitrary integer identifying individual rings within polygons), and one column per coordinate (x, y, and/or z and/or m).

**Examples**

```
wk_vertices(wkt("LINESTRING (0 0, 1 1)"))
wk_coords(wkt("LINESTRING (0 0, 1 1)"))

# wk_coords() replacement function
x <- xy(1:5, 1:5)
y <- as_wkt(x)
wk_coords(y) <- cbind(5:1, 0:4)
wk_coords(x) <- y[5:1]
y
x
```

---

 wk\_void

*Do nothing*


---

**Description**

This handler does nothing and returns NULL. It is useful for benchmarking readers and handlers and when using filters that have side-effects (e.g., `wk_debug()`). Note that this handler stops on the first parse error; to see a list of parse errors see the `wk_problems()` handler.

**Usage**

```
wk_void(handleable, ...)

wk_void_handler()
```

**Arguments**

|            |   |
|------------|---|
| handleable | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| ...        | Passed to the <code>wk_handle()</code> method.  |

**Value**

```
NULL
```

**Examples**

```
wk_void(wkt("POINT (1 4)"))
wk_handle(wkt("POINT (1 4)"), wk_void_handler())
```

---

```
wk_writer.sfc
```

```
Write geometry vectors
```

---

**Description**

When writing transformation functions, it is often useful to know which handler should be used to create a (potentially modified) version of an object. Some transformers (e.g., `wk_vertices()`) modify the geometry type of an object, in which case a generic writer is needed. This defaults to `wkb_writer()` because it is fast and can handle all geometry types.

**Usage**

```
## S3 method for class 'sfc'
wk_writer(handleable, ...)

## S3 method for class 'sf'
wk_writer(handleable, ...)

sfc_writer(promote_multi = FALSE)

wkb_writer(buffer_size = 2048L, endian = NA_integer_)

wkt_writer(precision = 16L, trim = TRUE)

wk_writer(handleable, ..., generic = FALSE)
```

```

## Default S3 method:
wk_writer(handleable, ...)

## S3 method for class 'wk_wkt'
wk_writer(handleable, ..., precision = 16, trim = TRUE)

## S3 method for class 'wk_wkb'
wk_writer(handleable, ...)

## S3 method for class 'wk_xy'
wk_writer(handleable, ..., generic = FALSE)

xy_writer()

```

### Arguments

|               |   |
|---------------|---|
| handleable    | A geometry vector (e.g., <code>wkb()</code> , <code>wkt()</code> , <code>xy()</code> , <code>rct()</code> , or <code>sf::st_sfc()</code> ) for which <code>wk_handle()</code> is defined. |
| ...           | Passed to the writer constructor.   |
| promote_multi | Use TRUE to promote all simple geometries to a multi type when reading to sfc. This is useful to increase the likelihood that the sfc will contain a single geometry type.                |
| buffer_size   | Control the initial buffer size used when writing WKB.  |
| endian        | Use 1 for little endian, 0 for big endian, or NA for system endian.   |
| precision     | If trim is TRUE, the total number of significant digits to keep for each result or the number of digits after the decimal place otherwise.  |
| trim          | Use FALSE to keep trailing zeroes after the decimal place.  |
| generic       | Use TRUE to obtain a writer that can write all geometry types.  |

### Value

A `wk_handler`.

---

xy

*Efficient point vectors*

---

### Description

Efficient point vectors

**Usage**

```

xy(x = double(), y = double(), crs = wk_crs_auto())

xyz(x = double(), y = double(), z = double(), crs = wk_crs_auto())

xym(x = double(), y = double(), m = double(), crs = wk_crs_auto())

xyzm(
  x = double(),
  y = double(),
  z = double(),
  m = double(),
  crs = wk_crs_auto()
)

xy_dims(x)

as_xy(x, ...)

## Default S3 method:
as_xy(x, ..., dims = NULL)

## S3 method for class 'wk_xy'
as_xy(x, ..., dims = NULL)

## S3 method for class 'matrix'
as_xy(x, ..., crs = NULL)

## S3 method for class 'data.frame'
as_xy(x, ..., dims = NULL, crs = NULL)

```

**Arguments**

|            |  |
|------------|--|
| x, y, z, m | Coordinate values.                                     |
| crs        | A value to be propagated as the CRS for this vector.   |
| ...        | Passed to methods.                                     |
| dims       | A set containing one or more of c("x", "y", "z", "m"). |

**Value**

A vector of coordinate values.

**Examples**

```

xy(1:5, 1:5)
xyz(1:5, 1:5, 10)
xym(1:5, 1:5, 10)
xyzm(1:5, 1:5, 10, 12)

```

```
# NA, NA maps to a null/na feature; NaN, NaN maps to EMPTY
as_wkt(xy(NaN, NaN))
as_wkt(xy(NA, NA))
```

---

xy\_x

*XY vector extractors*

---

### Description

XY vector extractors

### Usage

xy\_x(x)

xy\_y(x)

xy\_z(x)

xy\_m(x)

### Arguments

x                   An `xy()` vector

### Value

Components of the `xy()` vector or NULL if the dimension is missing

### Examples

```
x <- xyz(1:5, 6:10, 11:15)
xy_x(x)
xy_y(x)
xy_z(x)
xy_m(x)
```

# Index

## \* datasets

- wk\_example, 31
- wk\_proj\_crs\_view, 46
  
- as.raster(), 5
- as\_crc(crc), 3
- as\_grd\_rct(grd), 5
- as\_grd\_xy(grd), 5
- as\_rct(rct), 18
- as\_wk\_handler(wk\_handle.wk\_crc), 35
- as\_wk\_trans(wk\_trans\_inverse), 50
- as\_wkb(wkb), 21
- as\_wkt(wkt), 23
- as\_xy(xy), 53
  
- base::qr.solve(), 49
  
- ceiling(), 7, 9, 10
- character(), 24
- crc, 3
- crc(), 4, 14
- crc\_center(crc\_x), 4
- crc\_r(crc\_x), 4
- crc\_x, 4
- crc\_y(crc\_x), 4
  
- dim(), 5
  
- floor(), 7, 9, 10
  
- graphics::lines(), 17, 44
- graphics::plot(), 18, 44
- graphics::points(), 17, 44
- graphics::polypath(), 17, 44, 45
- graphics::rasterImage(), 17
- grd, 5
- grd(), 5, 8–11, 13, 14
- grd\_cell, 7
- grd\_cell(), 9
- grd\_cell\_range(grd\_cell), 7
- grd\_cell\_range(), 7, 9, 10
  
- grd\_cell\_rct(grd\_cell), 7
- grd\_cell\_xy(grd\_cell), 7
- grd\_crop(grd\_subset), 9
- grd\_data\_extract(grd\_extract), 8
- grd\_data\_subset(grd\_subset), 9
- grd\_extend(grd\_subset), 9
- grd\_extract, 8
- grd\_extract(), 8
- grd\_extract\_nearest(grd\_extract), 8
- grd\_rct(grd), 5
- grd\_rct(), 7, 8, 10–13
- grd\_snap\_next, 9
- grd\_snap\_next(), 9
- grd\_snap\_previous(grd\_snap\_next), 9
- grd\_snap\_previous(), 9
- grd\_subset, 9
- grd\_subset(), 8, 9, 12
- grd\_summary, 11
- grd\_tile, 11
- grd\_tile(), 11
- grd\_tile\_summary(grd\_tile\_template), 12
- grd\_tile\_summary(), 12
- grd\_tile\_template, 12
- grd\_tile\_template(), 11
- grd\_xy(grd), 5
- grd\_xy(), 7, 8, 10–13
  
- handle\_wkt\_without\_vector\_size, 13
- handleable, 7, 8
  
- is\_handleable(wk\_handle.wk\_crc), 35
- is\_wk\_handler(wk\_handle.wk\_crc), 35
- is\_wk\_wkb(new\_wk\_wkb), 15
- is\_wk\_wkt(new\_wk\_wkt), 16
  
- list(), 22
  
- native raster, 8, 10
- new\_wk\_crc, 14
- new\_wk\_grd, 14



- new\_wk\_handler (wk\_handle.wk\_crc), 35
- new\_wk\_rct, 15
- new\_wk\_trans (wk\_trans\_inverse), 50
- new\_wk\_wkb, 15
- new\_wk\_wkb(), 22
- new\_wk\_wkt, 16
- new\_wk\_wkt(), 24
- new\_wk\_xy, 16
- new\_wk\_xym (new\_wk\_xy), 16
- new\_wk\_xyz (new\_wk\_xy), 16
- new\_wk\_xyzm (new\_wk\_xy), 16
  
- package\_version(), 29
- parse\_wkb (wkb), 21
- parse\_wkt (wkt), 23
- plot.wk\_crc (wk\_plot), 43
- plot.wk\_grd\_rct (plot.wk\_grd\_xy), 17
- plot.wk\_grd\_xy, 17
- plot.wk\_rct (wk\_plot), 43
- plot.wk\_wkb (wk\_plot), 43
- plot.wk\_wkt (wk\_plot), 43
- plot.wk\_xy (wk\_plot), 43
  
- rasterImage(), 5
- raw(), 22, 23
- rct, 18
- rct(), 5–8, 10, 13, 15, 18–20, 25, 26, 30, 32–34, 36–41, 43–45, 47, 48, 51–53
- rct\_contains (rct\_xmin), 19
- rct\_height (rct\_xmin), 19
- rct\_intersection (rct\_xmin), 19
- rct\_intersects (rct\_xmin), 19
- rct\_width (rct\_xmin), 19
- rct\_xmax (rct\_xmin), 19
- rct\_xmin, 19
- rct\_ymax (rct\_xmin), 19
- rct\_ymin (rct\_xmin), 19
- Reduce(), 26
- round(), 7, 10
  
- sf::st\_sfc(), 13, 25, 26, 30, 32–41, 43–45, 47, 48, 51–53
- sfc\_writer (wk\_writer.sfc), 52
  
- validate\_wk\_wkb (new\_wk\_wkb), 15
- validate\_wk\_wkt (new\_wk\_wkt), 16
- validate\_wk\_xy (new\_wk\_xy), 16
- validate\_wk\_xym (new\_wk\_xy), 16
- validate\_wk\_xyz (new\_wk\_xy), 16
  
- validate\_wk\_xyzm (new\_wk\_xy), 16
- vctrs-methods, 20
- vctrs::vec\_cast(), 21
- vctrs::vec\_ptype2(), 21
- vec\_cast.wk\_crc (vctrs-methods), 20
- vec\_cast.wk\_rct (vctrs-methods), 20
- vec\_cast.wk\_wkb (vctrs-methods), 20
- vec\_cast.wk\_wkt (vctrs-methods), 20
- vec\_cast.wk\_xy (vctrs-methods), 20
- vec\_cast.wk\_xym (vctrs-methods), 20
- vec\_cast.wk\_xyz (vctrs-methods), 20
- vec\_cast.wk\_xyzm (vctrs-methods), 20
- vec\_ptype2.wk\_crc (vctrs-methods), 20
- vec\_ptype2.wk\_rct (vctrs-methods), 20
- vec\_ptype2.wk\_wkb (vctrs-methods), 20
- vec\_ptype2.wk\_wkt (vctrs-methods), 20
- vec\_ptype2.wk\_xy (vctrs-methods), 20
- vec\_ptype2.wk\_xym (vctrs-methods), 20
- vec\_ptype2.wk\_xyz (vctrs-methods), 20
- vec\_ptype2.wk\_xyzm (vctrs-methods), 20
  
- wk\_affine\_compose (wk\_trans\_affine), 48
- wk\_affine\_fit (wk\_trans\_affine), 48
- wk\_affine\_identity (wk\_trans\_affine), 48
- wk\_affine\_invert (wk\_trans\_affine), 48
- wk\_affine\_rescale (wk\_trans\_affine), 48
- wk\_affine\_rotate (wk\_trans\_affine), 48
- wk\_affine\_scale (wk\_trans\_affine), 48
- wk\_affine\_translate (wk\_trans\_affine), 48
- wk\_bbox, 24
- wk\_bbox\_handler (wk\_bbox), 24
- wk\_chunk\_strategy\_coordinates (wk\_chunk\_strategy\_single), 25
- wk\_chunk\_strategy\_coordinates(), 26
- wk\_chunk\_strategy\_feature (wk\_chunk\_strategy\_single), 25
- wk\_chunk\_strategy\_feature(), 26
- wk\_chunk\_strategy\_single, 25
- wk\_clockwise (wk\_orient), 42
- wk\_collection (wk\_linestring), 40
- wk\_collection\_filter (wk\_linestring), 40
- wk\_coords (wk\_vertices), 50
- wk\_coords(), 49
- wk\_coords<- (wk\_vertices), 50
- wk\_count, 26
- wk\_count\_handler (wk\_count), 26
- wk\_counterclockwise (wk\_orient), 42
- wk\_crs, 27

- wk\_crs(), 5
- wk\_crs<- (wk\_crs), 27
- wk\_crs\_auto (wk\_crs\_inherit), 28
- wk\_crs\_auto(), 28, 29
- wk\_crs\_auto\_value (wk\_crs\_inherit), 28
- wk\_crs\_equal, 28
- wk\_crs\_equal(), 28
- wk\_crs\_equal\_generic (wk\_crs\_equal), 28
- wk\_crs\_equal\_generic(), 28
- wk\_crs\_inherit, 28
- wk\_crs\_inherit(), 28
- wk\_crs\_longlat (wk\_crs\_inherit), 28
- wk\_crs\_output (wk\_crs), 27
- wk\_crs\_proj\_definition, 29
- wk\_crs\_projjson  
(wk\_crs\_proj\_definition), 29
- wk\_debug, 30
- wk\_debug(), 35, 51
- wk\_debug\_filter (wk\_debug), 30
- wk\_drop\_m (wk\_set\_z), 46
- wk\_drop\_z (wk\_set\_z), 46
- wk\_envelope (wk\_bbox), 24
- wk\_envelope\_handler (wk\_bbox), 24
- wk\_example, 31
- wk\_example\_wkt (wk\_example), 31
- wk\_flatten, 32
- wk\_flatten\_filter (wk\_flatten), 32
- wk\_format, 33
- wk\_format(), 35
- wk\_geodesic\_inherit (wk\_is\_geodesic), 39
- wk\_geometry\_type (wk\_meta), 41
- wk\_geometry\_type\_label (wk\_meta), 41
- wk\_handle (wk\_handle.wk\_crc), 35
- wk\_handle(), 13, 25, 26, 30, 32–41, 43–45,  
47, 48, 51–53
- wk\_handle.data.frame, 34
- wk\_handle.wk\_crc, 35
- wk\_handle.wk\_grd\_rct  
(wk\_handle.wk\_grd\_xy), 36
- wk\_handle.wk\_grd\_xy, 36
- wk\_handle\_slice  
(wk\_handle\_slice.data.frame),  
37
- wk\_handle\_slice.data.frame, 37
- wk\_handler, 13, 30, 32, 34, 36–40, 43, 47, 51,  
53
- wk\_identity, 38
- wk\_identity\_filter (wk\_identity), 38
- wk\_is\_geodesic, 39
- wk\_is\_geodesic<- (wk\_is\_geodesic), 39
- wk\_is\_geodesic\_output (wk\_crs), 27
- wk\_linestring, 40
- wk\_linestring\_filter (wk\_linestring), 40
- wk\_meta, 41
- wk\_meta(), 26
- wk\_meta\_handler (wk\_meta), 41
- wk\_orient, 42
- wk\_orient\_filter (wk\_orient), 42
- wk\_platform\_endian (wkb), 21
- wk\_plot, 43
- wk\_polygon (wk\_linestring), 40
- wk\_polygon\_filter (wk\_linestring), 40
- wk\_problems, 45
- wk\_problems(), 35, 51
- wk\_problems\_handler (wk\_problems), 45
- wk\_proj\_crs\_json (wk\_proj\_crs\_view), 46
- wk\_proj\_crs\_view, 46
- wk\_restore (wk\_identity), 38
- wk\_restore.data.frame  
(wk\_handle.data.frame), 34
- wk\_restore.sf (wk\_handle.data.frame), 34
- wk\_restore.tbl\_df  
(wk\_handle.data.frame), 34
- wk\_set\_crs (wk\_crs), 27
- wk\_set\_geodesic (wk\_is\_geodesic), 39
- wk\_set\_m (wk\_set\_z), 46
- wk\_set\_z, 46
- wk\_trans, 49
- wk\_trans\_affine, 48
- wk\_trans\_affine(), 49
- wk\_trans\_explicit, 49
- wk\_trans\_explicit(), 51
- wk\_trans\_inverse, 50
- wk\_trans\_set (wk\_set\_z), 46
- wk\_transform, 47
- wk\_transform\_filter (wk\_transform), 47
- wk\_translate (wk\_translate.sfc), 48
- wk\_translate.data.frame  
(wk\_handle.data.frame), 34
- wk\_translate.sf (wk\_handle.data.frame),  
34
- wk\_translate.sfc, 48
- wk\_translate.tbl\_df  
(wk\_handle.data.frame), 34
- wk\_vector\_meta (wk\_meta), 41
- wk\_vector\_meta\_handler (wk\_meta), 41

wk\_vertex\_filter (wk\_vertices), 50  
wk\_vertices, 50  
wk\_vertices(), 52  
wk\_void, 51  
wk\_void(), 35  
wk\_void\_handler (wk\_void), 51  
wk\_writer (wk\_writer.sfc), 52  
wk\_writer(), 35  
wk\_writer.sfc, 52  
wkb, 21  
wkb(), 13, 15, 17, 22, 25, 26, 30, 32–41,  
43–45, 47, 48, 51–53  
wkb\_to\_hex, 22  
wkb\_translate\_wkb (wkb\_translate\_wkt),  
23  
wkb\_translate\_wkt, 23  
wkb\_writer (wk\_writer.sfc), 52  
wkb\_writer(), 52  
wkt, 23  
wkt(), 13, 16, 17, 25, 26, 30–41, 43–45, 47,  
48, 51–53  
wkt\_format\_handler (wk\_format), 33  
wkt\_translate\_wkb (wkb\_translate\_wkt),  
23  
wkt\_translate\_wkt (wkb\_translate\_wkt),  
23  
wkt\_writer (wk\_writer.sfc), 52  
  
xy, 53  
xy(), 8, 13, 17, 25, 26, 30, 32–41, 43–45,  
47–49, 51–53, 55  
xy\_dims (xy), 53  
xy\_m (xy\_x), 55  
xy\_writer (wk\_writer.sfc), 52  
xy\_x, 55  
xy\_y (xy\_x), 55  
xy\_z (xy\_x), 55  
xym (xy), 53  
xym(), 47, 49, 51  
xyz (xy), 53  
xyz(), 47, 49, 51  
xyzm (xy), 53  
xyzm(), 47, 49, 51