Package ‘wk’

July 13, 2021

**Title**  Lightweight Well-Known Geometry Parsing

**Version**  0.5.0

**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 only useful in R 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

**RoxygenNote**  7.1.1

**LinkingTo**  cpp11

**SystemRequirements**  C++11

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

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

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

**Config/testthat/edition**  3

**NeedsCompilation**  yes

**Author**  Dewey Dunnington [aut, cre] (<https://orcid.org/0000-0002-9415-4582>),
              Edzer Pebesma [aut] (<https://orcid.org/0000-0001-8049-7069>)

**Repository**  CRAN

**Date/Publication**  2021-07-13 04:50:02 UTC
R topics documented:

crc .......................................................... 3
handle_wkt_without_vector_size ............................. 4
new_wk_crc .................................................. 4
new_wk_rct .................................................. 5
new_wk_wkb .................................................. 5
new_wk_wkt .................................................. 6
new_wk_xy .................................................... 6
rct ............................................................ 7
vctrs-methods ................................................. 8
wkb ............................................................ 9
wkb_format .................................................... 10
wkt ............................................................ 10
wk_bbox ......................................................... 11
wk_count ....................................................... 12
wk_crs ........................................................ 13
wk_crs_equal ................................................ 13
wk_crs_inherit ............................................... 14
wk_debug ....................................................... 15
wk_flatten ..................................................... 15
wk_format ...................................................... 16
wk_handle.data.frame ....................................... 17
wk_handle.sfg ............................................... 18
wk_identity .................................................. 20
wk_linestring ............................................... 21
wk_meta ....................................................... 22
wk_plot ......................................................... 23
wk_problems ................................................ 25
wk_set_z ....................................................... 26
wk_transform ................................................ 27
wk_translate.sfc ........................................... 27
wk_trans_affine ............................................. 28
wk_trans_inverse .......................................... 29
wk_vertices .................................................. 29
wk_void ....................................................... 30
wk_writer.sfc ............................................... 31
xy ........................................................... 32

Index 34
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)
**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., `wkb()`, `wkt()`, `xy()`, `rct()`, or `sf::st_sfc()`) for which `wk_handle()` is defined.
- `handler`: A `wk_handler` 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 `crc()`
- `crs`: A value to be propagated as the CRS for this vector.
new_wk_rct

S3 details for rct objects

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 rct()

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

new_wk_wkb

S3 Details for wk_wkb

Description

S3 Details for wk_wkb

Usage

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

validate_wk_wkb(x)

is_wk_wkb(x)

Arguments

x A (possibly) wkb() vector

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

S3 Details for wk_wkt

Description

S3 Details for wk_wkt

Usage

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

is_wk_wkt(x)

validate_wk_wkt(x)

Arguments

x A (possibly) wkt() vector

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

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_xy(x = list(x = double(), y = double(), z = double()), crs = NULL)

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

new_wk_xym(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.

---

**Description**

2D rectangle vectors

**Usage**

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

```r
as_rct(x, ...)
```

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

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

```r
## 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 `rct()`.
- **...**: Extra arguments passed to `as_rct()`.

**Value**

A vector along the recycled length of bounds.

**Examples**

```r
crt(1, 2, 3, 4)
```
### Description

Vctrs methods

### Usage

```r
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`. 
Mark lists of raw vectors as well-known binary

Usage

wkb(x = list(), crs = wk_crs_auto())
parse_wkb(x, crs = wk_crs_auto())
wk_platform_endian()
as_wkb(x, ...)

## Default S3 method:
as_wkb(x, ...)

## S3 method for class 'character'
as_wkb(x, ..., crs = NULL)

## S3 method for class 'wk_wkb'
as_wkb(x, ...)

## S3 method for class 'blob'
as_wkb(x, ..., crs = NULL)

## S3 method for class 'WKB'
as_wkb(x, ..., crs = NULL)

Arguments

x A list() of raw() vectors or NULL.

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

Value

A new_wk_wkb()

Examples

wkb(wkt_translate_wkb("POINT (20 10)"))
### wkb_format

**Deprecated functions**

**Description**

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

**Usage**

```r
wkb_format(wkb, max.coords = 3, precision = 6, trim = TRUE)
wkt_format(wkt, max.coords = 3, precision = 6, trim = TRUE)

wkb_problems(wkb)
wkt_problems(wkt)

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

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

**Arguments**

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

### wkt

**Mark character vectors as well-known text**

**Description**

Mark character vectors as well-known text
Usage

wkt(x = character(), crs = wk_crs_auto())

parse_wkt(x, crs = wk_crs_auto())

as_wkt(x, ...)

## Default S3 method:
as_wkt(x, ...)

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

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

Arguments

x A character() vector containing well-known text.
crs A value to be propagated as the CRS for this vector.
... Unused

Value

A new_wk_wkt()

Examples

wkt("POINT (20 10)")

---

wk_bbox 2D bounding rectangles

Description

2D bounding rectangles

Usage

wk_bbox(handleable, ...)

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

wk_bbox_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)"))

## 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, ...)

## 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

\[
\begin{align*}
\text{wk\_count(as\_wkt("LINESTRING (0 0, 1 1")"))} \\
\text{wk\_count(as\_wkb("LINESTRING (0 0, 1 1")"))}
\end{align*}
\]

---

**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**

\[
\begin{align*}
\text{wk\_crs(x)} \\
\text{\# S3 method for class 'wk\_vctr'} \\
\text{wk\_crs(x)} \\
\text{\# S3 method for class 'wk\_rcrd'} \\
\text{wk\_crs(x)} \\
\text{wk\_crs(x) <- value} \\
\text{wk\_set\_crs(x, crs)} \\
\text{wk\_crs\_output(...)}
\end{align*}
\]

**Arguments**

- **x, ...** Objects whose "crs" attribute is used to carry a CRS.
- **crs, value** An object that can be interpreted as a CRS

---

**wk\_crs\_equal**

*Compare CRS objects*

---

**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

x, y  
Objects stored in the crs attribute of a vector.

...  
Unused

Value

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

<table>
<thead>
<tr>
<th>wk_crs_inherit</th>
<th>Special CRS values</th>
</tr>
</thead>
</table>

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_auto()

wk_crs_auto_value(x, crs)

Arguments

x  
A raw input to a constructor whose length and crs attribute is used to determine the default CRS returned by wk_crs_auto().

crs  
A value for the coordinate reference system supplied by the user.

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_debug  

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_flatten  

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., `wkb()`, `wkt()`, `xy()`, `rct()`, or `sf::st_sfc()`) for which `wk_handle()` is defined.

max_depth  The maximum (outer) depth to remove.

handler  A `wk_handler` object.

add_details  Use `TRUE` to add a “wk_details” attribute, which contains columns `feature_id`, `part_id`, and `ring_id`.

Value

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

Examples

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

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

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

Arguments

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

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.

max_coords  The maximum number of coordinates to include in the output.

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

Value

A character vector of abbreviated well-known text.

Examples

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

---

**Description**

Use `data.frame` with `wk`

**Usage**

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

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

## S3 method for class 'data.frame'
wk_crs(x)

## S3 method for class 'data.frame'
wk_set_crs(x, crs)

## 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, ..., .env = parent.frame())

## S3 method for class 'tbl_df'
wk_translate(handleable, to, ..., .env = parent.frame())

## 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., `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.
- **.env**: Passed to `getS3method()`, which is used to find the column in a `data.frame()` for which a `wk_handle()` method is defined.
- **x**: Objects whose "crs" attribute is used to carry a CRS.
- **crs**: An object that can be interpreted as a CRS.
- **result**: The result of a filter operation intended to be a transformation.
- **to**: A prototype object.

**Examples**

```r
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())))
```

---

**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**

```r
## 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, ...)
```
## 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, ...)

new_wk_handler(handler_ptr, subclass = character())

is_wk_handler(handler)

as_wk_handler(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.
- **n_segments**, **resolution**: The number of segments to use when approximating a circle. The default uses `getOption("wk.crc_n_segments")` so that this value can be set for implicit conversions (e.g., `as_wkb()`). Alternatively, set the minimum distance between points on the circle (used to estimate `n_segments`). The default is obtained using `getOption("wk.crc_resolution")`.
- **handler_ptr**: An external pointer to a newly created WK handler.
- **subclass**: The handler subclass.
**wk_identity**

**Value**

A WK handler.

---

**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., `wkb()`, `wkt()`, `xy()`, `rct()`, or `sf::st_sfc()`) for which `wk_handle()` is defined.
- `...` Passed to the `wk_handle()` method.
- `handler` A `wk_handler` 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_linestring**

Create lines, polygons, and collections

### Description

Create lines, polygons, and collections

### Usage

```r
wk_linestring(handleable, feature_id = 1L, ...)
```

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

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

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

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

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

### Arguments

- **handleable**: A geometry vector (e.g., `wkb()`, `wkt()`, `xy()`, `rct()`, or `sf::st_sfc()`) for which `wk_handle()` is defined.
- **feature_id**: An identifier where changes in sequential values indicate a new feature. This is recycled silently as needed.
- **...**: Passed to the `wk_handle()` method.
- **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 `wk_handler` object.

### Value

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

```r
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)))
```

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

```r
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., `wkb()`, `wkt()`, `xy()`, `rct()`, or `sf::st_sfc()`) for which `wk_handle()` is defined.
- **...**: Passed to the `wk_handle()` 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.

Examples

```r
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"))
```

Description

Plot well-known geometry vectors

Usage

```r
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., `wkb()`, `wkt()`, `xy()`, `rct()`), or `sf::st_sfc()` for which `wk_handle()` is defined.
- ...: 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.
- asp, xlab, ylab: Passed to `graphics::plot()`
- bbox: The limits of the plot as a `rct()` or compatible object.
- rule: The rule to use for filling polygons (see `graphics::polypath()`)
- add: Should a new plot be created, or should handleable be added to the existing plot?
- x: A `wkb()` or `wkt()`
**Value**

The input, invisibly.

**Examples**

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

---

**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**

```r
wk_problems(handleable, ...)
wk_problems_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 character vector of parsing errors. `NA` signifies that there was no parsing error.

**Examples**

```r
wk_problems(new_wk_wkt(c("POINT EMTPY", "POINT (20 30)")))
wk_handle(
   new_wk_wkt(c("POINT EMTPY", "POINT (20 30)")),
   wk_problems_handler()
)
```
**wk_set_z**  
*Set coordinate values*

### 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., `wkb()`, `wkt()`, `xy()`, `rct()`, or `sf::st_sfc()`) for which `wk_handle()` 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 `wk_handle()` method.
- **value**: An `xy()`, `xyz()`, `xym()`, or `xyzm()` 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**

*Apply coordinate transformations*

**Description**

Apply coordinate transformations

**Usage**

```r
wk_transform(handleable, trans, ...)
wk_transform_filter(handler, trans)
```

**Arguments**

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

**Examples**

```r
wk_transform(xy(0, 0), wk_affine Translate (2, 3))
```

---

**wk_translate.sfc**

*Translate geometry vectors*

**Description**

Translate geometry vectors

**Usage**

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

**Examples**

```r
wk_translate(handleable, to, ...)
```
Arguments

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

to A prototype object.

Passed to the `wk_handle()` 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 `base::qr.solve()`).
rct_in, rct_out  The input and output bounds
...  Zero or more transforms in the order they should be applied.
x  A `wk_trans_affine()`
**wk_trans_inverse**

**Description**

Generic transform class

**Usage**

```r
wk_trans_inverse(trans, ...)  
as_wk_trans(x, ...)  
```  
```r
## S3 method for class 'wk_trans'  
as_wk_trans(x, ...)  
```  
```r
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**

```r
wk_vertices(handleable, ...)  
wk_coords(handleable, ...)  
wk_vertex_filter(handler, add_details = FALSE)  
```
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.
add_details  Use TRUE to add a "wk_details" attribute, which contains columns `feature_id`, `part_id`, and `ring_id`.

Value

• `wk_vertices()` extracts vertices and returns them 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_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., `wkb()`, `wkt()`, `xy()`, `rct()`, or `sf::st_sfc()`) for which `wk_handle()` is defined.
...

Value

NULL
Examples

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

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

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

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

sfc_writer()

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., `wkb()`, `wkt()`, `xy()`, `rct()`, or `sf::st_sfc()`) for which `wk_handle()` is defined.
- **...**: Passed to the writer constructor.
- **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**

```r
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:

```r
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

```r
xy(1:5, 1:5)
xyz(1:5, 1:5, 10)
xym(1:5, 1:5, 10)
xyzm(1:5, 1:5, 10, 12)
```
## Index

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_crc(crc)</td>
<td>3</td>
</tr>
<tr>
<td>as_rct(rct)</td>
<td>7</td>
</tr>
<tr>
<td>as_wk_handler(wk_handle.sfg)</td>
<td>18</td>
</tr>
<tr>
<td>as_wk_trans(wk_trans_inverse)</td>
<td>29</td>
</tr>
<tr>
<td>as_wkb(wkb)</td>
<td>9</td>
</tr>
<tr>
<td>as_wkt(wkt)</td>
<td>10</td>
</tr>
<tr>
<td>as_xy(xy)</td>
<td>32</td>
</tr>
<tr>
<td>base::qr.solve()</td>
<td>28</td>
</tr>
<tr>
<td>character()</td>
<td>11</td>
</tr>
<tr>
<td>crc</td>
<td>3</td>
</tr>
<tr>
<td>crc()</td>
<td>4</td>
</tr>
<tr>
<td>data.frame()</td>
<td>18</td>
</tr>
<tr>
<td>getS3method()</td>
<td>18</td>
</tr>
<tr>
<td>graphics::lines()</td>
<td>24</td>
</tr>
<tr>
<td>graphics::plot()</td>
<td>24</td>
</tr>
<tr>
<td>graphics::points()</td>
<td>24</td>
</tr>
<tr>
<td>graphics::polypath()</td>
<td>24</td>
</tr>
<tr>
<td>handle_wkt_without_vector_size</td>
<td>4</td>
</tr>
<tr>
<td>is_wk_handler(wk_handle.sfg)</td>
<td>18</td>
</tr>
<tr>
<td>is_wk_wkb(new_wk_wkb)</td>
<td>5</td>
</tr>
<tr>
<td>is_wk_wkt(new_wk_wkt)</td>
<td>6</td>
</tr>
<tr>
<td>list()</td>
<td>9</td>
</tr>
<tr>
<td>new_wk_crc</td>
<td>4</td>
</tr>
<tr>
<td>new_wk_handler(wk_handle.sfg)</td>
<td>18</td>
</tr>
<tr>
<td>new_wk_rct</td>
<td>5</td>
</tr>
<tr>
<td>new_wk_trans(wk_trans_inverse)</td>
<td>29</td>
</tr>
<tr>
<td>new_wkb</td>
<td>5</td>
</tr>
<tr>
<td>new_wk_wkb()</td>
<td>9</td>
</tr>
<tr>
<td>new_wk_wkt()</td>
<td>11</td>
</tr>
<tr>
<td>new_wk_xy</td>
<td>6</td>
</tr>
<tr>
<td>new_wk_xym(new_wk_xy)</td>
<td>6</td>
</tr>
<tr>
<td>new_wk_xyz(new_wk_xy)</td>
<td>6</td>
</tr>
<tr>
<td>new_wk_xyzm(new_wk_xy)</td>
<td>6</td>
</tr>
<tr>
<td>parse_wkb(wkb)</td>
<td>9</td>
</tr>
<tr>
<td>parse_wkt(wkt)</td>
<td>10</td>
</tr>
<tr>
<td>plot.wk_crc(wk_plot)</td>
<td>23</td>
</tr>
<tr>
<td>plot.wk_rct(wk_plot)</td>
<td>23</td>
</tr>
<tr>
<td>plot.wk_wkb(wk_plot)</td>
<td>23</td>
</tr>
<tr>
<td>plot.wk_wkt(wk_plot)</td>
<td>23</td>
</tr>
<tr>
<td>plot.wk_xy(wk_plot)</td>
<td>23</td>
</tr>
<tr>
<td>raw()</td>
<td>9, 10</td>
</tr>
<tr>
<td>rct</td>
<td>7</td>
</tr>
<tr>
<td>rct()</td>
<td>4, 5, 7, 12, 15, 16, 18–22, 24–28, 30, 32</td>
</tr>
<tr>
<td>sf::st_sfc()</td>
<td>4, 12, 15, 16, 18–22, 24–28, 30, 32</td>
</tr>
<tr>
<td>sfc_writer(wk_writer.sfc)</td>
<td>31</td>
</tr>
<tr>
<td>validate_wk_wkb(new_wk_wkb)</td>
<td>5</td>
</tr>
<tr>
<td>validate_wk_wkt(new_wk_wkt)</td>
<td>6</td>
</tr>
<tr>
<td>validate_wk_xy(new_wk_xy)</td>
<td>6</td>
</tr>
<tr>
<td>validate_wk_xym(new_wk_xy)</td>
<td>6</td>
</tr>
<tr>
<td>validate_wk_xyz(new_wk_xy)</td>
<td>6</td>
</tr>
<tr>
<td>validate_wk_xyzm(new_wk_xy)</td>
<td>6</td>
</tr>
<tr>
<td>vctrs-methods</td>
<td>8</td>
</tr>
<tr>
<td>vec_cast.wk_crc(vctrs-methods)</td>
<td>8</td>
</tr>
<tr>
<td>vec_cast.wk_rct(vctrs-methods)</td>
<td>8</td>
</tr>
<tr>
<td>vec_cast.wk_wkb(vctrs-methods)</td>
<td>8</td>
</tr>
<tr>
<td>vec_cast.wk_wkt(vctrs-methods)</td>
<td>8</td>
</tr>
<tr>
<td>vec_cast.wk_xy(vctrs-methods)</td>
<td>8</td>
</tr>
<tr>
<td>vec_cast.wk_xym(vctrs-methods)</td>
<td>8</td>
</tr>
<tr>
<td>vec_cast.wk_xyz(vctrs-methods)</td>
<td>8</td>
</tr>
<tr>
<td>vec_cast.wk_xyzm(vctrs-methods)</td>
<td>8</td>
</tr>
<tr>
<td>vec_ptype2.wk_crc(vctrs-methods)</td>
<td>8</td>
</tr>
<tr>
<td>vec_ptype2.wk_rct(vctrs-methods)</td>
<td>8</td>
</tr>
<tr>
<td>vec_ptype2.wk_wkb(vctrs-methods)</td>
<td>8</td>
</tr>
<tr>
<td>vec_ptype2.wk_wkt(vctrs-methods)</td>
<td>8</td>
</tr>
<tr>
<td>vec_ptype2.wk_xy(vctrs-methods)</td>
<td>8</td>
</tr>
</tbody>
</table>
wk_void_handler (wk_void), 30
wk_writer (wk_writer.sfc), 31
wk_writer(), 18
wk_writer.data.frame
    (wk_handle.data.frame), 17
wk_writer.sfc, 31
wkb, 9
wkb(), 4, 5, 12, 15, 16, 18–22, 24–28, 30, 32
wkb_format, 10
wkb_problems (wkb_format), 10
wkb_translate_wkb (wkb_format), 10
wkb_translate_wkt (wkb_format), 10
wkb_writer (wk_writer.sfc), 31
wkb_writer(), 31
wkt, 10
wkt(), 4, 6, 12, 15, 16, 18–22, 24–28, 30, 32
wkt_format (wkb_format), 10
wkt_format_handler (wk_format), 16
wkt_problems (wkb_format), 10
wkt_translate_wkb (wkb_format), 10
wkt_translate_wkt (wkb_format), 10
wkt_writer (wk_writer.sfc), 31

xy, 32
xy(), 4, 7, 12, 15, 16, 18–22, 24–28, 30, 32
xy_dims (xy), 32
xy_writer (wk_writer.sfc), 31
xym (xy), 32
xym(), 26
xyz (xy), 32
xyz() 26
xyzm (xy), 32
xyzm() 26