Package: gridmaker (via r-universe)

May 16, 2026

Title Create INSPIRE-Compliant Grids with IDs

Version 0.1.0

Description Creates GISCO compatible and INSPIRE-compliant grids with IDs that look like 'CRS3035RES1000mN3497000E4448000' or '1kmN3497E4447'. Input can be 'sf', 'sfc' objects or bounding boxes. Output can be 'sf' polygons, 'sf' centroids, or just 'data.frame' with grid cell center or bottom left corner coordinates. The resulting grids are always aligned to rounded coordinates as per INSPIRE requirements.

License MIT + file LICENSE

URL https://github.com/e-kotov/gridmaker, https://www.ekotov.pro/gridmaker/

BugReports https://github.com/e-kotov/gridmaker/issues

Imports later, promises, ps, sf, sfheaders, terra

Suggests carrier, furrr, future, giscoR, knitr, mirai, purrr, quarto, readr, rmarkdown, testthat (>= 3.0.0), withr

VignetteBuilder quarto

Config/testthat/edition 3

Encoding UTF-8

Language en

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://e-kotov.r-universe.dev

Date/Publication 2026-01-16 15:13:29 UTC

RemoteUrl https://github.com/e-kotov/gridmaker

RemoteRef HEAD

RemoteSha 944b015921bb18b8b698a399438d6b0025f9c3ef

Contents

`inspire_grid` *Create or Reconstruct an INSPIRE Grid*

Description

Generates a standard-compliant spatial grid aligned to the CRS origin. This function acts as a unified interface:

- If `x` is a spatial object (extent), it creates a new grid (calling `inspire_grid_from_extent`).
- If `x` is a character vector (INSPIRE IDs), it reconstructs the grid (calling `inspire_grid_from_ids`).

It combines high performance for large areas (using `sfheaders`) with a flexible and robust set of features for input handling and output formatting, including INSPIRE-compliant grid IDs and automatic parallel processing with `mirai` and `future` backends.

This function takes a vector of INSPIRE-compliant IDs and derives a regular spatial grid from it. For generating a spatial grid from a spatial extent, see `inspire_grid_from_extent`.

Usage

```
inspire_grid(
  x,
  cellsize_m = NULL,
  crs = NULL,
  output_type = "sf_polygons",
  clip_to_input = FALSE,
  use_convex_hull = FALSE,
  buffer_m = 0,
  id_format = "both",
  axis_order = "NE",
  include_llc = TRUE,
  point_type = "centroid",
  parallel = "auto",
  quiet = getOption("gridmaker.quiet", FALSE),
  dsn = NULL,
  layer = NULL,
  max_memory_gb = NULL,
  include_rat = FALSE,
  ...
)
```

```
## S3 method for class 'sf'
inspire_grid(
  x,
  cellsize_m = NULL,
  crs = NULL,
  output_type = "sf_polygons",
  clip_to_input = FALSE,
  use_convex_hull = FALSE,
  buffer_m = 0,
  id_format = "both",
  axis_order = "NE",
  include_llc = TRUE,
  point_type = "centroid",
  parallel = "auto",
  quiet = getOption("gridmaker.quiet", FALSE),
  dsn = NULL,
  layer = NULL,
  max_memory_gb = NULL,
  include_rat = FALSE,
  ...
)

## S3 method for class 'sfc'
inspire_grid(
  x,
  cellsize_m = NULL,
  crs = NULL,
  output_type = "sf_polygons",
  clip_to_input = FALSE,
  use_convex_hull = FALSE,
  buffer_m = 0,
  id_format = "both",
  axis_order = "NE",
  include_llc = TRUE,
  point_type = "centroid",
  parallel = "auto",
  quiet = getOption("gridmaker.quiet", FALSE),
  dsn = NULL,
  layer = NULL,
  max_memory_gb = NULL,
  include_rat = FALSE,
  ...
)

## S3 method for class 'bbox'
inspire_grid(
  x,
  cellsize_m = NULL,
```

```
  crs = NULL,
  output_type = "sf_polygons",
  clip_to_input = FALSE,
  use_convex_hull = FALSE,
  buffer_m = 0,
  id_format = "both",
  axis_order = "NE",
  include_llc = TRUE,
  point_type = "centroid",
  parallel = "auto",
  quiet = getOption("gridmaker.quiet", FALSE),
  dsn = NULL,
  layer = NULL,
  max_memory_gb = NULL,
  include_rat = FALSE,
  ...
)

## S3 method for class 'numeric'
inspire_grid(
  x,
  cellsize_m = NULL,
  crs = NULL,
  output_type = "sf_polygons",
  clip_to_input = FALSE,
  use_convex_hull = FALSE,
  buffer_m = 0,
  id_format = "both",
  axis_order = "NE",
  include_llc = TRUE,
  point_type = "centroid",
  parallel = "auto",
  quiet = getOption("gridmaker.quiet", FALSE),
  dsn = NULL,
  layer = NULL,
  max_memory_gb = NULL,
  include_rat = FALSE,
  ...
)

## S3 method for class 'matrix'
inspire_grid(
  x,
  cellsize_m = NULL,
  crs = NULL,
  output_type = "sf_polygons",
  clip_to_input = FALSE,
  use_convex_hull = FALSE,
```

```
  buffer_m = 0,
  id_format = "both",
  axis_order = "NE",
  include_llc = TRUE,
  point_type = "centroid",
  parallel = "auto",
  quiet = getOption("gridmaker.quiet", FALSE),
  dsn = NULL,
  layer = NULL,
  max_memory_gb = NULL,
  include_rat = FALSE,
  ...
)

## S3 method for class 'character'
inspire_grid(
  x,
  cellsize_m = NULL,
  crs = NULL,
  output_type = "sf_polygons",
  clip_to_input = FALSE,
  use_convex_hull = FALSE,
  buffer_m = 0,
  id_format = "both",
  axis_order = "NE",
  include_llc = TRUE,
  point_type = "llc",
  parallel = "auto",
  quiet = getOption("gridmaker.quiet", FALSE),
  dsn = NULL,
  layer = NULL,
  max_memory_gb = NULL,
  include_rat = FALSE,
  ...
)

inspire_grid_from_extent(
  grid_extent,
  cellsize_m,
  crs = NULL,
  output_type = "sf_polygons",
  clip_to_input = FALSE,
  use_convex_hull = FALSE,
  buffer_m = 0,
  id_format = "both",
  axis_order = "NE",
  include_llc = TRUE,
  point_type = "centroid",
```

```
  parallel = "auto",
  quiet = getOption("gridmaker.quiet", FALSE),
  dsn = NULL,
  layer = NULL,
  max_memory_gb = NULL,
  include_rat = FALSE,
  ...
)

inspire_grid_from_ids(
  ids,
  point_type = c("llc", "centroid"),
  output_type = c("sf_polygons", "sf_points", "dataframe"),
  include_llc = TRUE,
  id_format = c("both", "long", "short"),
  axis_order = c("NE", "EN"),
  quiet = getOption("gridmaker.quiet", FALSE),
  dsn = NULL,
  layer = NULL,
  ...
)
```

Arguments

x	The main input object: either a spatial object (extent) or a character vector (INSPIRE IDs).
cellsize_m	A single integer representing the grid cell size in metres (e.g., 1000 for a 1 km grid). Required for spatial inputs.
crs	The coordinate reference system (CRS) for the output grid. Accepts various formats handled by `sf::st_crs()`: an integer or numeric EPSG code (e.g., `3035`), a string representation like `"epsg:3035"`, or a `crs` object. If `NULL` (default), the CRS is inherited from the spatial input. If the input also lacks a CRS, the function will stop with an error.
output_type	The class of the output object: `"sf_polygons"` (default) creates a spatial object with polygon geometries, `"sf_points"` creates an `sf` object with point geometries, `"dataframe"` creates a data frame with grid cell centroid coordinates (`X_centroid`, `Y_centroid`), and `"spatraster"` creates a `terra::SpatRaster` object with grid cell IDs stored as factor levels (Raster Attribute Table). **Note:** `"spatraster"` is only supported by `inspire_grid_from_extent()`, not by `inspire_grid_from_ids()`.
clip_to_input	A logical value. If `TRUE`, the grid is filtered to include only cells that intersect the spatial input. This does not cut cell geometries.
use_convex_hull	A logical value. If `TRUE` and `clip_to_input` is `TRUE`, the grid is clipped to the convex hull of the input geometry, which can be faster and simpler than using a complex polygon.
buffer_m	A numeric value. If `clip_to_input` is `TRUE`, this specifies a buffer distance in metres to apply to the spatial input before clipping. Defaults to `0` (no buffer).

`id_format` A character string specifying which grid cell IDs to generate. Options are `"both"` (default), `"long"`, `"short"`, or `"none"`.

`axis_order` A character string specifying the coordinate order for the output Short INSPIRE IDs. This parameter is **only used when** `id_format` **is** `"short"` **or** `"both"`. It can be one of:

- `"NE"` (the default) to produce the format `{cellsize}N{y}E{x}`.
- `"EN"` to produce the format `{cellsize}E{x}N{y}` (e.g. this format is used in Danish national grid).

`include_llc` A logical value. If `TRUE` (default), columns for the lower-left corner coordinates (`X_LLC`, `Y_LLC`) of each cell are included in the output.

`point_type` A character string determining the location of the points when `output_type = "sf_points"`: `"centroid"` for the center of the cell, or `"llc"` for the lower-left corner. Default is `"llc"`.

`parallel` Controls parallel execution. Options are:

- `'auto'` **(default):** Automatically detects and uses a configured `mirai` or `future` backend if one is available. If both are set, it prefers the one with more available workers and issues a warning. If neither is configured, it runs sequentially.
- `TRUE`**:** Forces the function to attempt parallel execution. It will raise an error if a valid parallel backend (with >1 worker) is not configured.
- `FALSE`**:** Forces the function to run in single-threaded sequential mode.

For parallelism, you must configure a backend *before* calling this function, for example: `mirai::daemons(8)` or `future::plan("multisession", workers = 8)`. **Performance tip:** Benchmarks show 4-8 workers provide optimal performance for most grid sizes. Using >8 workers typically yields diminishing returns due to I/O bottlenecks. The function automatically limits active workers for small grids to minimize overhead: <50k cells use max 4 workers, <500k cells use max 8 workers, <2M cells use max 16 workers. This automatic limiting can be overridden by setting `options(gridmaker.tile_multiplier)`. **Note:** Parallel processing support depends on the backend and output type:

- `mirai` **backend:** Supports parallel processing for all outputs, including efficient disk streaming. This is the recommended modern backend that also runs asynchronously, which allows it to pass chunks of grid that are ready to the disk writer and therefore makes streaming to disk efficient, as there is no need to all data in memory first.
- `future` **backend:** Supports parallel processing **only** for in-memory vector generation (`sf`, `dataframe`). It does not support raster output or disk-based streaming (falls back to sequential), because it would need to first accumualte all data in memory before writing to disk, negating the benefits of streaming.

`quiet` Logical value. If `TRUE`, all progress messages and progress bars are suppressed. Defaults to `FALSE`.

`dsn` The destination for the output grid. For sf objects, this is passed to `sf::st_write`. For `spatraster` output, this uses `terra::writeRaster`. This can be a file path (e.g., `"path/to/grid.gpkg"` for vector data or `"path/to/grid.tif"` for raster

data) or a database connection string. If dsn is provided, the grid is written to the specified location instead of being returned as an object.

Supported vector formats for chunked disk writes:

- .gpkg (GeoPackage) - **Recommended** - Best balance of speed, compatibility, and modern features
- .parquet, .geoparquet (GeoParquet) - Modern columnar format, excellent for large grids (requires sf 1.0+/GDAL 3.5+)
- .shp (Shapefile) - Widely used, fast writes, but has limitations (10-char field names, 2GB limit)
- .geojson, .json (GeoJSON) - Web-friendly, works but slower for large grids
- .geojsonl, .geojsonseq (GeoJSONSeq) - Newline-delimited GeoJSON
- .sqlite (SQLite/SpatiaLite) - Database format (GeoPackage is built on SQLite)
- .fgb (FlatGeobuf) - Cloud-optimized format
- .csv, .tsv, .txt (for dataframe output only)

Other formats not listed have not been tested and will generate a warning.

layer — The name of the grid layer, passed directly to sf::st_write. Its interpretation depends on the destination driver. For a GeoPackage file, this will be the layer name. If dsn is a file path and layer is not specified, it defaults to the file's base name.

max_memory_gb — A numeric value. Maximum memory in gigabytes to use for grid creation. Default is NULL, in which case there is an automatic limit based on **available free system memory** (not total system RAM). Using this argument allows manual override, which is recommended on certain HPC (High Performance Computing) systems where jobs are allocated a fixed amount of memory that is less than the total free memory of the allocated node.

include_rat — Logical. If TRUE, generate a Raster Attribute Table (RAT) mapping numeric cell IDs to INSPIRE grid ID strings. Default is FALSE.

What is a RAT? A Raster Attribute Table stores metadata (like INSPIRE IDs) for each unique raster value. Without RAT, raster cells contain only numeric IDs (1, 2, 3...). With RAT, software like QGIS/R can display the IDs as human-readable labels.

Format-specific behavior:

- **GeoTIFF (.tif):** RAT stored in .tif.aux.xml sidecar file (XML). **Warning:** This sidecar can be **larger than the TIFF itself** for large grids. For chunked/streaming writes, requires a second pass (slower). Consider KEA or Erdas Imagine formats for large grids with labels.
- **KEA (.kea), Erdas Imagine (.img):** RAT embedded natively. **Recommended** for large grids requiring labels.
- **NetCDF (.nc), HDF5 (.hdf):** RAT **not supported**. An error is raised if include_rat = TRUE.

... — Additional arguments passed to backend handlers. When writing to text files (e.g., .csv, .tsv) via dsn, these arguments are passed to write_delim (e.g., na = "NA", quote = "all"). When writing to spatial files via dsn, these are passed

	to st_write. For output_type = "spatraster" writing, these are passed to writeRaster. For streaming backends (mirai or sequential), this can include max_cells_per_chunk to control memory usage.
grid_extent	The spatial object defining the extent. Can be an sf object, sfc geometry collection, bbox, numeric vector (as c(xmin, ymin, xmax, ymax)), or matrix.
ids	A character vector of INSPIRE-compliant grid cell IDs (e.g., "CRS3035RES100000mN26E43").

Details

This function creates a spatial grid aligned to the CRS origin, with support for clipping to input geometries, parallel processing, and multiple output formats.

Value

If dsn is NULL (the default), an sf object, data.frame, or SpatRaster representing the grid. If dsn is specified, the function writes the grid to a file and returns invisible(dsn).

An sf object or data.frame representing the grid derived from the INSPIRE IDs. If dsn is specified, returns invisible(dsn).

Examples

```
library(sf)
# Load the sample data from the sf package
nc_raw <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)

# Define target projected CRS and cell size
target_crs <- 5070 # NAD83 / Conus Albers
cellsize_m <- 10000 # 10 km

# Project the data
nc <- st_transform(nc_raw, target_crs)

# Create a grid covering the data
nc_grid <- inspire_grid_from_extent(
  grid_extent = nc,
  cellsize_m = cellsize_m,
  output_type = "sf_polygons",
  clip_to_input = TRUE
)

# Or using the S3 generic
nc_grid <- inspire_grid(nc, cellsize_m = cellsize_m, clip_to_input = TRUE)

head(nc_grid, 3)
library(sf)

inspire <- c(
  "CRS3035RES100000mN26E43", "CRS3035RES100000mN26E44",
  "CRS3035RES100000mN27E41", "CRS3035RES100000mN27E42",
  "CRS3035RES100000mN27E43", "CRS3035RES100000mN27E44"
)
```

```
grid <- inspire_grid_from_ids(inspire)
plot(grid$geometry)

# Or using the S3 generic
grid <- inspire_grid(inspire)
plot(grid$geometry)
```

`inspire_id_coords` *Generate INSPIRE IDs*

Description

Given pairs of coordinates, generates their INSPIRE grid representation. Given INSPIRE identifiers, can also extract the X and Y coordinates.

An INSPIRE ID contains information about the CRS, cell size and the ETRS89-LAEA coordinates of the lower-left corner of the grid cell in its format.

```
CRS3035{cellsize}mN{y}E{x} # long format
    {cellsize}N{y}E{x}         # short format (NE order)
    {cellsize}E{x}N{y}         # short format (EN order)
```

The long format always uses meters while the short format aggregates cell sizes greater or equal to 1000m to km.

Usage

```
inspire_id_from_coords(
  coords,
  cellsize_m = NULL,
  short = FALSE,
  axis_order = "NE",
  llc = FALSE,
  tolerance = 1e-06,
  sample = 2000
)

inspire_id_to_coords(inspire, as_sf = FALSE, crs = NULL)
```

Arguments

`coords`	A list, matrix, or dataframe where the X and Y coordinates are either in the columns `"x"` and `"y"` or in the first and second column position, respectively. Column names are converted to lowercase. Can also be a `sf`/`sfc` object in which case the coordinates are extracted using `st_coordinates`.

cellsize_m	A single integer representing the grid cell size in metres (e.g., 1000 for a 1 km grid). Required for spatial inputs.
short	If TRUE, generates short INSPIRE ID. Defaults to FALSE.
axis_order	A character string specifying the coordinate order for the output. This parameter is **only used when** short = TRUE. It can be one of: • "NE" (the default) to produce the format {cellsize}N{y}E{x}. • "EN" to produce the format {cellsize}E{x}N{y}.
llc	Do the coordinates in coords represent the lower-left corners of their cells? If FALSE, subtracts each coordinate by half of cellsize_m. If TRUE, leaves them as-is. Defaults to FALSE, i.e., treat coordinates as cell centroids.
tolerance	If res is NULL, controls the maximum acceptable difference between calculated cell spacings to consider them uniform. Defaults to 1e-6.
sample	If res is NULL, specifies the number of points to guess a resolution from. Defaults to 2000 to keep performance high. Increase this value if you are uncertain about the quality of your data.
inspire	A vector of INSPIRE IDs. Can be either legacy or non-legacy.
as_sf	Whether to return an object of class sfc or a dataframe.
crs	An optional numeric EPSG code or sf::st_crs object. If provided, this CRS will be assigned to all parsed coordinates, overriding any CRS information found in long-form IDs. If NULL (the default), the CRS is inferred from long-form IDs. When only short-form IDs are present, the function will default to EPSG:3035 (with a warning) for both sf and data.frame outputs.

Value

inspire_id_from_coords returns a character vector containing the INSPIRE identifiers.

inspire_id_to_coords returns a dataframe or sf dataframe (if as_sf = TRUE) containing the points extracted from the INSPIRE identifiers and information about the CRS and cell sizes. Note that the returned coordinates are always the centers of the grid cells as opposed to the lower-left corners.

Examples

```
# Generate IDs from a dataframe
coords <- data.frame(x = c(4334100, 4334200), y = 2684000)
gen <- inspire_id_from_coords(coords, llc = TRUE, cellsize_m = 100)
ext <- inspire_id_to_coords(gen)[c("x", "y")]
# Note: inspire_id_to_coords gives cell centers, so this won't be identical if llc=TRUE

# Generate long format IDs
inspire_id_from_coords(coords, llc = TRUE, cellsize_m = 100)

# Generate short format IDs with default "NE" axis order
inspire_id_from_coords(coords, llc = TRUE, cellsize_m = 1000, short = TRUE)

# Generate short format IDs with "EN" axis order
inspire_id_from_coords(coords, llc = TRUE, cellsize_m = 1000, short = TRUE, axis_order = "EN")
```

```
# Extract coordinates from short ID strings
inspire_id_to_coords("100mN34000E44000", crs = 3035)

# Generate IDs from an sf dataframe
if (requireNamespace("sf", quietly = TRUE)) {
  coords_df <- data.frame(x = c(4334100, 4334200), y = 2684000)
  coords_sf <- sf::st_as_sf(coords_df, coords = c("x", "y"), crs = 3035)
  inspire_id_from_coords(coords_sf, cellsize_m = 1000)
}
```

`inspire_id_format` *Convert INSPIRE IDs between long and short formats*

Description

This function converts a vector of INSPIRE-compliant grid IDs from their long format to the short format, or vice-versa. It automatically detects the input format and is fully vectorized to handle large inputs efficiently.

The long format is `CRS{epsg}RES{cellsize}mN{y}E{x}`. The short format can be either `{cellsize}N{y}E{x}` or `{cellsize}E{x}N{y}`.

Usage

```
inspire_id_format(ids, crs = 3035, axis_order = "NE")
```

Arguments

`ids`	A character vector of INSPIRE IDs. All IDs in the vector must be of the same format (either all long or all short).
`crs`	An integer representing the EPSG code. This parameter is **only used when converting from the short format to the long format**. It defaults to `3035` (ETRS89-LAEA).
`axis_order`	A character string specifying the coordinate order for the output. This parameter is **only used when converting from the long format to the short format**. It can be one of: • `"NE"` (the default) to produce the format `{cellsize}N{y}E{x}`. • `"EN"` to produce the format `{cellsize}E{x}N{y}`.

Value

A character vector of the converted INSPIRE IDs.

Examples

```
long_ids <- c("CRS3035RES1000mN2684000E4334000", "CRS3035RES10000mN2700000E4400000")
short_ids_ne <- c("1kmN2684E4334", "10kmN270E440")
short_ids_en <- c("1kmE4334N2684", "10kmE440N270")

# --- Long to Short ---

# Convert long to short with default "NE" order
inspire_id_format(long_ids)

# Convert long to short with specified "EN" order
inspire_id_format(long_ids, axis_order = "EN")

# --- Short to Long ---

# Convert short ("NE" format) to long with default CRS (3035)
inspire_id_format(short_ids_ne)

# The function also correctly parses the "EN" format
inspire_id_format(short_ids_en)

# Override the CRS when converting short to long
inspire_id_format(short_ids_ne, crs = 3857)
```

Index