

# Package ‘a5R’

May 7, 2026

**Title** 'A5' Discrete Global Grid System

**Version** 0.3.1

**Description** Bindings for the ``A5 geospatial index"  
<<https://a5geo.org/>>. 'A5' partitions the Earth's surface into pentagonal cells across 31 resolution levels using an equal-area projection onto a dodecahedron. Provides functions for indexing coordinates to cells, traversing the cell hierarchy, computing cell boundaries, and compacting/uncompacting cell sets. Powered by the 'A5' 'Rust' crate via 'extendr'.

**License** Apache License (>= 2)

**URL** <https://github.com/belian-earth/a5R>,  
<https://belian-earth.github.io/a5R/>

**BugReports** <https://github.com/belian-earth/a5R/issues>

**Depends** R (>= 4.2)

**Imports** cli, rlang (>= 1.1.0), units, vctrs (>= 0.6.0), wk (>= 0.9.0)

**Suggests** arrow, knitr, pillar, rmarkdown, sf, testthat (>= 3.0.0),  
tibble, withr

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Config/rextendr/version** 0.4.2

**SystemRequirements** Cargo (Rust's package manager), rustc

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**NeedsCompilation** yes

**Author** Hugh Graham [aut, cre],  
belian.earth [cph]

**Maintainer** Hugh Graham <[hugh@belian.earth](mailto:hugh@belian.earth)>

**Repository** CRAN

**Date/Publication** 2026-03-26 12:30:02 UTC

## Contents

a5_cell . . . . .	2
a5_cell_area . . . . .	3
a5_cell_distance . . . . .	4
a5_cell_from_arrow . . . . .	5
a5_cell_to_boundary . . . . .	6
a5_cell_to_children . . . . .	7
a5_cell_to_lonlat . . . . .	7
a5_cell_to_parent . . . . .	8
a5_compact . . . . .	9
a5_get_num_cells . . . . .	9
a5_get_num_children . . . . .	10
a5_get_res0_cells . . . . .	11
a5_get_resolution . . . . .	11
a5_grid . . . . .	12
a5_grid_disk . . . . .	13
a5_lonlat_to_cell . . . . .	14
a5_set_threads . . . . .	14
a5_spherical_cap . . . . .	15
a5_u64_to_hex . . . . .	15
a5_uncompact . . . . .	16
wk_methods . . . . .	17

<b>Index</b>	<b>18</b>
--------------	-----------

---

a5_cell	<i>A5 Cell Index Vector</i>
---------	-----------------------------

---

### Description

Create, test, and coerce A5 cell index vectors. Cells are stored as a record with eight raw-byte fields (b1–b8) representing the little-endian bytes of the u64 cell ID. This avoids the precision loss of floating-point storage and keeps memory compact.

### Usage

```
a5_cell(x = character())
```

```
is_a5_cell(x)
```

```
as_a5_cell(x)
```

```
a5_is_valid(x)
```

### Arguments

x	A character vector of hex-encoded A5 cell IDs, or an object coercible to one.
---	---

**Value**

An a5\_cell vector (a5\_cell, as\_a5\_cell), a logical scalar (is\_a5\_cell), or a logical vector (a5\_is\_valid).

**Examples**

```
cells <- a5_cell(c("0800000000000006", "0800000000000016"))
cells
a5_is_valid(c("0800000000000006", "not_a_cell", NA))
```

---

a5_cell_area	<i>Cell area at a given resolution</i>
--------------	--

---

**Description**

Returns the area of a single cell in square metres at the given resolution(s). Because A5 is an equal-area DGGS, all cells at the same resolution have identical area.

**Usage**

```
a5_cell_area(resolution, units = "m^2")
```

**Arguments**

resolution	Integer vector of resolutions (0–30).
units	Character scalar specifying the output area unit (default "m^2"). Any unit convertible from m^2 via <code>units::set_units()</code> is accepted (e.g. "km^2", "ha", "acre"). If NULL, the area is returned as a numeric vector in m^2.

**Value**

A `units::units` vector of areas.

**Examples**

```
a5_cell_area(0:5)
a5_cell_area(5, units = "km^2")
```

---

a5_cell_distance	<i>Distance between cell centroids</i>
------------------	--

---

### Description

Computes the distance between the centroids of pairs of A5 cells using the specified method.

### Usage

```
a5_cell_distance(  
  from,  
  to,  
  units = "m",  
  method = c("haversine", "geodesic", "rhumb")  
)
```

### Arguments

from, to	<a href="#">a5_cell</a> vectors (recycled to common length).
units	Character scalar specifying the distance unit (default "m"). Any unit convertible from metres via <a href="#">units::set_units()</a> is accepted (e.g. "km", "mi"). If NULL, the distance is returned as a numeric vector in metres.
method	Distance calculation method. One of "haversine" (great-circle, default), "geodesic" (WGS84 ellipsoid via Karney 2013), or "rhumb" (loxodrome / constant-bearing).

### Value

A [units::units](#) vector of distances.

### See Also

[a5\\_cell\\_to\\_lonlat\(\)](#) for cell centroids, [a5\\_cell\\_area\(\)](#) for cell areas.

### Examples

```
a <- a5_lonlat_to_cell(-3.19, 55.95, resolution = 24)  
b <- a5_lonlat_to_cell(-3.10, 55.90, resolution = 24)  
a5_cell_distance(a, b)  
a5_cell_distance(a, b, units = "km")  
a5_cell_distance(a, b, method = "geodesic")
```

---

a5\_cell\_from\_arrow      *Convert between a5\_cell and Arrow uint64 arrays*

---

### Description

Losslessly convert between [a5\\_cell](#) vectors and Arrow uint64 arrays. This avoids the precision loss that occurs when Arrow converts uint64 to R's double (which can only represent integers exactly up to  $2^{53}$ , while A5 cell IDs span the full  $0-2^{64}$  range).

### Usage

```
a5_cell_from_arrow(x)
```

```
a5_cell_to_arrow(x)
```

### Arguments

x                      For a5\_cell\_from\_arrow(), an Arrow Array or ChunkedArray of type uint64.  
For a5\_cell\_to\_arrow(), an [a5\\_cell](#) vector.

### Details

Internally these use Arrow's zero-copy View() to reinterpret uint64 bytes as fixed\_size\_binary(8), then convert to/from the raw-byte representation used by [a5\\_cell](#). The resulting Arrow arrays can be written directly to Parquet and read correctly by DuckDB, Python, and other Arrow-compatible tools.

### Value

a5\_cell\_from\_arrow() returns an [a5\\_cell](#) vector. a5\_cell\_to\_arrow() returns an Arrow Array of type uint64.

### See Also

[a5\\_u64\\_to\\_hex\(\)](#) for converting to hex strings instead.

### Examples

```
cell <- a5_lonlat_to_cell(135, 0, resolution = 10)
arr <- a5_cell_to_arrow(cell)
back <- a5_cell_from_arrow(arr)
identical(format(cell), format(back))
```

```
cells <- a5_lonlat_to_cell(c(-3.19, 135), c(55.95, 0), resolution = 10)
arr <- a5_cell_to_arrow(cells)
arr$type$ToString()
```

a5\_cell\_to\_boundary    *Get cell boundary polygons*

---

### Description

Returns the boundary of each cell as a `wk::wkt()` or `wk::wkb()` polygon geometry. Boundaries are pentagonal polygons on the WGS 84 ellipsoid.

### Usage

```
a5_cell_to_boundary(  
  cell,  
  format = c("wkb", "wkt"),  
  closed = TRUE,  
  segments = NULL  
)
```

### Arguments

<code>cell</code>	An <a href="#">a5_cell</a> vector.
<code>format</code>	Character scalar, either "wkb" (default) or "wkt".
<code>closed</code>	Logical scalar; if TRUE (default) the ring is closed (first vertex repeated at end).
<code>segments</code>	Integer scalar or NULL. Number of interpolation segments per edge for geodesic accuracy. NULL uses the default (straight edges).

### Value

A `wk_wkt` or `wk_wkb` vector of polygon geometries with `wk::wk_crs_longlat()` CRS.

### See Also

[a5\\_cell\\_to\\_lonlat\(\)](#) for cell centroids.

### Examples

```
cell <- a5_lonlat_to_cell(-3.19, 55.95, resolution = 5)  
a5_cell_to_boundary(cell)  
a5_cell_to_boundary(cell, format = "wkt")
```

---

a5\_cell\_to\_children    *Get child cells*

---

### Description

Returns the child cells of a single cell. By default returns the 4 immediate children (one resolution finer). Optionally target a specific finer resolution.

### Usage

```
a5_cell_to_children(cell, resolution = NULL)
```

### Arguments

cell                    A single [a5\\_cell](#) value.  
 resolution            Integer scalar target child resolution, or NULL for immediate children.

### Value

An [a5\\_cell](#) vector of child cells.

### See Also

[a5\\_cell\\_to\\_parent\(\)](#), [a5\\_get\\_resolution\(\)](#)

### Examples

```
cell <- a5_lonlat_to_cell(-3.19, 55.95, resolution = 5)
a5_cell_to_children(cell)
```

---

a5\_cell\_to\_lonlat      *Convert A5 cell indices to coordinates*

---

### Description

Returns the centre-point longitude and latitude of each cell.

### Usage

```
a5_cell_to_lonlat(cell, normalise = TRUE)
```

### Arguments

cell                    An [a5\\_cell](#) vector (or character coercible to one).  
 normalise            Logical scalar. If TRUE (default), longitudes are wrapped to  $[-180, 180]$  and returned as a [wk::xy\(\)](#) vector. If FALSE, raw unwrapped coordinates are returned as a data frame (lon, lat) — useful for calculations spanning the antimeridian.

**Value**

A `wk::xy()` vector (if `normalise = TRUE`) or a data frame with columns `lon` and `lat`.

**See Also**

[a5\\_lonlat\\_to\\_cell\(\)](#) for the inverse operation, [a5\\_cell\\_to\\_boundary\(\)](#) for full cell polygons.

**Examples**

```
cell <- a5_lonlat_to_cell(-3.19, 55.95, resolution = 5)
a5_cell_to_lonlat(cell)

# Raw unwrapped coordinates
cell2 <- a5_lonlat_to_cell(114.8, 4.1, resolution = 5)
a5_cell_to_lonlat(cell2, normalise = FALSE)
```

---

a5_cell_to_parent	<i>Navigate to parent cell(s)</i>
-------------------	-----------------------------------

---

**Description**

Returns the parent cell of each input cell. By default returns the immediate parent (one resolution coarser). Optionally target a specific coarser resolution.

**Usage**

```
a5_cell_to_parent(cell, resolution = NULL)
```

**Arguments**

`cell` An [a5\\_cell](#) vector.  
`resolution` Integer scalar target parent resolution, or `NULL` for the immediate parent.

**Value**

An [a5\\_cell](#) vector of parent cells.

**See Also**

[a5\\_cell\\_to\\_children\(\)](#), [a5\\_get\\_resolution\(\)](#)

**Examples**

```
cell <- a5_lonlat_to_cell(-3.19, 55.95, resolution = 10)
a5_cell_to_parent(cell)
a5_cell_to_parent(cell, resolution = 5)
```

---

a5_compact	<i>Compact a set of A5 cells</i>
------------	----------------------------------

---

**Description**

Merges complete sibling groups into their common parent, reducing the number of cells while preserving coverage.

**Usage**

```
a5_compact(cells)
```

**Arguments**

cells            An [a5\\_cell](#) vector.

**Value**

An [a5\\_cell](#) vector of compacted cells.

**See Also**

[a5\\_uncompact\(\)](#)

**Examples**

```
cell <- a5_lonlat_to_cell(-3.19, 55.95, resolution = 5)
children <- a5_cell_to_children(cell)
a5_compact(children) # back to the parent
```

---

a5_get_num_cells	<i>Total number of cells at a given resolution</i>
------------------	--

---

**Description**

Total number of cells at a given resolution

**Usage**

```
a5_get_num_cells(resolution)
```

**Arguments**

resolution       Integer scalar resolution (0–30).

**Value**

A numeric scalar (double) giving the total count. Returned as double because the count can exceed R's integer range.

**Examples**

```
a5_get_num_cells(0)
a5_get_num_cells(10)
```

---

a5_get_num_children	<i>Number of children between two resolutions</i>
---------------------	---

---

**Description**

Returns the number of child cells each parent cell contains when expanding from one resolution to another.

**Usage**

```
a5_get_num_children(parent_resolution, child_resolution)
```

**Arguments**

```
parent_resolution
    Integer scalar (0–30).
child_resolution
    Integer scalar (0–30), must be >= parent_resolution.
```

**Value**

A numeric scalar. Returned as double because the count can exceed R's integer range at large resolution deltas.

**See Also**

[a5\\_get\\_num\\_cells\(\)](#), [a5\\_cell\\_to\\_children\(\)](#), [a5\\_uncompact\(\)](#)

**Examples**

```
a5_get_num_children(5, 8) # 4^3 = 64
a5_get_num_children(0, 5)
```

---

a5\_get\_res0\_cells      *Get all resolution-0 root cells*

---

**Description**

Returns the 12 root cells corresponding to the 12 faces of the dodecahedron.

**Usage**

```
a5_get_res0_cells()
```

**Value**

An [a5\\_cell](#) vector of length 12.

**Examples**

```
a5_get_res0_cells()
```

---

a5\_get\_resolution      *Get the resolution of A5 cell indices*

---

**Description**

Extracts the resolution level (0–30) encoded in each cell index.

**Usage**

```
a5_get_resolution(cell)
```

**Arguments**

cell      An [a5\\_cell](#) vector.

**Value**

An integer vector of resolutions.

**See Also**

[a5\\_cell\\_to\\_parent\(\)](#), [a5\\_cell\\_to\\_children\(\)](#)

**Examples**

```
cell <- a5_lonlat_to_cell(-3.19, 55.95, resolution = 10)
a5_get_resolution(cell)
```

a5\_grid

*Generate a grid of A5 cells covering an area***Description**

Returns all cells at the target resolution that intersect the given geometry. Uses hierarchical flood-fill: starting from the 12 resolution-0 root cells, the algorithm repeatedly expands and prunes by spatial intersection until the target resolution is reached.

**Usage**

```
a5_grid(x, resolution)
```

**Arguments**

x	An area specification. One of: <ul style="list-style-type: none"> <li>• A numeric vector of length 4 (<code>c(xmin, ymin, xmax, ymax)</code>) interpreted as a WGS 84 bounding box.</li> <li>• Any geometry handleable by <code>wk::wk_handle()</code> (e.g. <code>wk::wkt()</code>, <code>wk::wkb()</code>, <code>sfc</code>, <code>sf</code>, <code>a5_cell</code>).</li> </ul>
resolution	Integer scalar target resolution (0–30).

**Details**

Grid generation runs entirely in Rust via hierarchical flood-fill with bounding-box pruning. For non-bbox geometry inputs, an exact intersection filter removes cells that fall outside the target shape. No cell count limit is imposed — high resolutions over large areas can consume significant memory.

Input geometries must use WGS 84 coordinates; projected geometries are not reprojected and will produce incorrect results. Multiple geometries are collected into a `GEOMETRYCOLLECTION` automatically. Antimeridian-crossing bounding boxes are supported: when `xmin > xmax` (e.g. `c(170, -50, -170, -30)`), the bbox is split at the antimeridian.

**Limitation:** spatial filtering uses planar geometry on lon/lat coordinates, which can produce incomplete results very close to the poles (above  $\sim 88^\circ$  latitude) or the antimeridian. Use a larger target geometry to ensure complete coverage in these areas.

**Value**

An `a5_cell` vector of cells at resolution that intersect x.

**See Also**

`a5_cell_to_boundary()` to convert result cells to geometries.

## Examples

```
# Grid from a bounding box
cells <- a5_grid(c(-3.3, 55.9, -3.1, 56.0), resolution = 5)
cells

# Grid from a WKT polygon
poly <- wk::wkt("POLYGON ((-3.3 55.9, -3.1 55.9, -3.1 56, -3.3 56, -3.3 55.9))")
cells <- a5_grid(poly, resolution = 5)
```

---

a5_grid_disk	<i>Cells within k hops of a cell</i>
--------------	--------------------------------------

---

## Description

Returns all cells reachable within k edge hops of a centre cell, including the centre cell itself.

## Usage

```
a5_grid_disk(cell, k, vertex = FALSE)
```

## Arguments

cell	A single <a href="#">a5_cell</a> value.
k	Integer scalar, number of hops.
vertex	Logical scalar. If FALSE (default), only edge-sharing neighbours (4-connected) are traversed. If TRUE, vertex-sharing neighbours are included (8-connected).

## Value

A compacted [a5\\_cell](#) vector.

## See Also

[a5\\_spherical\\_cap\(\)](#) for distance-based selection.

## Examples

```
cell <- a5_lonlat_to_cell(-3.19, 55.95, resolution = 8)
a5_grid_disk(cell, k = 1)
```

---

a5\_lonlat\_to\_cell      *Convert coordinates to A5 cell indices*

---

### Description

Maps longitude/latitude coordinates to A5 cell indices at the specified resolution.

### Usage

```
a5_lonlat_to_cell(lon, lat, resolution)
```

### Arguments

lon                    Numeric vector of longitudes in degrees.  
 lat                    Numeric vector of latitudes in degrees.  
 resolution            Integer scalar or vector of resolutions (0–30).

### Value

An [a5\\_cell](#) vector of cell indices.

### See Also

[a5\\_cell\\_to\\_lonlat\(\)](#) for the inverse operation.

### Examples

```
a5_lonlat_to_cell(-3.19, 55.95, resolution = 5)
```

---

a5\_set\_threads      *Set the number of threads used by a5R*

---

### Description

Controls the number of threads used for parallel processing in vectorised functions. Set to 1 (the default) for sequential processing with zero overhead, or higher for parallel execution via rayon.

### Usage

```
a5_set_threads(n = 1L)
```

```
a5_get_threads()
```

### Arguments

n                      Integer scalar. Number of threads. Must be  $\geq 1$ .

**Value**

Invisibly returns the previous thread count.  
Integer scalar.

---

a5_spherical_cap	<i>Cells within a great-circle radius</i>
------------------	---

---

**Description**

Returns all cells whose centres fall within a great-circle distance of a given cell's centre.

**Usage**

```
a5_spherical_cap(cell, radius)
```

**Arguments**

cell	A single <a href="#">a5_cell</a> value.
radius	Numeric scalar, great-circle radius in metres.

**Value**

A compacted [a5\\_cell](#) vector.

**See Also**

[a5\\_grid\\_disk\(\)](#) for hop-based selection.

**Examples**

```
cell <- a5_lonlat_to_cell(-3.19, 55.95, resolution = 8)
a5_spherical_cap(cell, radius = 1000)
```

---

a5_u64_to_hex	<i>Coerce between hex strings and A5 cell vectors</i>
---------------	---

---

**Description**

[a5\\_u64\\_to\\_hex\(\)](#) converts an [a5\\_cell](#) vector to 16-character zero-padded hex strings. [a5\\_hex\\_to\\_u64\(\)](#) converts hex strings to an [a5\\_cell](#) vector.

**Usage**

```
a5_u64_to_hex(x)
```

```
a5_hex_to_u64(x)
```

**Arguments**

x For `a5_u64_to_hex()`, an [a5\\_cell](#) vector (or object coercible to one). For `a5_hex_to_u64()`, a character vector of hex-encoded cell IDs.

**Details**

These are named to match `u64_to_hex / hex_to_u64` in the upstream Python, JavaScript, and DuckDB A5 bindings. In those languages the functions convert between a native 64-bit unsigned integer and its hex representation. Because R has no native `uint64` type, `a5_u64_to_hex()` accepts an [a5\\_cell](#) (which stores the u64 internally as eight raw bytes) instead of a bare integer.

**Value**

`a5_u64_to_hex()` returns a character vector. `a5_hex_to_u64()` returns an [a5\\_cell](#) vector.

**See Also**

[a5\\_cell\\_from\\_arrow\(\)](#) and [a5\\_cell\\_to\\_arrow\(\)](#) for lossless conversion between [a5\\_cell](#) and Arrow `uint64` arrays.

**Examples**

```
cell <- a5_lonlat_to_cell(-3.19, 55.95, resolution = 5)
hex <- a5_u64_to_hex(cell)
hex
a5_hex_to_u64(hex)
```

---

a5\_uncompact

*Uncompact a set of A5 cells to a target resolution*


---

**Description**

Expands each cell to its descendants at the target resolution.

**Usage**

```
a5_uncompact(cells, resolution)
```

**Arguments**

cells An [a5\\_cell](#) vector.  
resolution Integer scalar target resolution (0–30).

**Value**

An [a5\\_cell](#) vector of uncompact cells.

**See Also**

[a5\\_compact\(\)](#)

**Examples**

```
cell <- a5_lonlat_to_cell(-3.19, 55.95, resolution = 5)
a5_uncompact(cell, resolution = 7)
```

---

wk\_methods

*wk methods for a5\_cell*

---

**Description**

Integration with the [wk](#) geometry framework. Allows `a5_cell` vectors to be handled as geometry (via their boundary polygons) and to report their CRS.

**Usage**

```
## S3 method for class 'a5_cell'
wk_handle(handleable, handler, ...)
```

```
## S3 method for class 'a5_cell'
wk_crs(x)
```

**Arguments**

<code>handleable, x</code>	An <a href="#">a5_cell</a> vector.
<code>handler</code>	A <a href="#">wk handler</a> .
<code>...</code>	Passed to underlying methods.

**Value**

- `wk_handle()`: the result of the handler.
- `wk_crs()`: a [wk::wk\\_crs](#) object (WGS 84 lon/lat).

# Index

a5\_cell, [2](#), [4–9](#), [11–17](#)  
a5\_cell\_area, [3](#)  
a5\_cell\_area(), [4](#)  
a5\_cell\_distance, [4](#)  
a5\_cell\_from\_arrow, [5](#)  
a5\_cell\_from\_arrow(), [16](#)  
a5\_cell\_to\_arrow (a5\_cell\_from\_arrow), [5](#)  
a5\_cell\_to\_arrow(), [16](#)  
a5\_cell\_to\_boundary, [6](#)  
a5\_cell\_to\_boundary(), [8](#), [12](#)  
a5\_cell\_to\_children, [7](#)  
a5\_cell\_to\_children(), [8](#), [10](#), [11](#)  
a5\_cell\_to\_lonlat, [7](#)  
a5\_cell\_to\_lonlat(), [4](#), [6](#), [14](#)  
a5\_cell\_to\_parent, [8](#)  
a5\_cell\_to\_parent(), [7](#), [11](#)  
a5\_compact, [9](#)  
a5\_compact(), [17](#)  
a5\_get\_num\_cells, [9](#)  
a5\_get\_num\_cells(), [10](#)  
a5\_get\_num\_children, [10](#)  
a5\_get\_res0\_cells, [11](#)  
a5\_get\_resolution, [11](#)  
a5\_get\_resolution(), [7](#), [8](#)  
a5\_get\_threads (a5\_set\_threads), [14](#)  
a5\_grid, [12](#)  
a5\_grid\_disk, [13](#)  
a5\_grid\_disk(), [15](#)  
a5\_hex\_to\_u64 (a5\_u64\_to\_hex), [15](#)  
a5\_is\_valid (a5\_cell), [2](#)  
a5\_lonlat\_to\_cell, [14](#)  
a5\_lonlat\_to\_cell(), [8](#)  
a5\_set\_threads, [14](#)  
a5\_spherical\_cap, [15](#)  
a5\_spherical\_cap(), [13](#)  
a5\_u64\_to\_hex, [15](#)  
a5\_u64\_to\_hex(), [5](#)  
a5\_uncompact, [16](#)  
a5\_uncompact(), [9](#), [10](#)

as\_a5\_cell (a5\_cell), [2](#)

is\_a5\_cell (a5\_cell), [2](#)

units::set\_units(), [3](#), [4](#)  
units::units, [3](#), [4](#)

wk, [17](#)  
wk handler, [17](#)  
wk::wk\_crs, [17](#)  
wk::wk\_handle(), [12](#)  
wk::wkb(), [6](#), [12](#)  
wk::wkt(), [6](#), [12](#)  
wk::xy(), [7](#), [8](#)  
wk\_crs.a5\_cell (wk\_methods), [17](#)  
wk\_handle.a5\_cell (wk\_methods), [17](#)  
wk\_methods, [17](#)