

Package: urbioconnect (via r-universe)

October 12, 2026

Title Urban Habitat Connectivity Analysis

Version 0.1.0.9000

Description Analyse and visualise habitat connectivity in urban landscapes, accounting for barriers and buffer distances. Includes Shiny app for interactive analysis and report generation. Based on methods developed by Kirk et al (2023) <doi:10.1016/j.mex.2022.101989>.

License GPL (>= 3)

URL <https://urbio-ecology.r-universe.dev/urbioconnect>,
<https://github.com/urbio-ecology/urbioconnect>,
<https://urbio-ecology.github.io/urbioconnect/>

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 4.4.0)

Imports colorspace, dplyr, ggplot2, glue, knitr, purrr, quarto, readr, scico, sf, shiny, stringr, terra (>= 1.8-70), tibble, tidyr, tidyterra, tools, utils, zip, stats, grDevices, cli, rlang, scales, vctrs

Suggests bslib, chromote, conflicted, crew, diffviewer, DT, fasterize, gridExtra, igraph, marquee, magick, prettyunits, raster, rmarkdown, shinyjs, shinytest2, stars, tarchetypes, targets, testthat (>= 3.0.0), vdiff, withr, tictoc, ggspatial

SystemRequirements Quarto (>= 1.4) for the reports, which render PDF through Typst; <https://quarto.org/docs/get-started/>

Config/testthat/edition 3

Config/testthat/parallel true

VignetteBuilder knitr

BugReports <https://github.com/urbio-ecology/urbioconnect/issues>

LazyData true

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libicu-dev libuv1-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libx11-dev zlib1g-dev

Repository <https://urbio-ecology.r-universe.dev>

Date/Publication 2026-10-11 21:53:46 UTC

RemoteUrl <https://github.com/urbio-ecology/urbioconnect>

RemoteRef HEAD

RemoteSha f95672b88350618c5920ad6123e342f96405f921

Contents

add_patch_area	3
aggregate_connected_patches	4
assign_patches_to_fragments	5
clean	6
col2hex	6
compare_connectivity	7
compare_scenarios	8
connectivity_display	10
connectivity_file_stem	11
connectivity_probability	12
connectivity_report_data	13
create_barrier_mask	15
display_ids	15
drop_habitat_under_barrier	16
effective_mesh_size	17
empty_grid	17
example-lizard-data	18
example-wren-data	19
format_resolution	20
fragment_habitat	21
generate_connectivity_report	22
gg_barrier_habitat_interpatch_dist	23
gg_layer	25
habitat_buffer	26
habitat_connectivity	27
habitat_connectivity_comparison	28
habitat_connectivity_full	30
habitat_connectivity_scenarios	31
lizard_areas_connected	33
mean_patch_size	34
n_patches	34
new_patch_size_tbl	35
onto_grid	36
patch_sizes	37
pc-getters	37
plot_barrier_habitat_interpatch_dist	38

plot_connectivity	39
plot_patches	40
prepare_rasters	41
quarto_available	42
read_geometry	43
render_connectivity_report	43
round_by	44
run_connectivity_app	45
scenario_layer	46
sf_add_patch_area	47
sf_aggregate_connected_patches	47
sf_assign_patches_to_fragments	48
sf_drop_habitat_under_barrier	49
sf_fragment_habitat	50
sf_habitat_buffer	50
sf_habitat_connectivity	51
summarise-connectivity	52
total_habitat_area	53
urbio_colours	54
urbio_figure_size	55
write_connectivity_assets	55
write_connectivity_report	56
write_report_data	57
zip_connectivity_assets	58
zip_connectivity_reports	59

Index 61

add_patch_area	<i>Add patch area layer</i>
----------------	-----------------------------

Description

Adds an area layer to the raster of the area of each patch.

Usage

```
add_patch_area(raster)
```

Arguments

raster terra SpatRaster. In the workflow, this is the patch ID raster.

Value

terra SpatRaster with two layers: patch_id and area.

Examples

```
lizard_habitat <- example_habitat()
lizard_barrier <- example_barrier()
buffered_habitat <- habitat_buffer(lizard_habitat, 5)
barrier_mask <- create_barrier_mask(lizard_barrier)
fragmented <- fragment_habitat(buffered_habitat, barrier_mask)
remaining_habitat <- drop_habitat_under_barrier(
  habitat = lizard_habitat,
  barrier = lizard_barrier
)
fragment_patches <- assign_patches_to_fragments(
  remaining_habitat = remaining_habitat,
  fragment = fragmented
)
library(terra)
add_patch_area(fragment_patches)
```

aggregate_connected_patches

Aggregate connected patch areas

Description

Aggregate a raster with connected patch areas into a data frame where each row is a unique patch, and its area and area squared.

Usage

```
aggregate_connected_patches(raster)
```

Arguments

raster terra SpatRaster. Raster with patch_id and area layers.

Value

Data frame with patch areas and areas squared.

Examples

```
lizard_habitat <- example_habitat()
lizard_barrier <- example_barrier()
buffered_habitat <- habitat_buffer(lizard_habitat, 5)
barrier_mask <- create_barrier_mask(lizard_barrier)
fragmented <- fragment_habitat(buffered_habitat, barrier_mask)
remaining_habitat <- drop_habitat_under_barrier(
  habitat = lizard_habitat,
  barrier = lizard_barrier
)
```

```
fragment_patches <- assign_patches_to_fragments(  
  remaining_habitat = remaining_habitat,  
  fragment = fragmented  
)  
library(terra)  
patch_areas <- add_patch_area(fragment_patches)  
aggregate_connected_patches(patch_areas)
```

assign_patches_to_fragments
Assign patches to fragments

Description

Assign patches to fragments

Usage

```
assign_patches_to_fragments(remaining_habitat, fragment)
```

Arguments

remaining_habitat	Terra SpatRaster. Remaining habitat.
fragment	Terra SpatRaster. Fragment geometry.

Value

Terra SpatRaster with patch IDs.

Examples

```
lizard_habitat <- example_habitat()  
lizard_barrier <- example_barrier()  
buffered_habitat <- habitat_buffer(lizard_habitat, 5)  
barrier_mask <- create_barrier_mask(lizard_barrier)  
fragmented <- fragment_habitat(buffered_habitat, barrier_mask)  
remaining_habitat <- drop_habitat_under_barrier(  
  habitat = lizard_habitat,  
  barrier = lizard_barrier  
)  
fragment_patches <- assign_patches_to_fragments(  
  remaining_habitat = remaining_habitat,  
  fragment = fragmented  
)  
library(terra)  
plot(fragment_patches)
```

clean	<i>Clean any spatial data layer (shape file)</i>
-------	--

Description

Many shape files contain errors, places where the edges of a polygon cross over or polygons which overlap. This helps remove some of those errors by smoothing the edges of polygons, removing corners, and dissolving edges where polygons overlap. This reduces the complexity of the shape file, making future steps faster.

Usage

```
clean(spatial_data, ...)
```

Arguments

spatial_data	spatial data frame from sf.
...	extra options (currently not used)

Value

An sf object with simplified and validated geometry.

Examples

```
lizard_barrier_shp <- example_barrier_shp()
clean(lizard_barrier_shp)
```

col2hex	<i>Convert color name to hexadecimal</i>
---------	--

Description

Convert color name to hexadecimal

Usage

```
col2hex(color_name)
```

Arguments

color_name	Character. Color name recognized by R.
------------	--

Value

Character. Hexadecimal color code.

Examples

```
col2hex("forestgreen")
col2hex("blue")
```

compare_connectivity *Compare the connectivity of two scenarios*

Description

We can measure the connectivity of a given habitat and barrier with [habitat_connectivity\(\)](#). We can also compare the connectivity, say for example if you have the same area habitat and barrier, but you want to understand what the change in connectedness is when you remove, or add some habitat, or some barrier(s). This function helps you do that, by comparing a "scenario" connectivity object against a "baseline" connectivity object (both created by [habitat_connectivity\(\)](#) or [summarise_connectivity\(\)](#)).

Usage

```
compare_connectivity(scenario, baseline, scenario_name = NULL)
```

Arguments

scenario	A connectivity object (from habitat_connectivity() or summarise_connectivity()). The scenario to compare against baseline. Must be a single row.
baseline	A connectivity object (from habitat_connectivity() or summarise_connectivity()). The reference the scenario is compared against. Must be a single row, and match scenario on species, interpatch_distance, and resolution.
scenario_name	Character. An optional label for the scenario, for example "Bentley Project". Appears in the scenario_name column on every row. Defaults to NULL, which gives NA.

Details

If you have raster or vector layers rather than connectivity objects, use [habitat_connectivity_comparison\(\)](#), which runs [habitat_connectivity\(\)](#) on each scenario for you and then calls this function.

Value

A compare_connectivity object: a tibble with four rows (baseline, scenario, change, pct_change, in the measure column) and the same metric columns as [summarise_connectivity\(\)](#) output: scenario_name, measure, species, interpatch_distance, n_patches, effective_mesh_ha, prob_connectedness, patch_area_mean, patch_area_total_ha, and data_resolution. The change row is scenario - baseline, so a positive value means the scenario is higher than the baseline; the pct_change row expresses that same change as $100 * \text{change} / \text{baseline}$, which is the readable form for metrics whose absolute deltas are very small. Metric values are held at full precision - they are not rounded - so change is exact.

See Also

[habitat_connectivity_comparison\(\)](#), which starts from habitat and barrier layers instead of connectivity objects, [habitat_connectivity\(\)](#), and [summarise_connectivity\(\)](#).

Examples

```
# build `connectivity` objects cheaply (no spatial pipeline) and compare them
baseline <- summarise_connectivity(lizard_areas_connected)
# a scenario in which one connected patch (the first row) is lost
scenario <- summarise_connectivity(lizard_areas_connected[-1, ])
compare_connectivity(scenario = scenario, baseline = baseline)

# label the scenario so it can be told apart from others
compare_connectivity(
  scenario = scenario,
  baseline = baseline,
  scenario_name = "Bentley Project"
)
```

compare_scenarios	<i>Compare several scenarios against one baseline</i>
-------------------	---

Description

Where [compare_connectivity\(\)](#) compares a single scenario against a baseline, this compares as many as you like, and labels each one. Give it a named list of connectivity objects: the names become the scenario_name column, so a scenario can be identified in the result.

Usage

```
compare_scenarios(baseline, scenarios)
```

Arguments

baseline	A connectivity object (from habitat_connectivity() or summarise_connectivity()) that every scenario is compared against. Must be a single row.
scenarios	A named list of connectivity objects, one per scenario. Names must be present and unique, and each scenario must match baseline on species, interpatch_distance, and resolution.

Details

If you have raster or vector layers rather than connectivity objects, run [habitat_connectivity\(\)](#) on each one first, as in the second example below.

Value

A `compare_connectivity` object with four rows per scenario (baseline, scenario, change, pct_change), labelled in the `scenario_name` column. See [compare_connectivity\(\)](#) for the columns.

See Also

[compare_connectivity\(\)](#) for a single scenario, and [habitat_connectivity_comparison\(\)](#), which starts from habitat and barrier layers instead of connectivity objects.

Examples

```
baseline <- summarise_connectivity(lizard_areas_connected)

compare_scenarios(
  baseline = baseline,
  scenarios = list(
    "Bentley Project" = summarise_connectivity(lizard_areas_connected[-1, ]),
    "Barrier upgrade" = summarise_connectivity(
      lizard_areas_connected[-(1:5), ]
    )
  )
)

# starting from layers: build a `connectivity` object for each first
wren_habitat <- example_wren_habitat()
wren_barrier <- example_wren_barrier()

wren_baseline <- habitat_connectivity(
  habitat = wren_habitat,
  barrier = wren_barrier,
  species = "Superb Fairy Wren",
  interpatch_distance = 200,
  verbose = FALSE
)

bentley_project <- habitat_connectivity(
  habitat = example_wren_habitat_scenario(),
  barrier = wren_barrier,
  species = "Superb Fairy Wren",
  interpatch_distance = 200,
  verbose = FALSE
)

compare_scenarios(
  baseline = wren_baseline,
  scenarios = list("Bentley Project" = bentley_project)
)
```

connectivity_display *Prepare a result for display*

Description

Labels the columns and says how precisely each should be shown, so a table on screen and the same table in a report agree. Values come back **numeric**: the rounding is described rather than applied, because `DT::datatable()` needs numbers to stay numbers to sort them, while `knitr::kable()` wants them already rounded. `round_by()` applies the rounding when that is what you want.

Usage

```
connectivity_display(x, ...)

## S3 method for class 'connectivity'
connectivity_display(x, ...)

## S3 method for class 'patch_size_tbl'
connectivity_display(x, ...)

## S3 method for class 'compare_connectivity'
connectivity_display(x, wide = TRUE, ...)

## Default S3 method:
connectivity_display(x, ...)
```

Arguments

<code>x</code>	A connectivity, patch_size_tbl or compare_connectivity object.
<code>...</code>	Passed to methods.
<code>wide</code>	For a comparison, one row per metric and a column per measure, which is the shape a reader wants. FALSE keeps the object's own shape: one row per measure, a column per metric.

Value

A list of

- data, a tibble with display labels and numeric values
- digits, a tibble of column, kind ("round" or "signif") and digits, one row per column that needs rounding

See Also

`round_by()` to apply the rounding.

Examples

```
lizard <- habitat_connectivity(  
  habitat = example_habitat(),  
  barrier = example_barrier(),  
  species = "Blue Tongue Lizard",  
  interpatch_distance = 20,  
  verbose = FALSE  
)  
  
display <- connectivity_display(lizard)  
display$digits  
round_by(display)
```

connectivity_file_stem

The name to give a download: species and date

Description

The stem every download takes its name from, so a report, a zip and the folder inside that zip agree. The species is lower-cased with its spaces turned into dashes, which keeps it usable as a file name.

Usage

```
connectivity_file_stem(x)
```

Arguments

x A connectivity_report_data object from [connectivity_report_data\(\)](#).

Value

A string, with no extension.

See Also

[generate_connectivity_report\(\)](#) and [write_connectivity_report\(\)](#), whose path defaults to this, and [zip_connectivity_assets\(\)](#), which names the folder inside the archive with it.

Examples

```
report_data <- connectivity_report_data(  
  habitat = example_habitat(),  
  barrier = example_barrier(),  
  species = "Blue Tongue Lizard",  
  interpatch_distance = 20,  
  verbose = FALSE  
)
```

```
connectivity_file_stem(report_data)
```

```
connectivity_probability
```

Calculate connectivity probability

Description

Computes the probability two randomly chosen points within habitat are connected, accounting for fragmentation. This requires the effective mesh size (via [effective_mesh_size\(\)](#)), and the area of patches. This means that you can calculate the change in connectivity if you calculate the effective mesh size of a new habitat/barrier plan, and then use the baseline

Usage

```
connectivity_probability(effective_mesh_size, area_baseline)
```

Arguments

`effective_mesh_size`

As calculated by [effective_mesh_size\(\)](#)

`area_baseline` Numeric vector. Area of a connected patch. This argument is called "baseline" as when you are doing design scenarios you must refer to the baseline area when calculating connectivity probability. See vignette, TODO.

Value

Numeric. Probability of connectedness (0-1).

Examples

```
effective_mesh <- effective_mesh_size(  
  area = lizard_areas_connected$area  
)  
connectivity_probability(  
  effective_mesh_size = effective_mesh,  
  area_baseline = lizard_areas_connected$area  
)  
# if you wanted to compare to a scenario, you would consider the effective  
# mesh size to be the new scenario level, and the baseline as so:  
connectivity_probability(  
  # scenario 1  
  effective_mesh_size = effective_mesh,  
  area_baseline = lizard_areas_connected$area  
)
```

connectivity_report_data

Everything one connectivity analysis produces

Description

Runs the connectivity pipeline once per distance and keeps both the summary and the spatial layers the maps need. `habitat_connectivity()` returns the summary alone; this also keeps the buffered habitat and the patch-ID raster for each distance, which is what the downloadable assets and the report are built from.

Usage

```
connectivity_report_data(
  habitat,
  barrier,
  species,
  interpatch_distance,
  scenario = NULL,
  scenario_kind = NULL,
  scenario_name = NULL,
  verbose = TRUE
)
```

Arguments

habitat	Terra SpatRaster. The habitat layer.
barrier	Terra SpatRaster. The barrier layer.
species	Species name. E.g., "Superb Fairy Wren".
interpatch_distance	Numeric. The distance (in metres) at which habitat patches are considered connected. May be a scalar or a vector; a vector runs the pipeline once per distance.
scenario	Optional layer to compare against the baseline: an sf, SpatVector or SpatRaster standing in for one of habitat and barrier. It is put on the baseline's own grid with <code>onto_grid()</code> , so it does not have to arrive on one. Supplying it runs the pipeline a second time, on the scenario landscape, and fills the <code>scenario_*</code> fields and comparison below.
scenario_kind	Which layer scenario replaces, "habitat" or "barrier". Required with scenario. Only one layer may change at a time, so that the difference in connectivity is attributable to it.
scenario_name	Optional name for the scenario, carried in the comparison so several can be stacked.
verbose	Logical. Display progress messages (default: TRUE).

Value

A connectivity_report_data object: a list of

- connectivity, a connectivity summary with one row per distance
- habitat and barrier, the layers as supplied
- buffered_habitat and patch_id_raster, lists of SpatRaster, one per distance and named by it
- species and interpatch_distance

and, when a scenario was given, the same view of the scenario landscape: scenario_kind, scenario_habitat, scenario_barrier, scenario_buffered_habitat, and comparison. Without one these are all NULL. `scenario_layer()` returns whichever layer the scenario changed.

See Also

`habitat_connectivity()` when only the summary is wanted, and `habitat_connectivity_comparison()` to compare two landscapes you have already assembled yourself.

Examples

```
report_data <- connectivity_report_data(
  habitat = example_wren_habitat(),
  barrier = example_wren_barrier(),
  species = "Superb Fairy Wren",
  interpatch_distance = 200,
  verbose = FALSE
)

report_data$connectivity
names(report_data$patch_id_raster)

# with a scenario, the baseline and the scenario are each run once and
# the comparison is built from the two
with_scenario <- connectivity_report_data(
  habitat = example_wren_habitat(),
  barrier = example_wren_barrier(),
  species = "Superb Fairy Wren",
  interpatch_distance = 200,
  scenario = example_wren_barrier_scenario(),
  scenario_kind = "barrier",
  verbose = FALSE
)

with_scenario$comparison
```

create_barrier_mask	<i>Create barrier mask</i>
---------------------	----------------------------

Description

Converts a barrier layer into a multiplier mask for connectivity analysis. Takes a raster where barriers are coded as 1 (and non-barriers as NA), and inverts it to produce a mask with NA values where barriers exist and 1 elsewhere. This format allows barriers to be applied by multiplying the mask with connectivity surfaces, effectively blocking movement through barrier cells.

Usage

```
create_barrier_mask(barrier)
```

Arguments

barrier Terra SpatRaster. Barrier layer with 1 = barrier, NA = no barrier.

Value

Terra SpatRaster. Mask with 1 where movement is allowed, NA where barriers exist.

Examples

```
lizard_barrier <- example_barrier()
create_barrier_mask(lizard_barrier)
```

display_ids	<i>Which display columns identify a row rather than measure it</i>
-------------	--

Description

So a consumer that has to pivot or drop the identifiers can ask, rather than hardcoding the label text.

Usage

```
display_ids(display)
```

Arguments

display A list from [connectivity_display\(\)](#).

Value

A character vector of column labels.

See Also[connectivity_display\(\)](#)**Examples**

```
lizard <- habitat_connectivity(  
  habitat = example_habitat(),  
  barrier = example_barrier(),  
  species = "Blue Tongue Lizard",  
  interpatch_distance = 20,  
  verbose = FALSE  
)  
  
display_ids(connectivity_display(lizard))
```

`drop_habitat_under_barrier`*Remove habitat under barriers*

Description

Essentially just performs a [terra::mask\(\)](#) operation, to remove the habitat parts that are under the mask.

Usage

```
drop_habitat_under_barrier(habitat, barrier_mask)
```

Arguments

<code>habitat</code>	Terra SpatRaster. Habitat layer.
<code>barrier_mask</code>	Terra SpatRaster. Barrier mask.

Value

Terra SpatRaster with habitat remaining after barrier removal.

Examples

```
lizard_habitat <- example_habitat()  
lizard_barrier <- example_barrier()  
barrier_mask <- create_barrier_mask(lizard_barrier)  
remaining_habitat <- drop_habitat_under_barrier(  
  habitat = lizard_habitat,  
  barrier = lizard_barrier  
)  
remaining_habitat
```

effective_mesh_size	<i>Calculate effective mesh size</i>
---------------------	--------------------------------------

Description

Computes the effective mesh size metric for habitat connectivity. This represents the probability that two randomly chosen points within habitat remain connected. Intended for usage from objects created by `habitat_connectivity()`. See examples below.

Usage

```
effective_mesh_size(area, area_baseline = area)
```

Arguments

area	Numeric vector. Area of connected patches.
area_baseline	Optional. Defaults to area if not specified. Numeric vector of connected patches of a baseline area. This is to allow for comparing the effective mesh size when comparing different scenarios. See future vignette on this topic (TODO).

Value

Numeric. Effective mesh size, in hectares.

Examples

```
effective_mesh_size(lizard_areas_connected$area)
```

empty_grid	<i>Create Empty terra raster grid</i>
------------	---------------------------------------

Description

Create Empty terra raster grid

Usage

```
empty_grid(habitat, resolution = 10)
```

Arguments

habitat	SF object or terra SpatRaster.
resolution	Numeric. Cell size in meters (default: 10).

Value

Terra SpatRaster. Empty raster grid.

Examples

```
lizard_barrier_shp <- example_barrier_shp()
empty_grid(lizard_barrier_shp, resolution = 10)
```

example-lizard-data	<i>Lizard Habitat and Barrier Data from Melbourne.</i>
---------------------	--

Description

We provide Habitat and Barrier data on various lizard species (nominally, Blue-tongued Lizard). The data was collected from Darebin Creek in Melbourne, which runs between Preston and West Heidelberg. For analysis purposes, a interpatch distance of 200 metres is recommended for lizard connectivity assessments.

Usage

```
example_habitat()

example_barrier_shp()

example_barrier()

example_barrier_scenario()
```

Details

We provide helper functions to load the raster and shapefile data. These are required due to how the raster and vector data are stored. These functions provide easy access to example raster and shapefile data included with the package:

- `example_habitat()` Returns a raster of lizard habitat data.
- `example_barrier_shp()` Returns a shapefile of lizard barrier data as an SF object.
- `example_barrier()` Returns a raster of lizard barrier data.
- `example_barrier_scenario()` Returns the barrier raster with a new road cut through it, for comparing a scenario against the baseline.

Value

A terra raster object or sf object depending on the function called

Examples

```
library(terra)

# Load habitat raster
lizard_habitat <- example_habitat()
plot(lizard_habitat, col = "darkgreen", legend = FALSE, main = "Lizard Habitat")

# Load barrier shapefile
lizard_barrier_shp <- example_barrier_shp()
plot(lizard_barrier_shp)

# Load barrier raster
lizard_barrier <- example_barrier()
plot(lizard_barrier, col = c("grey", "white"), legend = FALSE, main = "Lizard Barriers")

plot(lizard_barrier, col = c("grey", "white"), legend = FALSE, main = "Lizard Habitat and Barrier")
plot(lizard_habitat, col = "darkgreen", legend = FALSE, add = TRUE)
```

example-wren-data

Fair Wren Habitat, Barrier, and scenario Data from City of Knox.

Description

We provide Habitat, Barrier, and scenario data on the species, "Superb Fairy Wren". The data was collected from City of Knox, a local government area in Melbourne, on the eastern suburbs, north of Dandenon and south of Ringwood. For analysis purposes, we have also implemented a scenario piece of data.

Usage

```
example_wren_habitat()

example_wren_habitat_scenario()

example_wren_barrier()

example_wren_barrier_scenario()
```

Details

We provide helper functions to load the raster and shapefile data. These are required due to how the raster and vector data are stored. These functions provide easy access to example raster and shapefile data included with the package:

- `example_wren_habitat()` Returns a raster of wren habitat data.
- `example_wren_habitat_scenario()` Returns a raster of wren habitat scenario data.
- `example_wren_barrier()` Returns a raster of wren barrier data.
- `example_wren_barrier_scenario()` Returns a raster of wren barrier scenario data.

Value

A terra raster object or sf object depending on the function called

Examples

```
library(terra)

# Load habitat raster
wren_habitat <- example_wren_habitat()
# Load barrier raster
wren_barrier <- example_wren_barrier()
# load barrier scenario
wren_barrier_scenario <- example_wren_barrier_scenario()
plot(wren_habitat, col = "darkgreen", legend = FALSE, main = "wren Habitat")

plot(
  wren_barrier,
  col = c("grey", "white"),
  legend = FALSE, main = "wren Barriers")

plot(
  wren_barrier,
  col = c("grey", "white"),
  legend = FALSE,
  main = "wren Habitat and Barrier"
)
plot(wren_habitat, col = "darkgreen", legend = FALSE, add = TRUE)

plot(
  wren_barrier_scenario,
  col = c("grey", "white"),
  legend = FALSE,
  main = "wren Habitat and Barrier scenario"
)
plot(wren_habitat, col = "darkgreen", legend = FALSE, add = TRUE)
```

format_resolution	<i>Format a raster resolution for reading</i>
-------------------	---

Description

A reprojected raster rarely has round cells: a nominally 10m grid comes out as 9.99673 x 10.00151. That precision matters in the data and reads as noise on a screen, so round it for display.

Usage

```
format_resolution(res, digits = 1)
```

Arguments

res	Either the numeric pair <code>terra::res()</code> returns, or the "9.99673x10.00151" string carried in a connectivity object's <code>data_resolution</code> column.
digits	Decimal places to round to. Default 1.

Value

A character vector, e.g. "10 x 10".

Examples

```
format_resolution(c(9.99673, 10.00151))
format_resolution("9.99673x10.00151")
```

fragment_habitat	<i>Fragment habitat</i>
------------------	-------------------------

Description

Takes a barrier mask (created with `create_barrier_mask()`) and fragments up the habitat where they intersect.

Usage

```
fragment_habitat(buffered_habitat, barrier_mask)
```

Arguments

buffered_habitat	Terra SpatRaster. Buffered habitat.
barrier_mask	Terra SpatRaster. Barrier mask.

Value

Terra SpatRaster with fragmented habitat.

Examples

```
lizard_habitat <- example_habitat()
lizard_barrier <- example_barrier()
buffered_habitat <- habitat_buffer(lizard_habitat, 5)
barrier_mask <- create_barrier_mask(lizard_barrier)
fragmented <- fragment_habitat(buffered_habitat, barrier_mask)
```

```
generate_connectivity_report
```

Render a connectivity report

Description

One document holding the maps, tables and summary for an analysis: the same figures `write_connectivity_assets()` writes, laid out to read. HTML is a single self-contained file; PDF is rendered through Typst, so no LaTeX is needed.

Usage

```
generate_connectivity_report(x, path = NULL, format = report_format(path))
```

Arguments

<code>x</code>	A <code>connectivity_report_data</code> object from <code>connectivity_report_data()</code> .
<code>path</code>	File to write. The extension sets the format. Defaults to the species, today's date and <code>.html</code> , in the working directory. The directory must already exist.
<code>format</code>	The Quarto format, when <code>path</code> can't say. Taken from <code>path</code> 's extension by default, which is what you want unless the destination is a name something else chose, as <code>shiny::downloadHandler()</code> does.

Details

`write_connectivity_report()` then `render_connectivity_report()`, in one call, through a temporary directory. Use those two instead when you want the Quarto source to keep or to change.

The report covers the tabular and visual results. The GIS layers travel with `zip_connectivity_assets()`, since a GeoTIFF can't live inside a document.

The format comes from `path`'s extension, as it does for `ggplot2::ggsave()`: `"report.pdf"` writes a PDF and `"report.html"` an HTML file. Write both by calling this twice.

Any format Quarto can write to a single file works, so `"report.docx"` and `"report.rtf"` also do what they look like. The Shiny app offers HTML and PDF only. A `.pdf` renders through Typst rather than Quarto's LaTeX-based pdf format, so no TeX install is needed.

Formats that keep their figures in a folder beside the document, such as `.md`, are refused: a report has to be one file, and copying the document alone would lose every figure. Note too that the tabbed sections are HTML-only and fall back to plain headings everywhere else.

Value

The absolute path written, invisibly.

See Also

`connectivity_report_data()` to build `x`, `write_connectivity_report()` for the Quarto source, and `zip_connectivity_assets()` for the GIS layers and full tables.

Examples

```

report_data <- connectivity_report_data(
  habitat = example_habitat(),
  barrier = example_barrier(),
  species = "Blue Tongue Lizard",
  interpatch_distance = 20,
  verbose = FALSE
)

# rendering needs the Quarto command line tool
if (quarto_available()) {
  generate_connectivity_report(
    report_data,
    file.path(tempdir(), "lizard-report.html")
  )
}

```

gg_barrier_habitat_interpatch_dist

Plot barrier, habitat, and interpatch distance layers

Description

Creates a visualisation of habitat, interpatch distance zone, and barriers using terra rasters.

Usage

```

gg_barrier_habitat_interpatch_dist(
  barrier,
  buffered,
  habitat,
  interpatch_distance,
  species,
  col_barrier = urbio_colours()$barrier,
  col_interpatch_dist = urbio_colours()$interpatch_distance,
  col_habitat = urbio_colours()$habitat,
  col_paper = NA
)

```

Arguments

barrier	Terra SpatRaster. Barrier layer (e.g., roads).
buffered	Terra SpatRaster. Buffered habitat layer.
habitat	Terra SpatRaster. Original habitat layer.

<code>interpatch_distance</code>	Numeric. The distance (in meters) where habitat patches are considered connected. E.g., if set to 500, patches 498m apart are connected, those 501m apart are not connected. This is passed internally to a spatial operation known as "buffering", where this distance is used as a radius from the edge of the habitat zone. This means the specified <code>interpatch_distance</code> is halved exactly. So an interpatch distance of 500 will be converted to 250.
<code>species</code>	Character. Species name for plot title.
<code>col_barrier</code>	Character. Colour for barrier layer. Defaults to the package palette, so a map drawn here matches the app, the downloads and the report. See urbio_colours() .
<code>col_interpatch_dist</code>	Character. Colour for interpatch distance zone.
<code>col_habitat</code>	Character. Colour for habitat patches.
<code>col_paper</code>	Character. Background color (default: "white").

Value

A `ggplot2` object.

Examples

```
lizard_habitat <- example_habitat()
lizard_barrier <- example_barrier()
lizard_buffered <- habitat_buffer(lizard_habitat, buffer_radius = 10)
gg_bar_hab_buf <- gg_barrier_habitat_interpatch_dist(
  barrier = lizard_barrier,
  buffered = lizard_buffered,
  habitat = lizard_habitat,
  interpatch_distance = 10,
  species = "Blue Tongue Lizard",
  col_barrier = "black",
  col_interpatch_dist = "lightgreen",
  col_habitat = "seagreen"
)
gg_bar_hab_buf

# add north arrow and scale bar with ggspatial
library(ggspatial)
library(tidyterra)
gg_bar_hab_buf +
  annotation_north_arrow(
    style = north_arrow_fancy_orienteering()
  ) +
  annotation_scale()
```

gg_layer	<i>Plot one layer on its own</i>
----------	----------------------------------

Description

The habitat or the barrier by itself, rather than combined as `gg_barrier_habitat_interpatch_dist()` draws them. For looking at what went into an analysis, and at a scenario layer against the layer it replaces.

Usage

```
gg_layer(layer, kind = c("habitat", "barrier"), title = NULL)
```

Arguments

layer	A SpatRaster. Cells equal to 1 are the layer; everything else is background, whether it is 0 or NA.
kind	Which layer this is, "habitat" or "barrier". Decides the fill and the background.
title	Plot title. Defaults to kind, sentence case.

Details

kind picks the colours, so a layer looks the same here as it does in the combined map. A barrier is white there, read as cuts through the interpatch zone, so a barrier drawn alone is white on that same green: white on white would be nothing at all.

Value

A ggplot.

See Also

`gg_barrier_habitat_interpatch_dist()` for the combined map, and `urbio_colours()` for the palette this draws from.

Examples

```
gg_layer(example_habitat(), "habitat")

gg_layer(example_barrier(), "barrier")

# a scenario layer, titled for what it changes
gg_layer(example_barrier(), "barrier", title = "Scenario: new roads")
```

habitat_buffer	<i>Buffer habitat raster</i>
----------------	------------------------------

Description

Buffer around the habitat a given distance in metres using `terra::focalMat(d = buffer_radius, type = "circle")`. This operation is used to identify connected patches of habitat. Two patches will connect when their edge-to-edge gap is $\leq 2 * \text{buffer_radius}$. So, we recommend you specify the `buffer_radius` value to be half the interpatch distance, which is the distance past which habitat patches are no longer considered connected. For example, if your interpatch distance is 500m, set `buffer_radius = 250`.

Usage

```
habitat_buffer(habitat, buffer_radius)
```

Arguments

<code>habitat</code>	Terra SpatRaster. Habitat raster.
<code>buffer_radius</code>	Numeric. The radius in metres around the habitat. Since patches of habitat will be connected when their edge-to-edge gap is $\leq 2 * \text{buffer_radius}$, we recommend you specify <code>buffer_radius</code> to be half the "interpatch distance". This is the distance past which habitat patches are no longer considered connected. For example, if your interpatch distance is 500m, set <code>buffer_radius = 250</code> . The buffer can only be represented if it is at least one raster cell, i.e. keep <code>resolution <= interpatch_distance / 2</code> . Below that the buffer is a no-op: <code>habitat_buffer()</code> warns and returns the habitat unchanged. See <code>vignette("interpatch-distance-a</code>

Value

Terra SpatRaster with buffered habitat.

See Also

`vignette("interpatch-distance-and-resolution")` for the relationship between interpatch distance, buffer radius, and resolution.

Examples

```
lizard_habitat <- example_habitat()
library(terra)
plot(lizard_habitat, col = "darkgreen", legend = FALSE)
# run with a small buffer radius
lizard_buff <- habitat_buffer(lizard_habitat, buffer_radius = 10)
plot(lizard_buff, col = "lightgreen", legend = FALSE)
plot(lizard_habitat, col = "darkgreen", legend = FALSE, add = TRUE)
```

habitat_connectivity *Calculate habitat connectivity using terra*

Description

This performs the entire connectivity workflow, returning a dataframe output. The steps are:

- `create_barrier_mask()`: Creating barrier mask.
- `drop_habitat_under_barrier()`: Removes Habitat underneath barrier.
- `habitat_buffer()`: Buffers the habitat layer by the interpatch distance (m).
- `fragment_habitat()`: Fragments habitat layer along barrier intersection.
- `assign_patches_to_fragments()`: Assign patch ID to fragments.
- `aggregate_connected_patches()`: Summarise area in each patch.

Usage

```
habitat_connectivity(
  habitat,
  barrier,
  species,
  interpatch_distance,
  verbose = TRUE
)
```

Arguments

habitat	Terra SpatRaster. Habitat raster.
barrier	Terra SpatRaster. Barrier raster.
species	Species name. E.g., "Blue-tongued Lizard".
interpatch_distance	Numeric. The distance (in meters) where habitat patches are considered connected. E.g., if set to 500, patches 498m apart are connected, those 501m apart are not connected. This is passed internally to a spatial operation known as "buffering", where this distance is used as a radius from the edge of the habitat zone. This means the specified interpatch_distance is halved exactly. So an interpatch distance of 500 will be converted to 250. For the buffer to be representable on the raster, keep resolution <= interpatch_distance / 2; below that the buffer is a no-op and a warning is raised. See <code>vignette("interpatch-distance-and-resolution")</code> .
verbose	Logical. Display progress messages (default: TRUE).

Value

A connectivity object (one row): a tibble of landscape-level connectivity metrics (patch count, effective mesh size, probability of connectedness, mean and total patch area) for the species and interpatch distance supplied. The per-patch areas the summary is built from are stored in a list column, "patch_size" – retrieve them with `patch_sizes()`.

See Also

`vignette("interpatch-distance-and-resolution")` for the relationship between interpatch distance, buffer radius, and resolution.

Examples

```
lizard_habitat <- example_habitat()
lizard_barrier <- example_barrier()
connectivity <- habitat_connectivity(
  habitat = lizard_habitat,
  barrier = lizard_barrier,
  species = "Blue-tongued Lizard",
  interpatch_distance = 12
)
connectivity

# get the patch size:
patch_sizes(connectivity)[[1]]
```

habitat_connectivity_comparison

Compare habitat connectivity of a scenario against a baseline

Description

Starts from habitat and barrier layers, rather than the connectivity objects `compare_connectivity()` takes. Give it a scenario habitat/barrier pair and a baseline habitat/barrier pair and it runs the full connectivity pipeline on each (via `habitat_connectivity()`) and compares them. `interpatch_distance` may be a vector, in which case the comparison is run once per distance and the per-distance results are stacked.

Usage

```
habitat_connectivity_comparison(
  habitat_scenario,
  barrier_scenario,
  habitat_baseline,
  barrier_baseline,
  species,
  interpatch_distance,
  scenario_name = NULL,
  verbose = TRUE
)
```

Arguments

habitat_scenario	Terra SpatRaster. The scenario habitat layer.
barrier_scenario	Terra SpatRaster. The scenario barrier layer.
habitat_baseline	Terra SpatRaster. The baseline habitat layer.
barrier_baseline	Terra SpatRaster. The baseline barrier layer.
species	Species name. E.g., "Superb Fairy Wren".
interpatch_distance	Numeric. The distance (in metres) at which habitat patches are considered connected. May be a scalar or a vector; a vector runs the comparison once per distance. See habitat_connectivity() .
scenario_name	Character. An optional label for the scenario, for example "Bentley Project". Passed to compare_connectivity() , and appears in the scenario_name column on every row. Defaults to NULL, which gives NA.
verbose	Logical. Display progress messages (default: TRUE).

Details

Change exactly one layer between baseline and scenario so the difference is attributable to a single variable. To assess a barrier change, pass the same habitat to both `habitat_baseline` and `habitat_scenario`; to assess a habitat change, pass the same barrier to both `barrier_baseline` and `barrier_scenario`. If both the habitat *and* the barrier differ the function aborts; if neither differs it warns (the scenario equals the baseline, so every change value is zero).

Note that this runs the full connectivity pipeline $2 * \text{length}(\text{interpatch_distance})$ times (buffering is the slow step), so a long vector of distances will take proportionally longer.

Value

A `compare_connectivity` object: a tibble with four rows (baseline, scenario, change, pct_change) per interpatch distance. A scalar `interpatch_distance` gives four rows; a length-3 vector gives twelve rows (four per distance). See [compare_connectivity\(\)](#) for the columns.

See Also

[compare_connectivity\(\)](#), [habitat_connectivity\(\)](#)

Examples

```
habitat_connectivity_comparison(
  habitat_scenario = example_wren_habitat(),
  barrier_scenario = example_wren_barrier_scenario(),
  habitat_baseline = example_wren_habitat(),
  barrier_baseline = example_wren_barrier(),
  species = "Superb Fairy Wren",
```

```

    interpatch_distance = 200,
    verbose = FALSE
  )

```

habitat_connectivity_full

Calculate habitat connectivity with visualization data

Description

Like `habitat_connectivity()`, but also returns the intermediate rasters (buffered habitat, patch ID raster, barrier mask, remaining habitat) useful for mapping and reporting.

Usage

```

habitat_connectivity_full(
  habitat,
  barrier,
  interpatch_distance,
  verbose = TRUE
)

```

Arguments

habitat	Terra SpatRaster. Habitat raster.
barrier	Terra SpatRaster. Barrier raster.
interpatch_distance	Numeric. The distance (in meters) where habitat patches are considered connected. E.g., if set to 500, patches 498m apart are connected, those 501m apart are not connected. This is passed internally to a spatial operation known as "buffering", where this distance is used as a radius from the edge of the habitat zone. This means the specified <code>interpatch_distance</code> is halved exactly. So an interpatch distance of 500 will be converted to 250. For the buffer to be representable on the raster, keep <code>resolution <= interpatch_distance / 2</code> ; below that the buffer is a no-op and a warning is raised. See vignette("interpatch-distance-and-resolution").
verbose	Logical. Display progress messages (default: TRUE).

Value

Named list with elements: `buffered_habitat`, `patch_id_raster`, `areas_connected`, `barrier_mask`, `remaining_habitat`.

Examples

```

lizard_habitat <- example_habitat()
lizard_barrier <- example_barrier()
result <- habitat_connectivity_full(
  lizard_habitat,
  lizard_barrier,
  interpatch_distance = 10,
  verbose = FALSE
)
names(result)

```

habitat_connectivity_scenarios

Compare several scenarios against one baseline, starting from layers

Description

The layer-level counterpart to [compare_scenarios\(\)](#). Give it one baseline habitat/barrier pair and named lists of scenario layers, and it runs the connectivity pipeline on each and compares them all against that baseline.

Usage

```

habitat_connectivity_scenarios(
  habitat_baseline,
  barrier_baseline,
  species,
  habitat_scenarios = NULL,
  barrier_scenarios = NULL,
  interpatch_distance,
  verbose = TRUE
)

```

Arguments

habitat_baseline	Terra SpatRaster. The baseline habitat layer.
barrier_baseline	Terra SpatRaster. The baseline barrier layer.
species	Species name. E.g., "Superb Fairy Wren".
habitat_scenarios	A named list of habitat layers, one per scenario, each compared against habitat_baseline with the barrier held at barrier_baseline. Names become the scenario_name column.
barrier_scenarios	A named list of barrier layers, the mirror of habitat_scenarios. Supply either list, or both; names must be unique across the two.

interpatch_distance	Numeric. The distance (in metres) at which habitat patches are considered connected. May be a scalar or a vector; a vector runs every scenario once per distance.
verbose	Logical. Display progress messages (default: TRUE).

Details

A scenario changes exactly one layer: habitat scenarios are run against the baseline barrier, and barrier scenarios against the baseline habitat. There's no way to write a scenario that changes both, which is what [habitat_connectivity_comparison\(\)](#) has to check for.

The baseline is run once per distance and reused by every scenario, so comparing three scenarios at one distance runs the pipeline four times, not six.

Value

A `compare_connectivity` object with four rows per scenario per distance, labelled in the `scenario_name` column. Habitat scenarios come first, then barrier scenarios, in the order given.

See Also

[compare_scenarios\(\)](#) when you already have connectivity objects, and [habitat_connectivity_comparison\(\)](#) for a single scenario.

Examples

```
wren_habitat <- example_wren_habitat()
wren_barrier <- example_wren_barrier()

habitat_connectivity_scenarios(
  habitat_baseline = wren_habitat,
  barrier_baseline = wren_barrier,
  species = "Superb Fairy Wren",
  habitat_scenarios = list(
    "Bentley Project" = example_wren_habitat_scenario()
  ),
  barrier_scenarios = list(
    "Barrier upgrade" = example_wren_barrier_scenario()
  ),
  interpatch_distance = 200,
  verbose = FALSE
)
```

lizard_areas_connected*Connected habitat patch areas for Blue-tongued Lizard*

Description

Pre-computed per-patch areas for the lizard example data at a 50 metre interpatch distance. This is a `patch_size_tbl` – the per-patch table carried inside the connectivity object returned by `habitat_connectivity()`, extracted with the `patch_sizes()` accessor. Contains one row per connected habitat patch.

Usage

```
lizard_areas_connected
```

Format

A `patch_size_tbl` with columns:

patch_id Integer. Connected fragment ID.

area Numeric. Total area of the connected patch in square metres.

Source

Generated from `example_habitat()` and `example_barrier()` at 50 metre interpatch distance.

See Also

`habitat_connectivity()`, `patch_sizes()`, `summarise_connectivity()`

Examples

```
# This was the code that was run to create this object. We don't run it
# as it takes some time to run
## Not run:
lizard_connectivity <- habitat_connectivity(
  habitat = example_habitat(),
  barrier = example_barrier(),
  species = "Blue-tongued Lizard",
  interpatch_distance = 50,
  verbose = FALSE
)
lizard_areas_connected <- patch_sizes(lizard_connectivity)[[1]]

## End(Not run)
lizard_areas_connected
```

mean_patch_size	<i>Calculate mean patch size</i>
-----------------	----------------------------------

Description

This is just a wrapper around `mean()`, however it is written to clearly identify it's usage in the context of the area data. Intended for usage from objects created by `habitat_connectivity()`. See examples below.

Usage

```
mean_patch_size(area, ...)
```

Arguments

area	Numeric vector. Area of a connected patch.
...	extra arguments to pass to <code>mean()</code> .

Value

Numeric. Mean patch size.

Examples

```
mean_patch_size(lizard_areas_connected$area)
```

n_patches	<i>Count number of habitat patches</i>
-----------	--

Description

Identify the number of habitat patches. A wrapper around `length()`, but named to establish its context. Intended for usage from objects created by `habitat_connectivity()`. See examples below.

Usage

```
n_patches(area)
```

Arguments

area	Numeric vector. Area of a connected patch.
------	--

Value

Integer. Number of patches.

Examples

```
n_patches(lizard_areas_connected$area)
```

new_patch_size_tbl	<i>A set of connected habitat patches</i>
--------------------	---

Description

A [tibble](#) of connected patch areas (one row per patch) that also carries the species and interpatch_distance the analysis was run with as attributes. [habitat_connectivity\(\)](#) builds one of these internally for each call; retrieve it from the returned connectivity summary with [patch_sizes\(\)](#).

Usage

```
new_patch_size_tbl(data, species, interpatch_distance, res = NA_real_)
```

```
patch_size_tbl(data, species, interpatch_distance, res = NA_real_)
```

Arguments

data	Data frame of connected patches. Must contain an area column.
species	Character of length 1. Species the analysis was run for.
interpatch_distance	Numeric of length 1. The interpatch distance (m) the analysis used.
res	resolution in pixels - defaults to NA (numeric), not required for vector based approaches.

Details

Because it is a tibble subclass it behaves like a data frame directly – \$, [, [DT::datatable\(\)](#), [utils::write.csv\(\)](#) and ggplot2 all work without ceremony. Read the metadata back with [pc_species\(\)](#) and [pc_interpatch_distance\(\)](#).

Value

A patch_size_tbl object: a tibble with species and interpatch_distance attributes.

 onto_grid

Put a layer on a grid, whichever form it arrives in

Description

A vector layer is rasterised; a `SpatRaster` is already cells, so it is resampled onto the grid instead. This is the one place that decides what an empty cell means, which is why `background` is required: `NA` for habitat, because a cell with no habitat in it has no value, and `0` for a barrier, because a cell with no barrier in it is not blocked.

Usage

```
onto_grid(layer, grid, background)
```

Arguments

<code>layer</code>	An <code>sf</code> , <code>SpatVector</code> or <code>SpatRaster</code> layer.
<code>grid</code>	A <code>SpatRaster</code> whose geometry the result takes, from <code>empty_grid()</code> or an already-prepared layer.
<code>background</code>	Value for cells the layer does not cover. Ignored when <code>layer</code> is a <code>SpatRaster</code> , which brings its own.

Details

Use it to put a scenario layer on the grid of the layer it replaces. A scenario rasterised onto a grid of its own extent is a different landscape, and `habitat_connectivity_comparison()` will refuse to compare it.

Value

A `SpatRaster` on `grid`'s geometry.

See Also

`prepare_rasters()`, which uses this for both of its layers.

Examples

```
grid <- empty_grid(example_barrier_shp(), resolution = 10)

# a vector layer is rasterised
onto_grid(example_barrier_shp(), grid, background = 0)

# a raster one is resampled
onto_grid(example_habitat(), grid, background = NA)
```

patch_sizes	<i>Extract the per-patch tables from a connectivity object</i>
-------------	--

Description

A connectivity object (from [habitat_connectivity\(\)](#) or [summarise_connectivity\(\)](#)) carries the underlying per-patch areas in a patch_size list-column. `patch_sizes()` returns them.

Usage

```
patch_sizes(x)
```

Arguments

x A connectivity object.

Value

A list of [patch_size_tbl\(\)](#) objects, one per row of x (always a list, even for a single-row summary).

pc-getters	<i>Metadata from a patch_size_tbl object</i>
------------	--

Description

Metadata from a patch_size_tbl object

Usage

```
pc_species(x)

pc_patches(x)

pc_res(x)

pc_interpatch_distance(x)
```

Arguments

x A [patch_size_tbl\(\)](#) object.

Value

- `pc_species()` Returns the species (character, length 1).
- `pc_interpatch_distance()` returns the interpatch distance (numeric, length 1).
- `pc_res()` returns the resolution (character, length 1 - e.g., "2x2").
- `pc_patches()` returns the number of patches - computed live from the number of rows (numeric, length 1).

`plot_barrier_habitat_interpatch_dist`

Save barrier habitat interpatch distance plot

Description

Draws the plot `gg_barrier_habitat_interpatch_dist()` makes and writes it to a PNG in the working directory, at the shared size from `urbio_figure_size()`. Returns the file path, which is what a `targets::tar_file()` target wants.

Usage

```
plot_barrier_habitat_interpatch_dist(
  barrier,
  buffered,
  habitat,
  interpatch_distance,
  species,
  col_barrier = urbio_colours()$barrier,
  col_interpatch_dist = urbio_colours()$interpatch_distance,
  col_habitat = urbio_colours()$habitat,
  col_paper = NA
)
```

Arguments

<code>barrier</code>	barrier layer
<code>buffered</code>	buffered layer
<code>habitat</code>	habitat layer
<code>interpatch_distance</code>	Numeric. The distance (in meters) where habitat patches are considered connected. E.g., if set to 500, patches 498m apart are connected, those 501m apart are not connected. This is passed internally to a spatial operation known as "buffering", where this distance is used as a radius from the edge of the habitat zone. This means the specified <code>interpatch_distance</code> is halved exactly. So an interpatch distance of 500 will be converted to 250.
<code>species</code>	character, species name, e.g., "Superb Fairy Wren"

col_barrier colour to colour the barrier layer. Defaults to the package palette, see [urbio_colours\(\)](#).
 col_interpatch_dist
 colour to colour the interpatch distance layer
 col_habitat colour to colour the habitat layer
 col_paper colour to colour the paper layer of ggplot

Value

Named character vector. The file path, named by the interpatch distance.

Examples

```
## Not run:
lizard_habitat <- example_habitat()
lizard_barrier <- example_barrier()
buffered <- habitat_buffer(lizard_habitat, interpatch_distance = 10)
# Creates plot-barrier-interpatch-dist-habitat-*.png in the working directory
plot_barrier_habitat_interpatch_dist(
  barrier = lizard_barrier,
  buffered = buffered,
  habitat = lizard_habitat,
  interpatch_distance = 10,
  species = "Blue-tongued Lizard",
  col_barrier = "white",
  col_interpatch_dist = "lightgreen",
  col_habitat = "seagreen",
  col_paper = "grey50"
)

## End(Not run)
```

plot_connectivity	<i>Plot connectivity metrics across interpatch distances</i>
-------------------	--

Description

Creates faceted line plots showing how connectivity metrics change with different interpatch distances. This works best when you have multiple interpatch distances, otherwise it will just be a plot with one point.

Usage

```
plot_connectivity(results_connect_habitat)
```

Arguments

results_connect_habitat
 Data frame. Connectivity summary results with columns for species, interpatch distance, and various metrics.

Value

A ggplot2 object with faceted plots of connectivity metrics.

Examples

```
lizard_habitat <- example_habitat()
lizard_barrier <- example_barrier()
results <- purrr::map(
  c(10, 20),
  function(d) {
    full <- habitat_connectivity_full(lizard_habitat, lizard_barrier,
      interpatch_distance = d, verbose = FALSE)
    summarise_connectivity(
      connectivity = full$areas_connected$area,
      interpatch_distance = d,
      data_resolution = 10,
      species = "Blue-tongued Lizard"
    )
  }
) |> purrr::list_rbind()
plot_connectivity(results)
```

plot_patches

Plot connected habitat patches

Description

Visualizes habitat patches colored by their connected fragment ID.

Usage

```
plot_patches(patch_id, interpatch_distance, species = "Species", n_cols = 7)
```

Arguments

patch_id	Terra SpatRaster. Raster with patch IDs.
interpatch_distance	Numeric. The distance (in meters) where habitat patches are considered connected. E.g., if set to 500, patches 498m apart are connected, those 501m apart are not connected. This is passed internally to a spatial operation known as "buffering", where this distance is used as a radius from the edge of the habitat zone. This means the specified interpatch_distance is halved exactly. So an interpatch distance of 500 will be converted to 250.
species	Character. Species name (default: "Species").
n_cols	Integer. Number of colors to cycle through (default: 7).

Value

A ggplot2 object showing patches with distinct colors.

Examples

```
lizard_habitat <- example_habitat()
lizard_barrier <- example_barrier()
interpatch_distance <- 20
buffer_radius <- interpatch_distance / 2
buffered_habitat <- habitat_buffer(lizard_habitat, buffer_radius)
barrier_mask <- create_barrier_mask(lizard_barrier)
fragmented <- fragment_habitat(buffered_habitat, barrier_mask)
remaining_habitat <- drop_habitat_under_barrier(
  habitat = lizard_habitat,
  barrier = lizard_barrier
)
fragment_patches <- assign_patches_to_fragments(
  remaining_habitat = remaining_habitat,
  fragment = fragmented
) |> add_patch_area()

plot_patches(fragment_patches, interpatch_distance = interpatch_distance)

#' add north arrow and scale bar with ggspatial
library(ggspatial)
library(tidyterra)
plot_patches(fragment_patches, interpatch_distance = interpatch_distance) +
  annotation_north_arrow(
    style = north_arrow_fancy_orienteering()
  ) +
  annotation_scale()
```

```
prepare_rasters
```

Prepare habitat and barrier rasters

Description

Put habitat and barrier layers on one grid. Either layer may be a vector (sf or SpatVector), which is rasterised, or a SpatRaster, which is resampled onto the grid.

Usage

```
prepare_rasters(
  habitat,
  barrier,
  data_resolution = 10,
  target_resolution = 500
)
```

Arguments

habitat SF object or SpatRaster. Habitat spatial data.
barrier SF object or SpatRaster. Barrier spatial data.
data_resolution Numeric. Fine resolution in meters. Default, 10.
target_resolution Numeric. Coarse resolution in meters. Default, 500.

Value

List with `habitat_raster` and `barrier_raster` elements.

Examples

```

lizard_habitat_sf <- terra::as.polygons(example_habitat(), dissolve = TRUE) |>
  sf::st_as_sf()
lizard_barrier_shp <- example_barrier_shp()
prepare_rasters(lizard_habitat_sf, lizard_barrier_shp)

# a raster layer is resampled onto the grid rather than rasterised
prepare_rasters(example_habitat(), lizard_barrier_shp)

```

quarto_available	<i>Is the Quarto command line tool installed?</i>
------------------	---

Description

Quarto is separate software, not an R package, so it can be missing. The reports need it; nothing else in `urbioconnect` does. Ask this to decide what to offer: the Shiny app disables its report buttons when it returns `FALSE`, and reports defaults to it in `write_connectivity_assets()`.

Usage

```
quarto_available()
```

Value

TRUE or FALSE.

See Also

`generate_connectivity_report()`, which needs it.

Examples

```
quarto_available()
```

read_geometry	<i>Read shapefile geometry</i>
---------------	--------------------------------

Description

Reads a shapefile and extracts only the spatial geometry, discarding attribute data.

Usage

```
read_geometry(shapefile)

## S3 method for class 'sf'
read_geometry(shapefile)

## Default S3 method:
read_geometry(shapefile)
```

Arguments

shapefile Character. File path to a shapefile, or an SF object.

Value

An sfc object containing only the spatial geometry.

Examples

```
# Read geometry from a file path
barrier_path <- system.file("ex/lizard_barrier.shp", package = "urbiocconnect")
barrier_geom <- read_geometry(barrier_path)

# Can also pass an existing SF object
barrier_sf <- sf::st_read(barrier_path, quiet = TRUE)
barrier_geom <- read_geometry(barrier_sf)
```

render_connectivity_report	<i>Render a connectivity report's source</i>
----------------------------	--

Description

Renders a .qmd from [write_connectivity_report\(\)](#). The document carries its own figure sizes, so rendering it yourself with the Render button or `quarto::quarto_render()` gives the same figures; this adds the choice of format by extension, the Typst routing for a .pdf, and puts the output where you ask rather than beside the input.

Usage

```
render_connectivity_report(input, path = NULL, format = report_format(path))
```

Arguments

input	The .qmd to render, from <code>write_connectivity_report()</code> . Its -data.rds must still be beside it.
path	File to write. The extension sets the format, as it does for <code>ggplot2::ggsave()</code> . Defaults to input with a .html extension. The directory must already exist.
format	The Quarto format, when path can't say. See <code>generate_connectivity_report()</code> .

Value

The absolute path written, invisibly.

See Also

`write_connectivity_report()` to write the source.

Examples

```
report_data <- connectivity_report_data(
  habitat = example_habitat(),
  barrier = example_barrier(),
  species = "Blue Tongue Lizard",
  interpatch_distance = 20,
  verbose = FALSE
)

qmd <- write_connectivity_report(
  report_data,
  file.path(tempdir(), "lizard-report.qmd")
)

# rendering needs the Quarto command line tool
if (quarto_available()) {
  render_connectivity_report(qmd)
}
```

round_by

Apply the rounding a display describes

Description

`round_by()` rounds and leaves the values numeric, which is right when each column holds one metric. `format_by()` formats each value to a string instead, which is what a column of several metrics needs: a numeric column spanning 1e-5 to 1e4 prints wholly in scientific notation, so a count of 3 reads as 3.00e+00. `DT::datatable()` formats cell by cell and so needs neither; `knitr::kable()` formats by column and so needs `format_by()` for a comparison.

Usage

```
round_by(display)
```

```
format_by(display)
```

Arguments

display A list from `connectivity_display()`.

Value

The data tibble: numeric from `round_by()`, character from `format_by()`.

See Also

`connectivity_display()`

Examples

```
lizard <- habitat_connectivity(  
  habitat = example_habitat(),  
  barrier = example_barrier(),  
  species = "Blue Tongue Lizard",  
  interpatch_distance = 20,  
  verbose = FALSE  
)  
  
round_by(connectivity_display(lizard))
```

run_connectivity_app *Launch the Connectivity Shiny App*

Description

Launches the connectivity analysis Shiny application.

Usage

```
run_connectivity_app()
```

Value

No return value, called for side effects (launches Shiny app)

Examples

```
## Not run:  
run_connectivity_app()  
  
## End(Not run)
```

scenario_layer	<i>The layer a scenario changed</i>
----------------	-------------------------------------

Description

An analysis carrying a scenario holds both of its landscapes, so the layer that actually changed is whichever of them the scenario stood in for. Ask rather than branching on scenario_kind at every call site.

Usage

```
scenario_layer(x)
```

Arguments

x A connectivity_report_data, from [connectivity_report_data\(\)](#).

Value

The scenario's SpatRaster, or NULL when there is no scenario.

See Also

[connectivity_report_data\(\)](#), and [gg_layer\(\)](#) to draw it.

Examples

```
analysis <- connectivity_report_data(  
  habitat = example_wren_habitat(),  
  barrier = example_wren_barrier(),  
  species = "Superb Fairy Wren",  
  interpatch_distance = 200,  
  scenario = example_wren_barrier_scenario(),  
  scenario_kind = "barrier",  
  verbose = FALSE  
)  
  
scenario_layer(analysis)
```

sf_add_patch_area	<i>Add patch area column</i>
-------------------	------------------------------

Description

Add patch area column

Usage

```
sf_add_patch_area(patchess)
```

Arguments

patchess SF object. Habitat patches.

Value

SF object with added area column in square meters.

Examples

```
lizard_habitat_sf <- terra::as.polygons(example_habitat(), dissolve = TRUE) |>
  sf::st_as_sf()
## Not run:
lizard_barrier_shp <- example_barrier_shp()
buffered <- sf_habitat_buffer(lizard_habitat_sf, buffer_radius = 10)
fragments <- sf_fragment_habitat(buffered, lizard_barrier_shp)
remaining <- sf_drop_habitat_under_barrier(lizard_habitat_sf, lizard_barrier_shp)
patchess <- sf_assign_patches_to_fragments(remaining, fragments)
sf_add_patch_area(patchess)

## End(Not run)
```

sf_aggregate_connected_patches	<i>Aggregate connected patch areas</i>
--------------------------------	--

Description

Groups habitat patches by their connected fragment ID and calculates total and squared areas for connectivity metrics.

Usage

```
sf_aggregate_connected_patches(patch_areas)
```

Arguments

patch_areas SF object. Habitat patches with area column.

Value

Data frame with patch_id, area, and area_squared columns.

Examples

```
lizard_habitat_sf <- terra::as.polygons(example_habitat(), dissolve = TRUE) |>
  sf::st_as_sf()
lizard_barrier_shp <- example_barrier_shp()
## Not run:
buffered <- sf_habitat_buffer(lizard_habitat_sf, buffer_radius = 10)
fragments <- sf_fragment_habitat(buffered, lizard_barrier_shp)
remaining <- sf_drop_habitat_under_barrier(lizard_habitat_sf, lizard_barrier_shp)
patches <- sf_assign_patches_to_fragments(remaining, fragments) |>
  sf_add_patch_area()
sf_aggregate_connected_patches(patches)

## End(Not run)
```

sf_assign_patches_to_fragments

Assign habitat patches to fragment IDs

Description

Determines which connected fragment each remaining habitat patch belongs to based on spatial intersection.

Usage

```
sf_assign_patches_to_fragments(remaining, fragment_id)
```

Arguments

remaining SF object. Remaining habitat patches after barrier removal.

fragment_id SF object. Fragment geometries with IDs.

Value

SF object with habitat patches labeled by their fragment ID.

Examples

```
lizard_habitat_sf <- terra::as.polygons(example_habitat(), dissolve = TRUE) |>
  sf::st_as_sf()
lizard_barrier_shp <- example_barrier_shp()
## Not run:
buffered <- sf_habitat_buffer(lizard_habitat_sf, buffer_radius = 10)
fragments <- sf_fragment_habitat(buffered, lizard_barrier_shp)
remaining <- sf_drop_habitat_under_barrier(lizard_habitat_sf, lizard_barrier_shp)
sf_assign_patches_to_fragments(remaining, fragments)

## End(Not run)
```

sf_drop_habitat_under_barrier

Remove habitat underneath barriers

Description

Removes all habitat areas that intersect with barriers and splits multipolygons into individual patches.

Usage

```
sf_drop_habitat_under_barrier(habitat, barrier)
```

Arguments

habitat	SF object. Original habitat geometry.
barrier	SF object. Barrier geometry.

Value

SF object with habitat patches that don't intersect barriers.

Examples

```
lizard_habitat_sf <- terra::as.polygons(example_habitat(), dissolve = TRUE) |>
  sf::st_as_sf()
lizard_barrier_shp <- example_barrier_shp()
sf_drop_habitat_under_barrier(lizard_habitat_sf, lizard_barrier_shp)
```

sf_fragment_habitat	<i>Fragment habitat along barriers</i>
---------------------	--

Description

Removes barrier areas from buffered habitat and splits the result into individual polygon fragments.

Usage

```
sf_fragment_habitat(habitat_buffered, barrier)
```

Arguments

habitat_buffered	SF object. Buffered habitat geometry.
barrier	SF object. Barrier geometry (e.g., roads).

Value

SF object with individual habitat fragments, each with a unique ID.

Examples

```
lizard_habitat_sf <- terra::as.polygons(example_habitat(), dissolve = TRUE) |>
  sf::st_as_sf()
lizard_barrier_shp <- example_barrier_shp()
## Not run:
buffered <- sf_habitat_buffer(lizard_habitat_sf, interpatch_distance = 10)
sf_fragment_habitat(buffered, lizard_barrier_shp)

## End(Not run)
```

sf_habitat_buffer	<i>Buffer habitat by distance</i>
-------------------	-----------------------------------

Description

Creates a buffer around habitat polygons and unions overlapping areas into a single polygon, using `sf::st_buffer()`. Unlike the raster `habitat_buffer()`, this works in continuous coordinate space, so there is **no resolution constraint**: any `buffer_radius` produces an exact buffer and the sub-cell "no buffer" problem of the raster path does not apply. The buffer arc is approximated by `nQuadSegs` straight segments per quarter circle (here 5, i.e. a 20-sided polygon), which affects the *smoothness* of the outline, not whether the buffer forms. See `vignette("interpatch-distance-and-resolution")`.

Usage

```
sf_habitat_buffer(habitat, buffer_radius)
```

Arguments

habitat	SF object. Habitat spatial data.
buffer_radius	Numeric. The radius in metres around the habitat. Specify it as half the interpatch distance (see habitat_buffer()). Because vector buffering is done in continuous space, there is no minimum representable radius - unlike the raster path, there is no resolution below which the buffer becomes a no-op.

Value

SF object with buffered and unioned habitat geometry.

Examples

```
lizard_habitat_sf <- terra::as.polygons(example_habitat(), dissolve = TRUE) |>
  sf::st_as_sf()
## Not run:
sf_habitat_buffer(lizard_habitat_sf, buffer_radius = 10)

## End(Not run)
```

sf_habitat_connectivity

Calculate habitat connectivity

Description

Performs complete habitat connectivity analysis using vector-based spatial operations. Buffers habitat, fragments it along barriers, and calculates areas of connected patches.

Usage

```
sf_habitat_connectivity(habitat, barrier, species, interpatch_distance)
```

Arguments

habitat	SF object. Original habitat spatial data.
barrier	SF object. Barrier spatial data (e.g., roads, waterways).
species	character. Species name.
interpatch_distance	Numeric. The distance (in meters) where habitat patches are considered connected. E.g., if set to 500, patches 498m apart are connected, those 501m apart are not connected. This is passed internally to a spatial operation known as "buffering", where this distance is used as a radius from the edge of the habitat zone. This means the specified interpatch_distance is halved exactly. So an interpatch distance of 500 will be converted to a 250m buffer radius.

Value

Data frame with connectivity metrics for each connected patch, including patch_id, area, and area_squared.

Examples

```
lizard_barrier_shp <- example_barrier_shp()
lizard_habitat_sf <- terra::as.polygons(
  example_habitat(),
  dissolve = TRUE
) |>
sf::st_as_sf()
result <- sf_habitat_connectivity(
  habitat = lizard_habitat_sf,
  barrier = lizard_barrier_shp,
  species = "Blue-tongued lizard",
  interpatch_distance = 10
)
result
```

summarise-connectivity

Summarise connectivity metrics

Description

Calculates a comprehensive set of habitat connectivity metrics including effective mesh size, probability of connectedness, and patch statistics. There are two methods, the default, and one that dispatches on objects of class, patch_size_tbl, created by [patch_size_tbl\(\)](#), which is used mostly internally inside of [habitat_connectivity\(\)](#). The default method requires "connectivity" to be a vector of areas, and also requires scalar values (values of length 1) of interpatch distance, resolution, and species. See examples below.

Usage

```
summarise_connectivity(connectivity, ...)

## Default S3 method:
summarise_connectivity(
  connectivity,
  interpatch_distance,
  data_resolution,
  species,
  ...
)
```

Arguments

connectivity	data.frame of class "patch_size_tbl", obtained via <code>patch_sizes()</code> from a connectivity object returned by <code>habitat_connectivity()</code> . Contains area measurements of connected patches.
...	extra arguments to pass through for default method.
interpatch_distance	Numeric. The distance (in meters) where habitat patches are considered connected. E.g., if set to 500, patches 498m apart are connected, those 501m apart are not connected. This is passed internally to a spatial operation known as "buffering", where this distance is used as a radius from the edge of the habitat zone. This means the specified interpatch_distance is halved exactly. So an interpatch distance of 500 will be converted to 250.
data_resolution	Numeric. Data resolution in meters.
species	Character. Name of species analysed.

Value

A tibble with connectivity metrics including number of patches, probability of connectedness, effective mesh size, mean and total patch areas.

Examples

```
# dispatch on `patch_size_tbl` class - `lizard_areas_connected`
summarise_connectivity(
  connectivity = lizard_areas_connected
)

# default method
# manually passing area, interpatch distance, resolution, and species
summarise_connectivity(
  connectivity = lizard_areas_connected$area,
  interpatch_distance = 10,
  data_resolution = 10,
  species = "Blue-tongued Lizard"
)
```

total_habitat_area	<i>Calculate total habitat area</i>
--------------------	-------------------------------------

Description

Calculate the total habitat area in hectares. A wrapper around summing the area and multiplying by 0.0001 to give the units in hectares. Intended for usage from objects created by `habitat_connectivity()`. See examples below.

Usage

```
total_habitat_area(area)
```

Arguments

area Numeric vector. Area of a connected patch.

Value

Numeric. Total habitat area in hectares.

Examples

```
total_habitat_area(lizard_areas_connected$area)
```

urbio_colours

The urbioconnect map colours

Description

The palette the package's maps use, so a map downloaded from the Shiny app matches the one on screen. Based on scico's tofino palette.

Usage

```
urbio_colours()
```

Value

A named list of colours: habitat, interpatch_distance and barrier.

Examples

```
urbio_colours()
```

urbio_figure_size *The default figure size for urbiocconnect maps and plots*

Description

Shared by the downloadable PNGs and the report, so a map keeps its shape across both. Width and heights are inches and dpi is dots per inch, as `ggplot2::ggsave()` and Quarto's `fig-width`, `fig-height` and `fig-dpi` expect. `tall_height` is for the faceted over-distance plot, which needs more vertical room than a map.

Usage

```
urbio_figure_size()
```

Value

A named list: `width`, `height`, `tall_height` and `dpi`.

Examples

```
urbio_figure_size()
```

write_connectivity_assets *Write the downloadable assets for one analysis*

Description

Writes every map, table and GIS layer a planner needs into `dir`, laid out in folders by interpatch distance, with a README explaining each file. `zip_connectivity_assets()` wraps this into a single archive, which is what the Shiny app's download button uses.

Usage

```
write_connectivity_assets(x, dir, reports = quarto_available())
```

Arguments

<code>x</code>	A <code>connectivity_report_data</code> object from <code>connectivity_report_data()</code> .
<code>dir</code>	Directory to write into. Created if it doesn't exist.
<code>reports</code>	Also render the HTML and PDF reports into <code>dir</code> . Defaults to whether the Quarto command line tool is installed, since it is the one thing here that needs it: without Quarto the maps, tables and GIS layers are still written, and a message says the reports were skipped. Costs about 15 seconds per format.

Value

The manifest, invisibly: a tibble of path, kind and description, one row per file written, with paths relative to dir.

See Also

[connectivity_report_data\(\)](#), [zip_connectivity_assets\(\)](#), and [generate_connectivity_report\(\)](#) for one report on its own.

Examples

```
report_data <- connectivity_report_data(
  habitat = example_habitat(),
  barrier = example_barrier(),
  species = "Blue Tongue Lizard",
  interpatch_distance = 20,
  verbose = FALSE
)

assets <- write_connectivity_assets(
  report_data,
  dir = tempfile(),
  reports = FALSE
)
assets$path
```

```
write_connectivity_report
```

Write a connectivity report's source, so you can render it yourself

Description

Writes the report as a Quarto document you own a copy of, next to the data it draws. Open it, change it, render it. [render_connectivity_report\(\)](#) renders it the way the package does, and [generate_connectivity_report\(\)](#) does both steps at once when the source isn't wanted.

Usage

```
write_connectivity_report(x, path = NULL)
```

Arguments

x	A connectivity_report_data object from connectivity_report_data() .
path	File to write, ending in .qmd. Defaults to the species and today's date, in the working directory. The directory must already exist.

Details

Two files are written: path, and the analysis beside it as <name>-data.rds. The document reads that file by name, so the two travel together. Moving one without the other breaks the render.

Value

The absolute path of the .qmd, invisibly.

See Also

[render_connectivity_report\(\)](#) to render it, [generate_connectivity_report\(\)](#) to do both.

Examples

```
report_data <- connectivity_report_data(
  habitat = example_habitat(),
  barrier = example_barrier(),
  species = "Blue Tongue Lizard",
  interpatch_distance = 20,
  verbose = FALSE
)

qmd <- write_connectivity_report(
  report_data,
  file.path(tempdir(), "lizard-report.qmd")
)

readLines(qmd, n = 5)
```

write_report_data	<i>Save and reload a connectivity analysis</i>
-------------------	--

Description

Writes a connectivity_report_data object to an .rds file, and reads it back. The summary, the habitat and barrier layers, and the patch raster for each distance all travel together in the one file.

Usage

```
write_report_data(x, path)

read_report_data(path)
```

Arguments

x	A connectivity_report_data object from connectivity_report_data() .
path	File to write to, or read from.

Details

`base::saveRDS()` on its own isn't enough. A `SpatRaster` points at memory or at a file on disk rather than carrying its own values, so it saves as a null pointer and comes back unusable. `terra::wrap()` packs the values into the object on the way out, and `terra::unwrap()` restores them on the way in.

This is also how an analysis reaches the separate R session that Quarto renders the report in.

Value

`write_report_data()` returns path invisibly; `read_report_data()` returns the `connectivity_report_data`.

See Also

`connectivity_report_data()`

Examples

```
report_data <- connectivity_report_data(
  habitat = example_habitat(),
  barrier = example_barrier(),
  species = "Blue Tongue Lizard",
  interpatch_distance = 20,
  verbose = FALSE
)

path <- write_report_data(report_data, tempfile(fileext = ".rds"))
read_report_data(path)$connectivity
```

zip_connectivity_assets

Write the downloadable assets as a single zip

Description

`write_connectivity_assets()` into a folder named for the species and the date, archived. This is what the Shiny app's download button returns.

Usage

```
zip_connectivity_assets(x, path, reports = quarto_available())
```

Arguments

<code>x</code>	A <code>connectivity_report_data</code> object from <code>connectivity_report_data()</code> .
<code>path</code>	Path to write the .zip to.
<code>reports</code>	Also render the reports into the archive. See <code>write_connectivity_assets()</code> .

Value

The absolute path written, invisibly.

See Also

[write_connectivity_assets\(\)](#) to write the files without archiving, and [zip_connectivity_reports\(\)](#) for the reports alone.

Examples

```
report_data <- connectivity_report_data(
  habitat = example_habitat(),
  barrier = example_barrier(),
  species = "Blue Tongue Lizard",
  interpatch_distance = 20,
  verbose = FALSE
)

zip_path <- zip_connectivity_assets(
  report_data,
  tempfile(fileext = ".zip"),
  reports = FALSE
)
zip::zip_list(zip_path)$filename
```

zip_connectivity_reports

Write the reports as a single zip

Description

The HTML and PDF reports, in a folder named for the species and the date, archived. This is the app's reports-only download, for someone who wants the write-up without the GIS layers. The analysis is packed once and rendered twice, so this costs less than two separate calls.

Usage

```
zip_connectivity_reports(x, path)
```

Arguments

x	A connectivity_report_data object from connectivity_report_data() .
path	Path to write the .zip to.

Value

The absolute path written, invisibly.

See Also

[zip_connectivity_assets\(\)](#) for everything, and [generate_connectivity_report\(\)](#) for one report on its own.

Examples

```
report_data <- connectivity_report_data(  
  habitat = example_habitat(),  
  barrier = example_barrier(),  
  species = "Blue Tongue Lizard",  
  interpatch_distance = 20,  
  verbose = FALSE  
)  
  
# rendering needs the Quarto command line tool  
if (quarto_available()) {  
  zip_path <- zip_connectivity_reports(  
    report_data,  
    tempfile(fileext = ".zip")  
  )  
  zip::zip_list(zip_path)$filename  
}
```

Index

- * **datasets**
 - example-lizard-data, [18](#)
 - example-wren-data, [19](#)
 - lizard_areas_connected, [33](#)
- add_patch_area, [3](#)
- aggregate_connected_patches, [4](#)
- aggregate_connected_patches(), [27](#)
- assign_patches_to_fragments, [5](#)
- assign_patches_to_fragments(), [27](#)
- base::saveRDS(), [58](#)
- clean, [6](#)
- col2hex, [6](#)
- compare_connectivity, [7](#)
- compare_connectivity(), [8](#), [9](#), [28](#), [29](#)
- compare_scenarios, [8](#)
- compare_scenarios(), [31](#), [32](#)
- connectivity_display, [10](#)
- connectivity_display(), [15](#), [16](#), [45](#)
- connectivity_file_stem, [11](#)
- connectivity_probability, [12](#)
- connectivity_report_data, [13](#)
- connectivity_report_data(), [11](#), [22](#), [46](#), [55–59](#)
- create_barrier_mask, [15](#)
- create_barrier_mask(), [21](#), [27](#)
- display_ids, [15](#)
- drop_habitat_under_barrier, [16](#)
- drop_habitat_under_barrier(), [27](#)
- DT::datatable(), [10](#), [35](#), [44](#)
- effective_mesh_size, [17](#)
- effective_mesh_size(), [12](#)
- empty_grid, [17](#)
- empty_grid(), [36](#)
- example-lizard-data, [18](#)
- example-wren-data, [19](#)
- example_barrier(example-lizard-data), [18](#)
- example_barrier(), [33](#)
- example_barrier_scenario
 - (example-lizard-data), [18](#)
- example_barrier_shp
 - (example-lizard-data), [18](#)
- example_habitat(example-lizard-data), [18](#)
- example_habitat(), [33](#)
- example_wren_barrier
 - (example-wren-data), [19](#)
- example_wren_barrier_scenario
 - (example-wren-data), [19](#)
- example_wren_habitat
 - (example-wren-data), [19](#)
- example_wren_habitat_scenario
 - (example-wren-data), [19](#)
- format_by(round_by), [44](#)
- format_by(), [44](#), [45](#)
- format_resolution, [20](#)
- fragment_habitat, [21](#)
- fragment_habitat(), [27](#)
- generate_connectivity_report, [22](#)
- generate_connectivity_report(), [11](#), [42](#), [44](#), [56](#), [57](#), [60](#)
- gg_barrier_habitat_interpatch_dist, [23](#)
- gg_barrier_habitat_interpatch_dist(), [25](#), [38](#)
- gg_layer, [25](#)
- gg_layer(), [46](#)
- ggplot2::ggsave(), [22](#), [44](#), [55](#)
- habitat_buffer, [26](#)
- habitat_buffer(), [27](#), [50](#), [51](#)
- habitat_connectivity, [27](#)
- habitat_connectivity(), [7](#), [8](#), [13](#), [14](#), [17](#), [28–30](#), [33–35](#), [37](#), [52](#), [53](#)

habitat_connectivity_comparison, 28
 habitat_connectivity_comparison(), 7–9, 14, 32, 36
 habitat_connectivity_full, 30
 habitat_connectivity_scenarios, 31

 knitr::kable(), 10, 44

 lizard_areas_connected, 33

 mean_patch_size, 34

 n_patches, 34
 new_patch_size_tbl, 35

 onto_grid, 36
 onto_grid(), 13

 patch_size_tbl(new_patch_size_tbl), 35
 patch_size_tbl(), 37, 52
 patch_sizes, 37
 patch_sizes(), 27, 33, 35, 53
 pc-getters, 37
 pc_interpatch_distance(pc-getters), 37
 pc_interpatch_distance(), 35
 pc_patches(pc-getters), 37
 pc_res(pc-getters), 37
 pc_species(pc-getters), 37
 pc_species(), 35
 plot_barrier_habitat_interpatch_dist, 38
 plot_connectivity, 39
 plot_patches, 40
 prepare_rasters, 41
 prepare_rasters(), 36

 quarto::quarto_render(), 43
 quarto_available, 42

 read_geometry, 43
 read_report_data(write_report_data), 57
 render_connectivity_report, 43
 render_connectivity_report(), 22, 56, 57
 round_by, 44
 round_by(), 10, 44, 45
 run_connectivity_app, 45

 scenario_layer, 46
 scenario_layer(), 14
 sf::st_buffer(), 50
 sf_add_patch_area, 47
 sf_aggregate_connected_patches, 47
 sf_assign_patches_to_fragments, 48
 sf_drop_habitat_under_barrier, 49
 sf_fragment_habitat, 50
 sf_habitat_buffer, 50
 sf_habitat_connectivity, 51
 summarise-connectivity, 52
 summarise_connectivity
 (summarise-connectivity), 52
 summarise_connectivity(), 7, 8, 33, 37

 terra::mask(), 16
 terra::res(), 21
 terra::unwrap(), 58
 terra::wrap(), 58
 tibble, 35
 total_habitat_area, 53

 urbio_colours, 54
 urbio_colours(), 24, 25, 39
 urbio_figure_size, 55
 urbio_figure_size(), 38
 utils::write.csv(), 35

 write_connectivity_assets, 55
 write_connectivity_assets(), 22, 42, 58, 59
 write_connectivity_report, 56
 write_connectivity_report(), 11, 22, 43, 44
 write_report_data, 57

 zip_connectivity_assets, 58
 zip_connectivity_assets(), 11, 22, 55, 56, 60
 zip_connectivity_reports, 59
 zip_connectivity_reports(), 59