Package 'hydroloom'

May 23, 2026

Title Utilities to Weave Hydrologic Fabrics

Version 1.2.0

Description A collection of utilities that support creation of network attributes for hydrologic networks. Methods and algorithms implemented are documented in Moore et al. (2019) <doi:10.3133/ofr20191096>), Cormen and Leiserson (2022) <ISBN:9780262046305> and Verdin and Verdin (1999) <doi:10.1016/S0022-1694(99)00011-6>.

Depends R (>= 4.1.0)

Imports dplyr, data.table, sf, units, stats, methods, utils, pbapply, tidyr, RANN, rlang, fastmap

Suggests testthat, nhdplusTools, future, lwgeom, future.apply, knitr, gifski, mapview, webshot, geos

License CC0

Encoding UTF-8

RoxygenNote 7.3.3

Config/testthat/edition 3

VignetteBuilder knitr

Language en-US

URL https://github.com/DOI-USGS/hydroloom, https://doi-usgs.github.io/hydroloom/

NeedsCompilation no

Author David Blodgett [aut, cre] (ORCID: <https://orcid.org/0000-0001-9489-1710>), Andrew Psoras [ctb]

Maintainer David Blodgett <dblodgett@usgs.gov>

Repository CRAN

Date/Publication 2026-05-23 11:20:12 UTC

Contents

`accumulate_downstream` *Accumulate Variable Downstream*

Description

given a variable, accumulate according to network topology. See details for required attributes and additional information.

Usage

```
accumulate_downstream(x, var, total = FALSE, quiet = FALSE)

## S3 method for class 'data.frame'
accumulate_downstream(x, var, total = FALSE, quiet = FALSE)

## S3 method for class 'hy'
accumulate_downstream(x, var, total = FALSE, quiet = FALSE)

## S3 method for class 'hy_node'
accumulate_downstream(x, var, total = FALSE, quiet = FALSE)

## S3 method for class 'hy_flownetwork'
accumulate_downstream(x, var, total = FALSE, quiet = FALSE)

## S3 method for class 'hy_topo'
accumulate_downstream(x, var, total = FALSE, quiet = FALSE)
```

Arguments

`x`	data.frame network compatible with hydroloom_names.
`var`	variable to accumulate.
`total`	logical if TRUE, accumulation will use "total" apportionment if FALSE, divergence or dendritic apportionment will apply (see details).
`quiet`	logical quiet messages?

Details

Required attributes: `id` and `toid` or `fromnode`, `tonode`, and `divergence`

Conditionally: `divergence_fraction` (if divergence apportioned routing is desired).

Accumulation Methods:

Divergence apportioned (divergence routing): Where upstream values are passed with fractional apportionment such that each downstream connection gets between 0 and 100 percent of the upstream value. Requires a "divergence_fraction" attribute and the "total" parameter to be FALSE.

Dendritic apportionment (no divergence routing): Where upstream values are not passed to secondary paths at all – this is essentially a special case of divergence apportioned where no diversion fraction value is provided and 0 is assumed for all divergences. Do not include a "divergence_fraction" and set "total" to FALSE.

No apportionment (total upstream): where upstream values are passed without being apportioned such that each downstream connection gets the full upstream value and there is special handling where diversions join back to the main flow to avoid double counting. This is also referred to as "total upstream routing". Set "total" to TRUE.

"No apportionment" (total upstream) routing includes considerably more logic and requires a notable amount more computation to avoid double counting through systems of diverted channels. The implementation has been tested to match the total drainage area calculations of NHDPlusV2.

When flow splits at a diversion, the duplicated part is tracked until it recombines with the non-duplicated part. In this tracking, both nested diversions and diversions that have two or more flow splits in one place are supported. For this algorithm to work, it is critical that the supplied data be a directed acyclic graph and have a complete divergence attribute where 0 indicates no diversion, 1 indicates the main catchment downstream of a diversion and 2 indicates a secondary (one or more) downstream of a diversion.

Value

vector of the same length as nrow(x) containing values of var accumulated downstream

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

net <- navigate_network_dfs(x, 8893236, "up")

x <- x[x$COMID %in% unlist(net), ]

# All default gives dendritic routing
x$dend_totdasqkm <- accumulate_downstream(add_toids(x), "AreaSqKM")
x$diff <- x$TotDASqKM - x$dend_totdasqkm

# notice that diversions reset as if they were headwaters
plot(x['dend_totdasqkm'], lwd = x$dend_totdasqkm / 20)

# add a diversion_fraction that splits flow evenly
# max(dplyr::n()) is the number of flowlines in a FromNode group.
y <- x |>
  dplyr::group_by(FromNode) |>
  dplyr::mutate(divergence_fraction = 1 / max(dplyr::n())) |>
  dplyr::ungroup()

y$div_totdasqkm <- accumulate_downstream(y, "AreaSqKM")

# notice that diversions don't reset -- they carry a fraction of area
```

```
plot(y['div_totdasqkm'], lwd = y$div_totdasqkm / 20)

# total not implemented yet, but will be soon
z <- x |>
  dplyr::select(COMID, FromNode, ToNode, Divergence, AreaSqKM, TotDASqKM)

z$tot_totdasqkm <- accumulate_downstream(z, "AreaSqKM", total = TRUE)

plot(z['tot_totdasqkm'], lwd = z$tot_totdasqkm / 20)

# equivalent values from the nhdplusv2 match!

any(abs(z$tot_totdasqkm - z$TotDASqKM) > 0.001)
```

`add_divergence` *Add Divergence Attribute*

Description

Given a non-dendritic flow network and required attributes, adds a divergence attribute according to NHDPlus data model methods.

Usage

```
add_divergence(
  x,
  coastal_outlet_ids,
  inland_outlet_ids,
  name_attr,
  type_attr,
  major_types
)

## S3 method for class 'data.frame'
add_divergence(
  x,
  coastal_outlet_ids,
  inland_outlet_ids,
  name_attr,
  type_attr,
  major_types
)

## S3 method for class 'hy'
add_divergence(
  x,
  coastal_outlet_ids,
```

```
  inland_outlet_ids,
  name_attr,
  type_attr,
  major_types
)

## S3 method for class 'hy_topo'
add_divergence(
  x,
  coastal_outlet_ids,
  inland_outlet_ids,
  name_attr,
  type_attr,
  major_types
)

## S3 method for class 'hy_node'
add_divergence(
  x,
  coastal_outlet_ids,
  inland_outlet_ids,
  name_attr,
  type_attr,
  major_types
)
```

Arguments

`x`	data.frame network compatible with hydroloom_names.
`coastal_outlet_ids`	vector of identifiers for network outlets that terminate at the coast.
`inland_outlet_ids`	vector of identifiers for network outlets that terminate inland.
`name_attr`	character attribute name of attribute containing a feature name or name identifier.
`type_attr`	character attribute name of attribute containing a feature type indicator.
`major_types`	vector of values of `type_attr` that should be interpreted as being "major". e.g. river might be major and canal might be minor.

Details

Required attributes: `id`, `fromnode`, `tonode`

When considering downstream connections with diversions, there are three factors considered to determine which is primary. 1a) same name 1b) is named 2) feature type (type_attr controls this) 3) flows to coast (has a coastal connection is preferred)

The following list describes the order of precedence for tests 1: 1a, 2, 3 2: 1a, 2 3: The NHDPlus uses diverted fraction this is not used currently. 4: 1b, 2, 3 5: 2, 3 6: 1b, 3 7: 3, 8: 1b, 2 9: 2 10: 1b

If all checks return and no primary connection has been identified, the connection with a smaller id is chosen.

In the case that there are two or more upstream connections, the upstream name to use is chosen 1) if there is only one upstream flowline with a name 2) if one of the upstream flowlines with a name matches the downstream line, 3) if one of the upstream flowlines is of a "major" type and others are not, and, 4) if no criteria exist to select one, the smallest id value otherwise.

Value

returns x with a `divergence` attribute appended

Examples

```
f <- system.file("extdata/coastal_example.gpkg", package = "hydroloom")

g <- sf::read_sf(f)
g <- g[g$FTYPE != "Coastline", ]

outlets <- g$COMID[!g$ToNode %in% g$FromNode]

g <- dplyr::select(g, COMID, gnis_id, FTYPE,
  FromNode, ToNode)

add_divergence(g,
  coastal_outlet_ids = outlets,
  inland_outlet_ids = c(),
  name_attr = "gnis_id",
  type_attr = "FTYPE",
  major_types = c("StreamRiver", "ArtificialPath", "Connector"))
```

`add_levelpaths` *Add Level Paths*

Description

Assigns level paths using the stream-leveling approach of NHD and NHDPlus. If arbolate sum is provided in the weight column, this will match the behavior of NHDPlus. Any numeric value can be included in this column and the largest value will be followed when no nameid is available.

x must include id, toid, and conditionally divergence attributes. If a "topo_sort" (hydrosequence in nhdplus terms) attribute is included, it will be used instead of recreation.

Usage

```
add_levelpaths(
  x,
  name_attribute,
  weight_attribute,
```

```
  override_factor = NULL,
  status = FALSE
)

## S3 method for class 'data.frame'
add_levelpaths(
  x,
  name_attribute,
  weight_attribute,
  override_factor = NULL,
  status = FALSE
)

## S3 method for class 'hy'
add_levelpaths(
  x,
  name_attribute,
  weight_attribute,
  override_factor = NULL,
  status = FALSE
)

## S3 method for class 'hy_node'
add_levelpaths(
  x,
  name_attribute,
  weight_attribute,
  override_factor = NULL,
  status = FALSE
)

## S3 method for class 'hy_flownetwork'
add_levelpaths(
  x,
  name_attribute,
  weight_attribute,
  override_factor = NULL,
  status = FALSE
)

## S3 method for class 'hy_topo'
add_levelpaths(
  x,
  name_attribute,
  weight_attribute,
  override_factor = NULL,
  status = FALSE
)
```

Arguments

x	data.frame network compatible with hydroloom_names.
name_attribute	character attribute to be used as name identifiers.
weight_attribute	character attribute to be used as weight.
override_factor	numeric multiplier to use to override name_attribute. See details.
status	boolean if status updates should be printed.

Details

The levelpath algorithm defines upstream mainstem paths through a network. At a given junction with two or more upstream flowlines, the main path is either 1) the path with the same name, 2) the path with any name, 3) or the path with the larger weight. If the weight_attribute is override_factor times larger on a path, it will be followed regardless of the name_attribute indication.

If id and toid are non-dendritic so id:toid is many to one and id is non-unique, a divergence attribute must be included such that the dendritic network can be extracted after the network is sorted.

Value

data.frame with id, levelpath_outlet_id, topo_sort, and levelpath columns. See details for more info.

See Also

hy_leveled, hy_topo, add_pfafstetter(), to_flownetwork()

Examples

```
g <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

test_flowline <- add_toids(g)

# use NHDPlus attributes directly
add_levelpaths(test_flowline,
  name_attribute = "GNIS_ID",
  weight_attribute = "ArbolateSu")

# use hy attributes where they can be mapped
add_levelpaths(hy(test_flowline),
  name_attribute = "GNIS_ID",
  weight_attribute = "arbolate_sum")
```

add_measures *Add aggregate id measures to flowlines*

Description

given a set of connected flowlines that have ids and aggregate ids, adds from_aggregate_id_measure and to_aggregate_id_measure for use with index_points_to_lines

Aggregate ids, such as mainstem ids or reachcodes span multiple flowlines. Linear referencing along these features requires knowledge of the portion of the aggregate line a given flowline makes up. This function assumes that the complete aggregate feature is included and calculates the measure of the top and bottom of each flowline along each aggregate line.

Usage

```
add_measures(x)

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

## S3 method for class 'hy'
add_measures(x)
```

Arguments

x	sf data.frame compatible with hydroloom_names with at least id and aggregate_id attributes. A pre-populated toid attribute will be used if present.

Details

If no "toid" attribute is included, make_attribute_topology is used to to create one. This is required to ensure the flowlines making up each aggregate line are sorted in a known upstream to downstream order.

This function assumes that all flowlines that make up an aggregate feature are included. Returned measures will be incomplete and incorrect if aggregate features (mainstems of reaches) are truncated.

Value

x with aggregate measures added to it

Examples

```
g <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

d <- dplyr::select(g, COMID, REACHCODE) |>
  sf::st_cast("LINESTRING")

add_measures(d)
```

add_pathlength *Add Path Length*

Description

Generates the main path length to a basin's terminal path.

Usage

```
add_pathlength(x)

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

## S3 method for class 'hy'
add_pathlength(x)

## S3 method for class 'hy_node'
add_pathlength(x)

## S3 method for class 'hy_topo'
add_pathlength(x)
```

Arguments

x	data.frame network compatible with hydroloom_names.

Details

Required attributes: `id`, `toid`, `length_km`

Value

data.frame containing pathlength_km

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

x <- add_toids(x)

x <- add_pathlength(x)

plot(x["Pathlength"])
```

`add_pfafstetter` *Add Pfafstetter Codes*

Description

Determines Pfafstetter codes for a dendritic network. Topo_sort and levelpath attributes must be self consistent (levelpath values are the same as the outlet topo_sort value) as generated by add_levelpaths.

Usage

```
add_pfafstetter(x, max_level = 2, status = FALSE)

## S3 method for class 'data.frame'
add_pfafstetter(x, max_level = 2, status = FALSE)

## S3 method for class 'hy'
add_pfafstetter(x, max_level = 2, status = FALSE)

## S3 method for class 'hy_topo'
add_pfafstetter(x, max_level = 2, status = FALSE)

## S3 method for class 'hy_node'
add_pfafstetter(x, max_level = 2, status = FALSE)

## S3 method for class 'hy_leveled'
add_pfafstetter(x, max_level = 2, status = FALSE)
```

Arguments

`x`	data.frame network compatible with hydroloom_names.
`max_level`	integer number of levels to attempt to calculate. If the network doesn't have resolution to support the desired level, unexpected behavior may occur.
`status`	boolean if status updates should be printed.

Details

Required attributes: `id`, `toid`, `total_da_sqkm`, `topo_sort`, `levelpath`

Value

data.frame with added pfafstetter column

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

x <- add_toids(x)

pfaf <- add_pfafstetter(x, max_level = 2)

plot(pfaf["pf_level_2"], lwd = 2)


if (require(nhdplusTools)) {

  # uses tempdir for example
  work_dir <- nhdplusTools::nhdplusTools_data_dir(tempdir())

  try(
    source(system.file("extdata/nhdplushr_data.R", package = "nhdplusTools"))
  )
  if (exists("hr_data")) {
    x <- hy(hr_data$NHDFlowline)

    x <- add_toids(x)

    x <- dplyr::select(x, id, toid, da_sqkm)

    #' add terminal_id -- add in function?
    x <- sort_network(x, split = TRUE)

    x$total_da_sqkm <- accumulate_downstream(x, "da_sqkm")
    x$name <- ""

    x <- add_levelpaths(x, name_attribute = "name", weight_attribute = "total_da_sqkm")

    x <- add_pfafstetter(x, max_level = 3)

    plot(x["pf_level_3"], lwd = 2)

    pfaf <- add_pfafstetter(x, max_level = 4)

    hr_catchment <- dplyr::left_join(hr_data$NHDPlusCatchment,
      sf::st_drop_geometry(pfaf), by = c("FEATUREID" = "id"))

    colors <- data.frame(pf_level_4 = unique(hr_catchment$pf_level_4),
      color = sample(terrain.colors(length(unique(hr_catchment$pf_level_4)))))

    hr_catchment <- dplyr::left_join(hr_catchment, colors, by = "pf_level_4")

    plot(hr_catchment["color"], border = NA, reset = FALSE)
    plot(sf::st_geometry(x), col = "blue", add = TRUE)
  } else {
    message("nhdplusTools > 1.0 required for this example")
  }
```

```
}
```

`add_return_divergence` *Add Return Divergence*

Description

Adds a return divergence attribute to the provided network. The method implemented matches that of the NHDPlus except in the rare case that a diversion includes more than one secondary path.

See add_divergence and make_node_topology.

Usage

```
add_return_divergence(x, status = TRUE)

## S3 method for class 'data.frame'
add_return_divergence(x, status = TRUE)

## S3 method for class 'hy'
add_return_divergence(x, status = TRUE)

## S3 method for class 'hy_topo'
add_return_divergence(x, status = TRUE)

## S3 method for class 'hy_node'
add_return_divergence(x, status = TRUE)
```

Arguments

`x`	data.frame network compatible with hydroloom_names.
`status`	boolean if status updates should be printed.

Details

Required attributes: `id`, `fromnode`, `tonode`, `divergence`

Algorithm:

All network connections with more than one downstream feature are considered.

navigate_network_dfs is used to find all downstream features emanating from the primary (`divergence == 1`) outlet of the diversion in question and secondary (`divergence == 2`) outlet(s) starting with the primary outlet.

navigate_network_dfs is called with `reset = FALSE` such that the secondary diversion paths terminate where they combine with a previously visited feature.

If the diverted paths result in only one outlet, the feature it flows to is marked as a return divergence.

If the diverted paths result in more than one outlet, the one that flows to the most upstream feature in the set of features downstream of the primary outlet of the diversion is marked as the return divergence.

Value

data.frame containing `return_divergence` attribute

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

x <- hy(x)

x <- add_return_divergence(x)

sum(x$return_divergence == x$RtnDiv)

# see description for documentation of one that does not match
```

`add_streamlevel` *Add Streamlevel*

Description

Applies a topological sort and calculates stream level. Algorithm: Terminal level paths are assigned level 1 (see note 1). Paths that terminate at a level 1 are assigned level 2. This pattern is repeated until no paths remain.

If a TRUE/FALSE coastal attribute is included, coastal terminal paths begin at 1 and internal terminal paths begin at 4 as is implemented by the NHD stream leveling rules.

Usage

```
add_streamlevel(x, coastal = NULL)

## S3 method for class 'data.frame'
add_streamlevel(x, coastal = NULL)

## S3 method for class 'hy'
add_streamlevel(x, coastal = NULL)

## S3 method for class 'hy_topo'
add_streamlevel(x, coastal = NULL)

## S3 method for class 'hy_leveled'
add_streamlevel(x, coastal = NULL)
```

Arguments

`x`	data.frame network compatible with hydroloom_names.
`coastal`	character attribute name containing a logical flag indicating if a given terminal catchment flows to the coast of is an inland sink. If no coastal flag is included, all terminal paths are assumed to be coastal.

Details

Required attributes: levelpath, dn_levelpath

Value

data,frame containing added stream_level attribute

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

x <- add_toids(x)

x <- dplyr::rename(x, orig_stream_level = StreamLeve)

y <- add_streamlevel(x)

plot(sf::st_geometry(y), lwd = y$stream_level, col = "blue")

x$coastal <- rep(FALSE, nrow(x))

y <- add_streamlevel(x, coastal = "coastal")

unique(y$stream_level)

x$coastal[!x$Hydroseq == min(x$Hydroseq)] <- TRUE

y <- add_streamlevel(x)

unique(y$stream_level)
```

---

add_streamorder *Add Streamorder*

Description

Adds a strahler stream order.

Algorithm: If more than one upstream flowline has an order equal to the maximum upstream order then the downstream flowline is assigned the maximum upstream order plus one. Otherwise it is assigned the maximum upstream order.

To match the NHDPlus algorithm, non-dendritic network connectivity must be included. All secondary paths will have the stream_order of upstream primary paths and a stream_calculator value of 0. Secondary paths have no affect on the order of downstream paths.

Usage

```
add_streamorder(x, status = TRUE)

## S3 method for class 'data.frame'
add_streamorder(x, status = TRUE)

## S3 method for class 'hy'
add_streamorder(x, status = TRUE)

## S3 method for class 'hy_node'
add_streamorder(x, status = TRUE)

## S3 method for class 'hy_flownetwork'
add_streamorder(x, status = TRUE)

## S3 method for class 'hy_topo'
add_streamorder(x, status = TRUE)
```

Arguments

x	data.frame network compatible with hydroloom_names.
status	boolean if status updates should be printed.

Details

Required attributes: `id` and `toid` or `fromnode`, `tonode`, and `divergence`

Value

data.frame containing added `stream_order` and `stream_calculator` attribute.

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

x <- dplyr::select(x, COMID, FromNode, ToNode, Divergence)

x <- add_streamorder(x)

plot(sf::st_geometry(x), lwd = x$stream_order, col = "blue")
plot(sf::st_geometry(x), lwd = x$stream_calculator, col = "blue")
```

add_toids *Add Downstream IDs*

Description

Generates a toid attribute from node topology by joining tonode and fromnode attributes.

Usage

```
add_toids(x, return_dendritic = TRUE)

## S3 method for class 'data.frame'
add_toids(x, return_dendritic = TRUE)

## S3 method for class 'hy'
add_toids(x, return_dendritic = TRUE)

## S3 method for class 'hy_topo'
add_toids(x, return_dendritic = TRUE)

## S3 method for class 'hy_node'
add_toids(x, return_dendritic = TRUE)
```

Arguments

x	data.frame network compatible with hydroloom_names.
return_dendritic	logical remove non dendritic paths if TRUE. Requires a "divergence" flag where 1 is main and 2 is secondary.

Details

Required attributes: fromnode, tonode

Conditionally: divergence (if return_dendritic = TRUE)

Value

hy object with toid attribute

See Also

hy_node, hy_topo, make_node_topology()

Examples

```
g <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

x <- add_toids(hy(g))

y <- add_toids(g)

names(g)[1:4]

names(x)[1:4]

names(y)[1:4]
```

`add_topo_sort` *Add topo_sort*

Description

calls sort_network without support for splitting the network and adds a `nrow:1` topo_sort attribute.

Usage

```
add_topo_sort(x, outlets = NULL)

## S3 method for class 'data.frame'
add_topo_sort(x, outlets = NULL)

## S3 method for class 'hy'
add_topo_sort(x, outlets = NULL)

## S3 method for class 'hy_node'
add_topo_sort(x, outlets = NULL)

## S3 method for class 'hy_flownetwork'
add_topo_sort(x, outlets = NULL)

## S3 method for class 'hy_topo'
add_topo_sort(x, outlets = NULL)
```

Arguments

`x`	data.frame network compatible with hydroloom_names.
`outlets`	same as id in `x`. if specified, only the network emanating from these outlets will be considered and returned.

Details

Required attributes: `id`, `toid`

Value

data.frame containing a topo_sort attribute.

`align_names` *Align Names to Hydroloom Convention*

Description

this function aligns the attribute names in x with those used in hydroloom. See hydroloom_names for how to add more attribute name mappings if the attributes in your data are not supported.

See hydroloom_name_definitions for definitions of the names used in hydroloom.

Usage

```
align_names(x)
```

Arguments

`x`	data.frame network compatible with hydroloom_names.

Value

data.frame renamed to match hydroloom as possible.

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

names(x)

x <- align_names(x)

names(x)
```

check_hy_graph *Check hy Graph*

Description

check that a network graph doesn't contain localized loops.

Usage

```
check_hy_graph(x, loop_check = FALSE)
```

Arguments

x	data.frame network compatible with hydroloom_names.
loop_check	logical if TRUE, the entire network is walked from top to bottom searching for loops. This loop detection algorithm visits a node in the network only once all its upstream neighbors have been visited. A complete depth first search is performed at each node, searching for paths that lead to an already visited (upstream) node. This algorithm is often referred to as "recursive depth first search".

Details

Required attributes: id, toid

Value

if no localized loops are found, returns TRUE. If localized loops are found, problem rows with a row number added.

Examples

```
# notice that row 4 (id = 4, toid = 9) and row 8 (id = 9, toid = 4) is a loop.
test_data <- data.frame(id = c(1, 2, 3, 4, 6, 7, 8, 9),
  toid = c(2, 3, 4, 9, 7, 8, 9, 4))
check_hy_graph(test_data)
```

check_valid *Check and Repair Geometry Validity*

Description

Validates sf geometry, repairs invalid features, unifies geometry types, and optionally reprojects. Handles GEOMETRYCOLLECTION artifacts from st_make_valid() by extracting the dominant geometry type.

Usage

```
check_valid(x, ...)

## S3 method for class 'sf'
check_valid(x, out_prj = sf::st_crs(x), ...)

## S3 method for class 'sfc'
check_valid(x, out_prj = sf::st_crs(x), ...)
```

Arguments

x	sf data.frame, sfc geometry, or NULL.
...	additional arguments passed to methods.
out_prj	crs. Target CRS for the output, passed to st_transform. Defaults to the CRS of x.

Value

Same class as input: sf data.frame, sfc, or NULL.

Examples

```
library(sf)
p1 <- st_polygon(list(
  rbind(c(0, 0), c(10, 0), c(10, 10), c(0, 10), c(0, 0)),
  rbind(c(2, 2), c(2, 8), c(8, 8), c(8, 2), c(2, 2))
))
x <- st_sf(geometry = st_sfc(p1, crs = 5070))
result <- check_valid(x)
```

---

disambiguate_indexes *Disambiguate Flowline Indexes*

Description

Given a set of flowline indexes and numeric or ascii criteria, return closest match. If numeric criteria are used, the minimum difference in the numeric attribute is used for disambiguation. If ascii criteria are used, the adist function is used with the following algorithm: `1 - adist_score / max_string_length`. Comparisons ignore case.

Usage

```
disambiguate_indexes(indexes, flowpath, hydro_location)
```

Arguments

indexes	data.frame as output from index_points_to_lines with more than one hydrologic location per indexed point.
flowpath	data.frame with two columns. The first should join to the id field of the indexes and the second should be the numeric or ascii metric such as drainage area or Name. Names of this data.frame are not used.
hydro_location	data.frame with two columns. The first should join to the id field of the indexes and the second should be the numeric or ascii metric such as drainage area or GNIS Name. Names of this data.frame are not used.

Value

data.frame indexes deduplicated according to the minimum difference between the values in the metric columns. If two or more result in the same "minimum" value, duplicates will be returned.

Examples

```
if (require(nhdplusTools)) {
  source(system.file("extdata", "sample_flines.R", package = "nhdplusTools"))

  hydro_location <- sf::st_sf(id = c(1, 2, 3),
    geom = sf::st_sfc(list(sf::st_point(c(-76.86934, 39.49328)),
      sf::st_point(c(-76.91711, 39.40884)),
      sf::st_point(c(-76.88081, 39.36354))),
    crs = 4326),
    totda = c(23.6, 7.3, 427.9),
    nameid = c("Patapsco", "", "Falls Run River"))

  indexes <- index_points_to_lines(sample_flines,
    hydro_location,
    search_radius = units::set_units(0.2, "degrees"),
    max_matches = 10)

  disambiguate_indexes(indexes,
    dplyr::select(sample_flines, COMID, TotDASqKM),
    dplyr::select(hydro_location, id, totda))

  result <- disambiguate_indexes(indexes,
    dplyr::select(sample_flines, COMID, GNIS_NAME),
    dplyr::select(hydro_location, id, nameid))

  result[result$point_id == 1, ]

  result[result$point_id == 2, ]

  result[result$point_id == 3, ]
}
```

`dissolve_polygons` *Dissolve Polygons*

Description

Dissolves (unions) a set of polygons, optionally by group, fills interior holes below a configurable area threshold, and returns clean, valid polygon geometry. Designed for merging catchments, HUC boundaries, or other polygon coverages where sliver gaps and pinhole holes are artifacts rather than real features.

Usage

```
dissolve_polygons(polys, ...)

## S3 method for class 'sf'
dissolve_polygons(
  polys,
  group_id = NULL,
  .fns = NULL,
  max_hole_area = Inf,
  gap_tolerance = 0,
  single_polygon = FALSE,
  work_crs = 5070,
  ...
)

## S3 method for class 'sfc'
dissolve_polygons(
  polys,
  max_hole_area = Inf,
  gap_tolerance = 0,
  single_polygon = FALSE,
  work_crs = 5070,
  ...,
  groups = NULL
)
```

Arguments

`polys`	sf data.frame or sfc geometry. Input polygons with POLYGON or MULTIPOLYGON geometry. When sfc is provided, `group_id` is ignored and the return value is sfc.
`...`	additional arguments passed to methods.
`group_id`	character or NULL. Column name to dissolve by. When NULL, all polygons are dissolved into a single geometry. Ignored for sfc input. Default NULL.

.fns	named list of summary functions, or NULL. When non-NULL, each element name must match a column in polys and the corresponding function is applied to that column within each group (or across all rows when group_id is NULL). Requires group_id when more than one group is desired. Example: list(AreaSqKM = sum). Default NULL (geometry only).
max_hole_area	numeric. Maximum hole area (in square meters when using a projected CRS) below which holes are removed. Use 0 to keep all holes, or Inf to remove all holes. Default Inf.
gap_tolerance	numeric. Buffer distance (in CRS units, typically meters) for the expand-contract step that absorbs sliver gaps between input polygons. Use 0 to skip. Default 0.
single_polygon	logical. If TRUE, extract the largest polygon from each MULTIPOLYGON result, guaranteeing single POLYGON output per row. Default FALSE.
work_crs	anything accepted by st_crs, or NULL. When non-NULL, input is transformed to this CRS for processing and back to the original CRS on return. When NULL, the input CRS is used as-is. Default 5070 (CONUS Albers Equal Area).
groups	character or factor vector, or NULL. Optional grouping vector the same length as polys. When non-NULL, polygons are dissolved per group and the result has one geometry per unique group value. Typically passed internally by the sf method; users should use group_id instead.

Value

sf data.frame or sfc (matching input class) with dissolved, hole-filled polygon(s).

Note

For very large inputs (10,000+ polygons), pre-dissolving with terra::aggregate() may be faster. The geos package is used automatically when installed for faster union operations.

Examples

```
# Three adjacent squares
p1 <- sf::st_polygon(list(rbind(c(0, 0), c(1, 0), c(1, 1), c(0, 1), c(0, 0))))
p2 <- sf::st_polygon(list(rbind(c(1, 0), c(2, 0), c(2, 1), c(1, 1), c(1, 0))))
p3 <- sf::st_polygon(list(rbind(c(2, 0), c(3, 0), c(3, 1), c(2, 1), c(2, 0))))
polys <- sf::st_sf(
  grp = c("a", "a", "b"),
  geometry = sf::st_sfc(p1, p2, p3, crs = 5070)
)

# Dissolve all into one polygon
dissolve_polygons(polys, work_crs = NULL)

# Dissolve by group
dissolve_polygons(polys, group_id = "grp", work_crs = NULL)

# sfc input returns sfc
dissolve_polygons(sf::st_geometry(polys), work_crs = NULL)
```

```
# Dissolve tributary basins with attribute summarisation
cats <- sf::read_sf(system.file("extdata/walker_cats.gpkg", package = "hydroloom"))
flines <- sf::read_sf(system.file("extdata/walker.gpkg", package = "hydroloom"))

# chosen manually for demonstration
outlets <- c(5329365, 5329313, 5329303)

# Navigate upstream from each outlet to define basins
basins <- navigate_network_dfs(flines, outlets, reset = FALSE, direction = "up")

# Label each catchment with its basin outlet
cats$basin <- NA_character_
for (i in seq_along(basins)) {
  cats$basin[cats$featureid %in% unlist(basins[[i]])] <- as.character(outlets[i])
}
cats <- cats[!is.na(cats$basin), ]

# Join stream names for summarisation
cats <- dplyr::left_join(cats,
  sf::st_drop_geometry(dplyr::select(flines, COMID, GNIS_NAME)),
  by = c("featureid" = "COMID"))

# Most common non-empty name in a group
most_common <- function(x) {
  x <- x[!is.na(x) & x != " "]
  if (length(x) == 0) return(NA_character_)
  names(sort(table(x), decreasing = TRUE))[1]
}

result <- dissolve_polygons(cats, group_id = "basin",
  .fns = list(areasqkm = sum, GNIS_NAME = most_common))
dplyr::select(result, basin, GNIS_NAME, areasqkm)

plot(sf::st_geometry(result), col = sf::sf.colors(nrow(result)))
```

`fix_flowdir` *Fix Flow Direction*

Description

If flowlines aren't digitized in the expected direction, this will reorder the nodes so they are.

Usage

```
fix_flowdir(id, network = NULL, fn_list = NULL)
```

Arguments

`id`	integer The id of the flowline to check

network	data.frame network compatible with hydroloom_names.
fn_list	list containing named elements flowline, network, and check_end, where flowline is the flowline to be checked and network the feature up or downstream of the flowline to be checked, and check_end is "start" or "end" depending if the network input is upstream ("start") or downstream ("end") of the flowline to be checked. This option allows pre-compilation of pairs of features which may be useful for very large numbers of flow direction checks.

Value

a geometry for the feature that has been reversed if needed.

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

# We add a tocomid with prepare_nhdplus
x <- add_toids(hy(x))

# Look at the end node of the 10th line.
(n1 <- get_node(x[10, ], position = "end"))

# Break the geometry by reversing it.
sf::st_geometry(x)[10] <- sf::st_reverse(sf::st_geometry(x)[10])

# Note that the end node is different now.
(n2 <- get_node(x[10, ], position = "end"))

# Pass the broken geometry to fix_flowdir with the network for toCOMID
sf::st_geometry(x)[10] <- fix_flowdir(x$id[10], x)

# Note that the geometry is now in the right order.
(n3 <- get_node(x[10, ], position = "end"))

plot(sf::st_geometry(x)[10])
plot(n1, add = TRUE)
plot(n2, add = TRUE, col = "blue")
plot(n3, add = TRUE, cex = 2, col = "red")
```

---

format_index_ids *DEPRECATED: Format Index ids*

Description

format_index_ids is deprecated and will be removed in a future release.

Usage

```
format_index_ids(g, return_list = FALSE)
```

Arguments

g	data.frame graph with id, inid and toindid as returned by make_index_ids
return_list	logical if TRUE, the returned list will include a "froms_list" element containing all from ids in a list form.

Value

list containing an adjacency matrix and a lengths vector indicating the number of connections from each node. If return_list is TRUE, return will also include a data.frame with an indid column and a toindid list column.

---

get_bridge_flowlines *Get Bridge Flowlines*

Description

Identifies bridge flowlines (cut edges) in the network. Bridge flowlines are those whose removal would disconnect the network. Flowlines are edges in the underlying node graph, so bridge detection correctly identifies the sole-path flowlines that separate parts of the network – including flowlines within diversion systems that do not rejoin.

Usage

```
get_bridge_flowlines(x, quiet = FALSE)

## S3 method for class 'data.frame'
get_bridge_flowlines(x, quiet = FALSE)

## S3 method for class 'hy'
get_bridge_flowlines(x, quiet = FALSE)

## S3 method for class 'hy_node'
get_bridge_flowlines(x, quiet = FALSE)

## S3 method for class 'hy_flownetwork'
get_bridge_flowlines(x, quiet = FALSE)

## S3 method for class 'hy_topo'
get_bridge_flowlines(x, quiet = FALSE)
```

Arguments

x	data.frame network compatible with hydroloom_names.
quiet	logical quiet messages?

Details

Required attributes: `id`, `toid`

Value

vector of flowline ids that are bridge flowlines in the network

Examples

```
x <- data.frame(
  id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  toid = c(2, 3, 4, 5, 0, 7, 8, 9, 4)
)

# 1 -> 2 -> 3 -> 4 -> 5
#               ^
#               |
# 6 -> 7 -> 8 -> 9
#
# Dendritic tree: all flowlines are bridges
get_bridge_flowlines(x)
```

`get_hydro_location` *Get Hydro Location*

Description

given a flowline index, returns the hydrologic location (point) along the specific linear element referenced by the index.

Usage

```
get_hydro_location(indexes, flowpath)
```

Arguments

`indexes`	data.frame as output from index_points_to_lines.
`flowpath`	data.frame with three columns: id, frommeas, and tomeas as well as geometry.

Value

sfc_POINT simple feature geometry list of length `nrow(indexes)`

Examples

```
if (require(nhdplusTools)) {
  source(system.file("extdata", "sample_flines.R", package = "nhdplusTools"))

  indexes <- index_points_to_lines(sample_flines,
    sf::st_sfc(sf::st_sfc(list(sf::st_point(c(-76.86934, 39.49328)),
      sf::st_point(c(-76.91711, 39.40884)),
      sf::st_point(c(-76.88081, 39.36354))),
    crs = 4326)))

  get_hydro_location(indexes, sample_flines)
}
```

---

get_node *Get Line Node*

Description

Given one or more lines, returns a particular node from the line.

Usage

```
get_node(x, position = "end")
```

Arguments

x	sf sf data.frame with one or more LINESTRING features
position	character either "start" or "end"

Value

sf data.frame containing requested nodes

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

start <- get_node(x, "start")
end <- get_node(x, "end")

plot(sf::st_zm(sf::st_geometry(x)),
  lwd = x$StreamOrde, col = "blue")
plot(sf::st_geometry(start), add = TRUE)

plot(sf::st_zm(sf::st_geometry(x)),
  lwd = x$StreamOrde, col = "blue")
plot(sf::st_geometry(end), add = TRUE)
```

get_partial_length *Get Partial Flowpath Length*

Description

Finds the upstream and downstream lengths along a given flowline. Internally, the function rescales the aggregate_id_measure to a id_measure and applies that rescaled measure to the length of the flowline.

Usage

```
get_partial_length(hydro_location, network = NULL, flowpath = NULL)
```

Arguments

hydro_location	list containing a hydrologic locations with names aggregate_id (reachcode) and aggregate_id_measure (reachcode measure).
network	data.frame network compatible with hydroloom_names.
flowpath	data.frame containing one flowpath that corresponds to the hydro_location. Not required if x is provided. x is not required if flowpath is provided.

Value

list containing up and dn elements with numeric length in km.

Examples

```
x <- sf::read_sf(system.file("extdata", "walker.gpkg", package = "hydroloom"))

hydro_location <- list(comid = 5329339,
  reachcode = "18050005000078",
  reach_meas = 30)

(pl <- get_partial_length(hydro_location, x))
```

hy *Create a hy Fabric S3 Object*

Description

converts a compatible dataset into a fabric s3 class. Automatically detects the network representation and classifies the result as the most specific hydroloom subclass (e.g. hy_topo, hy_node). With add_topo = TRUE, also auto-builds topology (e.g. toid from fromnode/tonode).

Usage

```
hy(x, clean = FALSE, add_topo = FALSE)
```

Arguments

x	data.frame network compatible with hydroloom_names.
clean	logical if TRUE, geometry and non-hydroloom compatible attributes will be removed.
add_topo	logical. If TRUE, auto-build topology (e.g. toid from fromnode/tonode) and classify into subclass hierarchy. If FALSE (default), classify based on existing columns only without constructing new attributes.

Value

hy object with attributes compatible with the hydroloom package.

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

hy(x)

hy(x, clean = TRUE)[1:10, ]

attr(hy(x), "orig_names")

# auto-build toid from fromnode/tonode
hy(x, add_topo = TRUE)
```

---

hydroloom_names *Get or Set Hydroloom Names*

Description

Retrieve hydroloom name mapping from hydroloom environment. Hydroloom uses a specific set of attribute names within the package and includes mappings from names used in some data sources. This function will return those names and can be used to set additional name mappings.

NOTE: these values will reset when R is restarted. Add desired settings to a project or user .Rprofile to make long term additions.

Usage

```
hydroloom_names(x = NULL, clear = FALSE)
```

Arguments

x	named character vector of additional names to add to the hydroloom environment. If not specified, no names will be added and the current value stored in the hydroloom environment will be returned.
clear	logical if TRUE, all names will be removed and replaced with x.

Value

named character vector containing hydroloom names with registered attribute name mappings in `names`.

Examples

```
hydroloom_names()
```

---

hydroloom_name_definitions *Hydroloom Name Definitions*

Description

A names character vector containing definitions of all attributes used in the hydroloom package.

Value

named character vector with `hydroloom_names` class to support custom print method

Examples

```
hydroloom_name_definitions
```

---

hy_capabilities *What operations are available for this network?*

Description

Returns a named logical vector indicating which hydroloom functions can operate on the input object given its current class and columns.

Functions marked TRUE are directly callable. Use `include_conversions = TRUE` to see what becomes available after a single representation conversion.

Usage

```
hy_capabilities(x, include_conversions = FALSE)
```

Arguments

x	object to query.
include_conversions	logical. If TRUE, also marks functions reachable via a single representation conversion.

Value

named logical vector.

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))
hy_capabilities(hy(x))
```

---

hy_flownetwork *hy_flownetwork: non-dendritic junction table*

Description

A hy_flownetwork object is a junction table — a non-dendritic edge list of id and toid rows where id may repeat. Each row records one connection between an upstream catchment or flowline and a downstream one, and optional upmain/downmain logical columns annotate which of the connections at a divergence is the main path. This is the only edge-list-shaped representation in hydroloom that preserves divergences directly.

Unlike the other class pages on this index, hy_flownetwork does **not** inherit from hy — id is not guaranteed to be a primary key, and a hy_flownetwork does not pass is.hy(). The user-facing producer to_flownetwork() drops the orig_names round-trip metadata that hy carries; a hy_flownetwork produced by hy() or classify_hy() from a non-dendritic data.frame may still have orig_names attached internally so name-aligned dispatch can restore the user's column names on return.

Details

hy_flownetwork is what hydroloom's internal classifier produces (via hy()) when an id/toid table has duplicated id values, and what to_flownetwork() produces from a hy_leveled input. The upmain/downmain annotation, when present, lets divergence-aware operations choose a main path without having to re-derive it from levelpath identity each time.

Several hydroloom operations that were originally written against hy_topo accept hy_flownetwork directly through delegate methods, because the underlying algorithm already tolerates non-dendritic input: add_streamorder(), accumulate_downstream(), get_bridge_flowlines(), add_levelpaths(), and make_node_topology() all work on hy_flownetwork without conversion.

Required columns

- `id` — catchment or flowline identifier; may be non-unique
- `toid` — downstream `id`

Optional:

- `upmain` — logical, `TRUE` for the main upstream connection at each junction
- `downmain` — logical, `TRUE` for the main downstream connection at each junction

See hydroloom_name_definitions for the canonical column definitions.

Functions that operate on hy_flownetwork

- Native operations: `navigate_network_dfs()`, `make_index_ids()`
- Delegated to the `hy_topo` algorithm: `add_streamorder()`, `accumulate_downstream()`, `get_bridge_flowlines()`, `add_levelpaths()`, `make_node_topology()`

Call `hy_capabilities()` on a specific object for the authoritative list given its current columns.

Conversions to other representations

- To hy_node (bipartite graph): `make_node_topology()`

Going the other direction — from `hy_node` or `hy_leveled` into `hy_flownetwork` — is what `to_flownetwork()` is for.

See Also

hy, hy_topo, hy_leveled, hy_node, `hy_capabilities()`, `hy_network_type()`, `is_dendritic()`, `to_flownetwork()`, `navigate_network_dfs()`, `make_index_ids()`

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

z <- to_flownetwork(x)

hy_network_type(z)

z
```

hy_leveled *hy_leveled: enriched self-referencing edge list*

Description

A hy_leveled object is a hy_topo carrying the additional columns produced by stream leveling: topo_sort, levelpath, and levelpath_outlet_id. These columns are what mainstem-aware operations (Pfafstetter coding, stream level, junction-table conversion) need in order to run.

hy_leveled is a strict subclass of hy_topo, so every function that accepts a hy_topo accepts a hy_leveled as well.

Details

hy_leveled exists to mark a hy_topo as having been through add_levelpaths() so downstream functions can dispatch on the presence of leveling without re-checking column names. The leveling columns encode mainstem path identity (levelpath), the outlet that closes that path (levelpath_outlet_id), and the topological order along the network (topo_sort).

Like hy_topo, hy_leveled requires unique id and cannot represent divergences as duplicated rows. Convert to hy_flownetwork via to_flownetwork() to preserve main and diverted paths in junction-table form.

Required columns

- id — catchment or flowline identifier, unique across rows
- toid — id of the immediately downstream feature
- topo_sort — topological sort order (NHDPlus hydrosequence)
- levelpath — mainstem path identifier
- levelpath_outlet_id — outlet id that closes each levelpath

See hydroloom_name_definitions for the canonical column definitions.

Functions that operate on hy_leveled

All hy_topo methods, plus the leveling-aware operations:

- Mainstem coding: add_pfafstetter(), add_streamlevel()
- Junction-table conversion: to_flownetwork()

Call hy_capabilities() on a specific object for the authoritative list given its current columns.

Conversions to other representations

- To hy_node (bipartite graph): make_node_topology()
- To hy_flownetwork (junction table): to_flownetwork(), which uses levelpath to decide which connection at a divergence is the main path

See Also

hy, hy_topo, hy_node, hy_flownetwork, hy_capabilities(), hy_network_type(), add_levelpaths(), add_pfafstetter(), to_flownetwork()

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

z <- add_levelpaths(add_toids(hy(x)),
                    name_attribute = "GNIS_ID",
                    weight_attribute = "arbolate_sum")

hy_network_type(z)
```

---

hy_network_type *What representation pattern does this network use?*

Description

Returns the most specific hydroloom class. The class names map to database / graph representation patterns:

- "hy_leveled": enriched edge list (+ topo_sort, levelpath)
- "hy_topo": self-referencing edge list (id/toid, dendritic)
- "hy_node": fromnode/tonode graph (id/fromnode/tonode)
- "hy_flownetwork": junction table (id/toid/upmain/downmain)
- "hy": name-aligned base object

Usage

```
hy_network_type(x)
```

Arguments

x	object to query.

Value

character. Most specific hydroloom class name, or "not a hydroloom object" if x has no hydroloom class.

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))
hy_network_type(hy(x))
```

hy_node *hy_node: bipartite feature-and-nexus graph*

Description

A hy_node object is a bipartite graph over catchments or flowlines and nexuses: each catchment (or flowline) is one row, carrying a fromnode (the upstream nexus it leaves) and a tonode (the downstream nexus it enters). Connectivity is recovered by matching tonode to fromnode across rows. This is the representation that natively supports divergences without duplicating rows — a divergence is simply a nexus with more than one outgoing feature.

As with hy_topo, a hy_node represents either a catchment topology or a flowline topology. Catchments carry a known local drainage area and are 1:1 with their flowpaths; flowlines are linear features without a committed local drainage area. The two graphs are practically related but functionally distinct.

hy_node inherits from hy.

Details

Because divergences are encoded through shared node identifiers rather than row duplication, hy_node requires unique id even on non-dendritic networks. The fromnode/tonode pair carries the connectivity that an id/toid edge list cannot.

hy_node is the entry point for hydroloom's divergence-aware operations: add_divergence() flags main and diverted paths from geometry, add_return_divergence() marks where diverted paths return to the main, and subset_network() walks the bipartite graph during subsetting.

Required columns

- id — catchment or flowline identifier, unique across rows
- fromnode — upstream nexus identifier
- tonode — downstream nexus identifier

See hydroloom_name_definitions for the canonical column definitions.

Functions that operate on hy_node

- Divergence handling: add_divergence(), add_return_divergence()
- Subsetting: subset_network()
- Edge-list construction: add_toids()

Call hy_capabilities() on a specific object for the authoritative list given its current columns.

Conversions to other representations

- To hy_topo (self-referencing edge list): add_toids(). With return_dendritic = TRUE (the default) the resulting toid drops secondary paths at divergences; set the option to FALSE and route through to_flownetwork() to keep them.

See Also

hy, hy_topo, hy_leveled, hy_flownetwork, hy_capabilities(), hy_network_type(), make_node_topology(), add_toids(), add_divergence(), subset_network()

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

z <- hy(x)

hy_network_type(z)

z
```

---

hy_reverse *Reverse* hy *to Original Names*

Description

renames hy object to original names and removes hy object attributes.

Usage

```
hy_reverse(x)
```

Arguments

x	data.frame network compatible with hydroloom_names.

Value

returns x with attribute names converted to original names provided to hy

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))
x <- hy(x)

hy_reverse(x)
```

hy_topo *hy_topo: self-referencing edge list*

Description

A hy_topo object is a self-referencing edge list representing either a catchment topology or a flowline topology: one row per feature, carrying an id and a toid that points at the immediately downstream feature. This is the main representation for downstream traversal, topological sorting, and most accumulation algorithms in hydroloom.

Catchment topology and flowline topology are functionally separate graphs. A catchment carries a known local drainage area and is 1:1 with its flowpath (the linear realization of the catchment). A flowline is a linear feature without a committed local drainage area — it may coincide with a catchment's flowpath, but that relationship is not guaranteed. hy_topo represents either graph; the two are practically related but should not be conflated.

hy_topo inherits from hy. The enriched subclass hy_leveled extends hy_topo with topo-sort and levelpath columns, so methods written for hy_topo apply to hy_leveled objects without modification.

Details

hy_topo requires unique id. A divergence would need two downstream connections from the same upstream feature — that is, two rows with the same id — and hy_topo does not permit that. Networks with divergences belong in hy_node (a bipartite graph through fromnode/tonode) or in hy_flownetwork (a junction table that annotates main and diverted paths). When the internal classifier called from hy() sees a duplicated id in an id/toid table, it produces an hy_flownetwork rather than an hy_topo.

For the authoritative, programmatic view of which functions are callable on a particular object, use hy_capabilities().

Required columns

- id — catchment or flowline identifier, unique across rows
- toid — id of the immediately downstream feature; network outlets carry the reserved outlet value (0 for numeric ids, "" for character)

See hydroloom_name_definitions for the canonical column definitions and accepted aliases.

Functions that operate on hy_topo

- Topology and sorting: sort_network(), add_topo_sort(), check_hy_graph()
- Path enrichment: add_levelpaths(), add_pathlength(), add_streamorder()
- Accumulation and traversal: accumulate_downstream(), navigate_hydro_network(), navigate_network_dfs(), make_index_ids()

Call hy_capabilities() on a specific object to see which functions are callable on it given its current columns.

Conversions to other representations

- To hy_node (bipartite graph): `make_node_topology()`
- To hy_leveled (enriched edge list): `add_levelpaths()`, which adds `topo_sort`, `levelpath`, and `levelpath_outlet_id`
- To hy_flownetwork (junction table): `add_levelpaths()` then `to_flownetwork()`

See Also

hy, hy_leveled, hy_node, hy_flownetwork, `hy_capabilities()`, `hy_network_type()`, `is_dendritic()`, `add_toids()`, `sort_network()`, `make_node_topology()`

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

z <- add_toids(hy(x))

hy_network_type(z)

z
```

---

index_points_to_lines *Index Points to Lines*

Description

given an sf point geometry column, return id, aggregate_id (e.g. reachcode), and aggregate id measure for each point.

Usage

```
index_points_to_lines(
  x,
  points,
  search_radius = NULL,
  precision = NA,
  max_matches = 1,
  ids = NULL
)

## S3 method for class 'data.frame'
index_points_to_lines(
  x,
  points,
  search_radius = NULL,
  precision = NA,
```

```
  max_matches = 1,
  ids = NULL
)

## S3 method for class 'hy'
index_points_to_lines(
  x,
  points,
  search_radius = NULL,
  precision = NA,
  max_matches = 1,
  ids = NULL
)
```

Arguments

x	data.frame network compatible with hydroloom_names.
points	sf or sfc of type POINT in analysis projection. NOTE: x will be projected to the projection of the points layer.
search_radius	units distance for the nearest neighbor search to extend in analysis projection. If missing or NULL, and points are in a lon lat projection, a default of 0.01 degree is used, otherwise 200 m is used. Conversion to the linear unit used by the provided crs of points is attempted. See RANN nn2 documentation for more details.
precision	numeric the resolution of measure precision in the output in meters.
max_matches	numeric the maximum number of matches to return if multiple are found in search_radius
ids	vector of ids corresponding to flowline ids from x of the same length as and order as points. If included, index searching will be constrained to one and only one flowline per point. search radius is still used with this option but max_matches is overridden.

Details

Required attributes: id and sf linestring geometry

Note 1: Inputs are cast into LINESTRINGS. Because of this, the measure output of inputs that are true multipart lines may be in error.

Note 2: This algorithm finds the nearest node in the input flowlines to identify which flowline the point should belong to. As a second pass, it can calculate the measure to greater precision than the nearest flowline geometry node.

Note 3: Offset is returned in units consistent with the projection of the input points.

Note 4: See dfMaxLength input to sf::st_segmentize() for details of handling of precision parameter.

Note 5: "from" is downstream – 0 is the outlet "to" is upstream – 100 is the inlet

Note 6: This function does not assume that it has access to the complete aggregate feature. From and to aggregate id measures must be included for each flowline in order to have aggregate id measures (reachcode or mainstem measures) in the output.

Value

data.frame with up to five columns, point_id, id, aggregate_id, aggregate_id_measure, and offset. point_id is the row or list element in the point input. If an aggregate_id (e.g. mainstem or reachcode) is not included in x. it will not be included in the output. If from and to measures are not included for each id in x, measures will not be included in the output.

Examples

```
if (require(nhdplusTools)) {
  source(system.file("extdata", "sample_flines.R", package = "nhdplusTools"))

  if (!any(lengths(sf::st_geometry(sample_flines)) > 1))
    sample_flines <- sf::st_cast(sample_flines, "LINESTRING", warn = FALSE)

  point <- sf::st_sfc(sf::st_point(c(-76.87479, 39.48233)),
    crs = 4326)

  index_points_to_lines(sample_flines, point)

  point <- sf::st_transform(point, 5070)

  index_points_to_lines(sample_flines, point,
    search_radius = units::set_units(200, "m"))

  index_points_to_lines(sample_flines, point, precision = 30)

  points <- sf::st_sfc(list(sf::st_point(c(-76.86934, 39.49328)),
    sf::st_point(c(-76.91711, 39.40884)),
    sf::st_point(c(-76.88081, 39.36354))),
  crs = 4326)

  index_points_to_lines(sample_flines, points,
    search_radius = units::set_units(0.2, "degrees"),
    max_matches = 10)

  index_points_to_lines(sample_flines, points,
    search_radius = units::set_units(0.2, "degrees"),
    ids = c(11689926, 11690110, 11688990))

}
```

---

index_points_to_waterbodies

Index Points to Waterbodies

Description

given an sf point geometry column, return waterbody id, and id of dominant artificial path

Usage

```
index_points_to_waterbodies(
  waterbodies,
  points,
  flines = NULL,
  search_radius = NULL
)
```

Arguments

waterbodies	sf data.frame of type POLYGON or MULTIPOLYGON including a "wbid" attribute.
points	sfc of type POINT
flines	sf data.frame (optional) of type LINESTRING or MULTILINESTRING including id, wbid, and topo_sort attributes. If omitted, only waterbody indexes are returned.
search_radius	units class with a numeric value indicating how far to search for a waterbody boundary in units of provided projection. Set units with set_units.

Value

data.frame with columns `in_wb_COMID` (or `in_wbid`), `near_wb_COMID` (or `near_wbid`), `near_wb_dist`, and `outlet_fline_COMID` (or `wb_outlet_id`). Column names use COMID when input contains a COMID attribute, otherwise hydroloom names (wbid) are used. Distance is in units of provided projection.

Examples

```
if (require(nhdplusTools)) {

  source(system.file("extdata/sample_data.R", package = "nhdplusTools"))

  waterbodies <- sf::st_transform(
    sf::read_sf(sample_data, "NHDWaterbody"), 5070)

  points <- sf::st_transform(
    sf::st_sfc(sf::st_point(c(-89.356086, 43.079943)),
      crs = 4326), 5070)

  index_points_to_waterbodies(waterbodies, points,
    search_radius = units::set_units(500, "m"))

}
```

is.hy *Is Valid* hy *Class?*

Description

test if object is a valid according to the hy s3 class

Usage

```
is.hy(x, silent = FALSE)
```

Arguments

x	object to test
silent	logical should messages be emitted?

Value

logical TRUE if valid

is_dendritic *Is the network dendritic?*

Description

Checks whether the network contains any divergences. Returns FALSE if a divergence column exists with any value of 2, or if the table has a toid column and duplicated id values (indicating row duplication from divergences). Otherwise returns TRUE.

Usage

```
is_dendritic(x, validate_data = TRUE)
```

Arguments

x	data.frame or hy object to test.
validate_data	logical. If FALSE, only checks the divergence column (fast). If TRUE (default), also checks for duplicated ids.

Value

logical. TRUE if the network is dendritic.

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))
is_dendritic(hy(x))
```

make_attribute_topology *Make Attribute Topology*

Description

given a set of lines with starting and ending nodes that form a geometric network, construct an attribute topology.

Usage

```
make_attribute_topology(x, min_distance)

## S3 method for class 'data.frame'
make_attribute_topology(x, min_distance)

## S3 method for class 'hy'
make_attribute_topology(x, min_distance)
```

Arguments

x	data.frame network compatible with hydroloom_names.
min_distance	numeric distance in units compatible with the units of the projection of lines. If no nodes are found within this distance, no connection will be returned.

Details

Required attributes: id and sf linestring geometry

If a future plan is set up, node distance calculations will be applied using future workers.

Value

data.frame with id and toid

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

y <- dplyr::select(x, COMID)

y <- sf::st_transform(y, 5070)

z <- make_attribute_topology(y, 10)

x <- add_toids(hy(x), return_dendritic = FALSE)

x[x$id == x$id[1], ]$toid
z[z$COMID == x$id[1], ]$toid
```

make_fromids *DEPRECATED Convert "to" index ids to "from" index ids*

Description

make_fromids is deprecated and will be removed in a future release. Use make_index_ids with `mode = "from"` instead.

Usage

```
make_fromids(index_ids, return_list = FALSE, upmain = NULL)
```

Arguments

`index_ids`	data.frame as returned by make_index_ids
`return_list`	logical if TRUE, the returned list will include a "froms_list" element containing all from ids in a list form.
`upmain`	data.frame containing `id` and `upmain` columns. `upmain` should be a logical value indicating if the id is the upmain connection from its downstream neighbors.

Value

list containing a "froms" matrix, "lengths" vector, and optionally "froms_list" elements.

make_index_ids *Make Index ids*

Description

makes index ids for the provided hy object. These can be used for graph traversal algorithms such that the row number and id are equal.

Usage

```
make_index_ids(x, mode = "to", long_form = NULL)

## S3 method for class 'data.frame'
make_index_ids(x, mode = "to", long_form = NULL)

## S3 method for class 'hy'
make_index_ids(x, mode = "to", long_form = NULL)

## S3 method for class 'hy_flownetwork'
make_index_ids(x, mode = "to", long_form = NULL)

## S3 method for class 'hy_topo'
make_index_ids(x, mode = "to", long_form = NULL)
```

Arguments

x	data.frame network compatible with hydroloom_names.
mode	character indicating the mode of the graph. Choose from "to", "from", or "both". Default is "to". Se Details for more information.
long_form	logical DEPRECATED

Details

Required attributes: id, toid

mode determines the direction of the graph. If "to", the graph will be directed from the id to the toid. If "from", the graph will be directed from the toid to the id. If "both", the list will contain both a "from" and a "to" element containing each version of the graph.

Value

list containing named elements:

to adjacency matrix with columns that correspond to unqiue(x$id)

lengths vector indicating the number of connections from each node

to_list a data.frame with an id, indid and a toindid list column.

List will have names froms, lengths, and froms_list for mode "from".

NOTE: the long_form output is deprecated and will be removed in a future release.

Examples

```
x <- data.frame(
  id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  toid = c(2, 3, 4, 5, 0, 7, 8, 9, 4)
)

make_index_ids(x, mode = "to")

make_index_ids(x, mode = "from")

make_index_ids(x, mode = "both")

x <- hy(sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom")))

x <- add_toids(x, return_dendritic = FALSE)

x <- make_index_ids(x)

names(x)
class(x$to)
class(x$lengths)
class(x$to_list)
is.list(x$to_list$toindid)
```

make_node_topology *Make Node Topology from Edge Topology*

Description

creates a node topology table from an edge topology

Usage

```
make_node_topology(x, add_div = NULL, add = TRUE)

## S3 method for class 'data.frame'
make_node_topology(x, add_div = NULL, add = TRUE)

## S3 method for class 'hy'
make_node_topology(x, add_div = NULL, add = TRUE)

## S3 method for class 'hy_flownetwork'
make_node_topology(x, add_div = NULL, add = TRUE)

## S3 method for class 'hy_topo'
make_node_topology(x, add_div = NULL, add = TRUE)
```

Arguments

x	data.frame network compatible with hydroloom_names.
add_div	data.frame of logical containing id and toid diverted paths to add. Should have id and toid fields. If TRUE, the network will be interpreted as a directed acyclic graph with downstream diversions included in the edge topology.
add	logical if TRUE, node topology will be added to x in return.

Details

Required attributes: id, toid

Value

data.frame containing id, fromnode, and tonode attributes or all attributes provided with id, fromnode and tonode in the first three columns.

If add_div is TRUE, will also add a divergence attribute where the provided diverted paths are assigned value 2, existing main paths that emanate from a divergence are assigned value 1, and all other paths are assigned value 0.

See Also

hy_topo, hy_node, add_toids()

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

y <- dplyr::select(add_toids(x), -ToNode, -FromNode)

y <- make_node_topology(y)

# just the divergences which have unique fromids in x but don't in new hope.
div <- add_toids(dplyr::select(x, COMID, FromNode, ToNode),
  return_dendritic = FALSE)
div <- div[div$toid %in%
  x$COMID[x$Divergence == 2], ]

y <- dplyr::select(add_toids(x), -ToNode, -FromNode)

y <- make_node_topology(y, add_div = div)
```

`navigate_connected_paths` *Navigate Connected Paths*

Description

Given a dendritic network and set of ids, finds paths or lengths between all identified flowpath outlets. This algorithm finds paths between outlets regardless of flow direction.

Usage

```
navigate_connected_paths(x, outlets, status = FALSE)
```

Arguments

`x`	data.frame network compatible with hydroloom_names.
`outlets`	vector or list of length 2 of ids from data.frame
`status`	logical print status and progress bars?

Details

Required attributes: `id`, `toid`, `length_km`

If `outlets` is a vector, all combinations (combn) of the given `outlets` will be produced. If `outlets` is a list, it must be of length 2 (froms, tos). Recycling is performed via expand.grid if child vector lengths do not match. `outlets` may also be a 2-column dataframe.

Value

data.frame containing the distance between pairs of network outlets and a list column containing flowpath identifiers along path that connect outlets. For a network with one terminal outlet, the data.frame will have `nrow(x)^2` rows.

Examples

```
x <- sf::read_sf(system.file("extdata", "walker.gpkg", package = "hydroloom"))

outlets <- c(5329303, 5329357, 5329317, 5329365, 5329435, 5329817)

x <- add_toids(hy(x))

navigate_connected_paths(x, outlets)
```

`navigate_hydro_network` *Navigate Hydro Network*

Description

Navigates a network of connected catchments using NHDPlus style network attributes.

Usage

```
navigate_hydro_network(x, start, mode, distance = NULL)

## S3 method for class 'data.frame'
navigate_hydro_network(x, start, mode, distance = NULL)

## S3 method for class 'hy'
navigate_hydro_network(x, start, mode, distance = NULL)

## S3 method for class 'hy_topo'
navigate_hydro_network(x, start, mode, distance = NULL)

## S3 method for class 'hy_node'
navigate_hydro_network(x, start, mode, distance = NULL)

## S3 method for class 'hy_leveled'
navigate_hydro_network(x, start, mode, distance = NULL)
```

Arguments

`x`	data.frame network compatible with hydroloom_names.
`start`	character or numeric to match identifier attribute. The starting catchment is included.
`mode`	character chosen from c(UM, DM, UT, DD) or equivalently c(upmain, downmain, up, down). 1. UM / upmain: upstream mainstem 2. DM / downmain: downstream mainstem

3. UT / up: upstream with tributaries
4. DD / down: downstream with diversions

`distance` numeric distance in km to limit navigation. The first catchment that exceeds the provided distance is included.

Details

if only `mode` is supplied, require network attributes are displayed.

NOTE: for "Upstream with tributaries" navigation, if a tributary emanates from a diversion and is the minor path downstream of that diversion, it will be included. This can have a very large impact when a diversion between two large river systems. To strictly follow the dendritic network, set the "dn_minor_topo_sort" attribute to all 0 in x.

Value

vector of identifiers found along navigation

Examples

```
plot_fun <- function(x, s, n) {
  plot(sf::st_geometry(x), col = "grey")
  plot(sf::st_geometry(x[x$id %in% n, ]), add = TRUE)
  plot(sf::st_geometry(x[x$id %in% s, ]), col = "red", lwd = 3, add = TRUE)
}

x <- hy(sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom")))

start <- 8891126
dm <- navigate_hydro_network(x, start, "DM")

plot_fun(x, start, dm)

dd <- navigate_hydro_network(x, start, "DD")

plot_fun(x, start, dd)

start <- 8894356

um <- navigate_hydro_network(x, start, "UM")

plot_fun(x, start, um)

ut <- navigate_hydro_network(x, start, "UT")

plot_fun(x, start, ut)
```

navigate_network_dfs *Navigate all Paths Depth First*

Description

given a starting node, return all reachable paths. Once visited, a node is marked as visited and will not take part in a future path.

Usage

```
navigate_network_dfs(x, starts, direction = "down", reset = FALSE)
```

Arguments

x	data.frame containing hydroloom compatible network or list as returned by make_index_ids ("to" mode for downstream or "from" mode for upstream). The index list format avoids recreating the index ids for every call to navigate network dfs in the case that it needs to be called many times.
starts	vector with ids from x to start at.
direction	character "up", "upmain", "down", or "downmain". If "upmain" or "downmain", x must contain sufficient information to construct an upmain and downmain network (see details).
reset	logical if TRUE, reset graph for each start such that later paths will have overlapping results.

Details

navigate_network_dfs offers two usage patterns. In the simple case, you can provide an hy in which case preprocessing is performed automatically, or you can do the preprocessing ahead of time and provide index ids. The latter is more complicated but can be much faster in certain circumstances.

hy object:

If the function will only be called one or a few times, it can be called with x containing (at a minimum) id and toid. For "upmain" and "downmain" support, x also requires attributes for determination of the primary upstream and downstream connection across every junction.

In this pattern, the hy object will be passed to make_index_ids called for every call to navigate_network_dfs and the resulting index ids will be used for network navigation.

Index ids:

If the function will be called repeatedly or index_ids are available for other reasons, the index_id list as created by make_index_ids ("to" mode for downstream or "from" mode for upstream). For "upmain" and "downmain" support, the main element must be included.

Value

list containing dfs result for each start.

Examples

```
x <- hy(sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom")))

x <- add_toids(x, return_dendritic = FALSE)

navigate_network_dfs(x, 8893402)

navigate_network_dfs(x, 8897784, direction = "up")
```

---

rename_geometry *Rename Geometry*

Description

correctly renames the geometry column of a sf object.

Usage

```
rename_geometry(g, name)
```

Arguments

g	sf data.table
name	character name to be used for geometry

Value

sf data.frame with geometry column renamed according to name parameter

Examples

```
(g <- sf::st_sf(a = 3, geo = sf::st_sfc(sf::st_point(1:2))))
rename_geometry(g, "geometry")
```

rescale_measures *Rescale Aggregate id Measure to id Measure*

Description

Given an aggregate_id (e.g. reachcode in NHDPlus) measure and the from and to measure for an id (e.g. COMID flowline in NHDPlus), returns the measure along the id flowline. This is useful where many flowlines make up a single aggregate feature. "Measures" are typically referenced to the aggregate feature. Flowlines have a stated from-measure / to-measure. In some cases it is useful to rescale the measure such that it is relative only to the flowline.

from is downstream – 0 is the outlet to is upstream – 100 is the inlet

Usage

```
rescale_measures(measure, from, to)
```

Arguments

measure	numeric aggregate measure between 0 and 100
from	numeric from-measure relative to the aggregate
to	numeric to-measure relative to the aggregate

Value

numeric rescaled measure

Examples

```
rescale_measures(40, 0, 50)
rescale_measures(60, 50, 100)
```

sort_network *Sort Network*

Description

given a network with an id and and toid, returns a sorted and potentially split set of output. Sort is from top to bottom so traversing the response from top to bottom will go from upstream to downstream.

Can also be used as a very fast implementation of upstream with tributaries navigation. The full network from each outlet is returned in sorted order.

If a network includes diversions, all flowlines downstream of the diversion are visited prior to continuing upstream. See note on the `outlets` parameter for implications of this implementation detail.

Usage

```
sort_network(x, split = FALSE, outlets = NULL)

## S3 method for class 'data.frame'
sort_network(x, split = FALSE, outlets = NULL)

## S3 method for class 'hy'
sort_network(x, split = FALSE, outlets = NULL)

## S3 method for class 'hy_node'
sort_network(x, split = FALSE, outlets = NULL)

## S3 method for class 'hy_flownetwork'
sort_network(x, split = FALSE, outlets = NULL)

## S3 method for class 'hy_topo'
sort_network(x, split = FALSE, outlets = NULL)
```

Arguments

x	data.frame network compatible with hydroloom_names.
split	logical if TRUE, the result will be split into independent networks identified by the id of their outlet. The outlet id of each independent network is added as a "terminal_id" attribute.
outlets	same as id in x. if specified, only the network emanating from these outlets will be considered and returned. NOTE: If outlets does not include all outlets from a given network containing diversions, a partial network may be returned.

Details

Required attributes: `id`, `toid`

Value

data.frame containing a topologically sorted version of the requested network and optionally a terminal id.

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

g <- add_toids(x)

head(g <- sort_network(g))

g$topo_sort <- nrow(g):1

plot(g['topo_sort'])
```

```
g <- add_toids(x, return_dendritic = FALSE)

g <- sort_network(g)

g$topo_sort <- nrow(g):1

plot(g['topo_sort'])
```

`st_compatibalize` *Make Spatial Inputs Compatible*

Description

makes sf1 compatible with sf2 by projecting into the projection of 2 and ensuring that the geometry columns are the same name.

Usage

```
st_compatibalize(sf1, sf2)
```

Arguments

`sf1`	sf data.frame
`sf2`	sf data.frame

Value

sf1 transformed and renamed to be compatible with sf2

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

one <- dplyr::select(x)
two <- sf::st_transform(one, 5070)

attr(one, "sf_column") <- "geotest"
names(one)[names(one) == "geom"] <- "geotest"

st_compatibalize(one, two)
```

subset_network *Subset Network*

Description

Subsets a network to features upstream of a given outlet. For non-dendritic networks, the function identifies flowpaths connected to the upstream basin via fromnode/tonode that are not reachable by upstream navigation alone. This is useful for networks where an upstream navigation returns a basin that contains a nested basin (closed or intersecting) connected through diversions. If only_up is FALSE, those nested basins are captured by navigating downstream from missed diversions to find their outlets, then navigating upstream from those outlets.

Note: If a diversion leaves a basin entirely, the subset will include the entire basin upstream of where the diversion terminates. this function will return both closed basins and intersecting basins.

Usage

```
subset_network(x, outlet, only_up = FALSE)

## S3 method for class 'data.frame'
subset_network(x, outlet, only_up = FALSE)

## S3 method for class 'hy'
subset_network(x, outlet, only_up = FALSE)

## S3 method for class 'hy_topo'
subset_network(x, outlet, only_up = FALSE)

## S3 method for class 'hy_node'
subset_network(x, outlet, only_up = FALSE)
```

Arguments

x	data.frame network compatible with hydroloom_names.
outlet	identifier of the outlet flowpath to subset upstream from.
only_up	logical if TRUE, only upstream navigation is used and any missed diversion connections are disconnected. If FALSE (default), nested endorheic basins reachable through diversions are also included.

Details

Required attributes: id, fromnode, tonode

Conditionally: divergence (if non-dendritic)

Value

data.frame subset of x containing flowpaths upstream of the outlet.

Note: if "toid" is included in the input, it will be returned without modification. This may result in one or more "toid" entries that contain ids that are not part of the subset.

Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

sub <- subset_network(x, 8893420)

nrow(sub)
```

---

to_flownetwork *To Flownetwork*

Description

converts an hy object into a flownetwork with "id", "toid", "upmain" and "downmain attributes.

Usage

```
to_flownetwork(x, warn_dendritic = TRUE)

## S3 method for class 'data.frame'
to_flownetwork(x, warn_dendritic = TRUE)

## S3 method for class 'hy'
to_flownetwork(x, warn_dendritic = TRUE)

## S3 method for class 'hy_node'
to_flownetwork(x, warn_dendritic = TRUE)

## S3 method for class 'hy_topo'
to_flownetwork(x, warn_dendritic = TRUE)

## S3 method for class 'hy_leveled'
to_flownetwork(x, warn_dendritic = TRUE)
```

Arguments

x	data.frame network compatible with hydroloom_names.
warn_dendritic	logical if TRUE and a dendritic toid attribute is provided, a warning will be emitted as toid is expected to be non-dendritic for any downmain to be FALSE.

Details

Required attributes:

`id` and `toid` or `fromnode` and `tonode`

`divergence` an attribute containing 0, 1, or 2 where 0 indicates there is only one downstream connection, 1 is the main connection downstream of a diversion and 2 is secondary connection downstream of a diversion.

`levelpath`, integer attribute which will have one and only one matching value upstream at a confluence.

Value

data.frame "id", "toid", "upmain" and "downmain attributes. A check is run to ensure upmain and downmain are valid with one and only one upmain and one and only one downmain from any given network element.

See Also

hy_flownetwork, hy_leveled, hy_node, `add_levelpaths()`

Examples

```
f <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))
to_flownetwork(f)
```

Index