Package 'ipumsr'

February 26, 2024

Title An R Interface for Downloading, Reading, and Handling IPUMS Data

Version 0.7.1

Description An easy way to work with census, survey, and geographic data provided by IPUMS in R. Generate and download data through the IPUMS API and load IPUMS files into R with their associated metadata to make analysis easier. IPUMS data describing 1.4 billion individuals drawn from over 750 censuses and surveys is available free of charge from the IPUMS website <https://www.ipums.org>.

License Mozilla Public License 2.0

URL https://tech.popdata.org/ipumsr/, https://github.com/ipums/ipumsr, https://www.ipums.org

BugReports https://github.com/ipums/ipumsr/issues

Depends R (>= 3.6)

Imports dplyr (>= 0.7.0), haven (>= 2.2.0), hipread (>= 0.2.0), httr, jsonlite, lifecycle, purrr, R6, readr, rlang, tibble, tidyselect, xml2, zeallot

Suggests biglm, covr, crayon, DBI, dbplyr, DT, ggplot2, htmltools, knitr, rmapshaper, rmarkdown, RSQLite (>= 2.3.3), rstudioapi, scales, sf, shiny, testthat, tidyr, vcr (>= 0.6.0), withr

VignetteBuilder knitr

Contact ipums@umn.edu

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author Greg Freedman Ellis [aut],
Derek Burk [aut, cre],
Finn Roberts [aut],
Joe Grover [ctb],
Dan Ehrlich [ctb],
Renae Rodgers [ctb],
Institute for Social Research and Data Innovation [cph]

Maintainer Derek Burk <ipums+cran@umn.edu>

Repository CRAN

Date/Publication 2024-02-26 16:00:03 UTC

R topics documented:

`define_extract-micro` *Define an extract request for an IPUMS microdata collection*

Description

Define the parameters of an IPUMS microdata extract request to be submitted via the IPUMS API.

Currently supported microdata collections include:

- **IPUMS USA**: `define_extract_usa()`
- **IPUMS CPS**: `define_extract_cps()`
- **IPUMS International**: `define_extract_ipumsi()`

Learn more about the IPUMS API in `vignette("ipums-api")` and microdata extract definitions in `vignette("ipums-api-micro")`.

Usage

```
define_extract_usa(
  description,
  samples,
  variables,
  data_format = "fixed_width",
  data_structure = "rectangular",
  rectangular_on = NULL,
  case_select_who = "individuals",
  data_quality_flags = NULL
)

define_extract_cps(
  description,
  samples,
  variables,
  data_format = "fixed_width",
  data_structure = "rectangular",
  rectangular_on = NULL,
  case_select_who = "individuals",
  data_quality_flags = NULL
)

define_extract_ipumsi(
  description,
  samples,
  variables,
  data_format = "fixed_width",
  data_structure = "rectangular",
  rectangular_on = NULL,
```

```
  case_select_who = "individuals",
  data_quality_flags = NULL
)
```

Arguments

`description`	Description of the extract.
`samples`	Vector of samples to include in the extract request. Use `get_sample_info()` to identify sample IDs for a given collection.
`variables`	Vector of variable names or a list of detailed variable specifications to include in the extract request. Use `var_spec()` to create a `var_spec` object containing a detailed variable specification. See examples.
`data_format`	Format for the output extract data file. Either `"fixed_width"` or `"csv"`. Note that while `"stata"`, `"spss"`, or `"sas9"` are also accepted, these file formats are not supported by ipumsr data-reading functions. Defaults to `"fixed_width"`.
`data_structure`	Data structure for the output extract data. • `"rectangular"` provides person records with all requested household information attached to respective household members. • `"hierarchical"` provides household records followed by person records. Defaults to `"rectangular"`.
`rectangular_on`	If `data_structure` is `"rectangular"`, records on which to rectangularize. Currently only `"P"` (person records) is supported. Defaults to `"P"` if `data_structure` is `"rectangular"` and `NULL` otherwise.
`case_select_who`	Indication of how to interpret any case selections included for variables in the extract definition. • `"individuals"` includes records for all individuals who match the specified case selections. • `"households"` includes records for all members of each household that contains an individual who matches the specified case selections. Defaults to `"individuals"`. Use `var_spec()` to add case selections for specific variables.
`data_quality_flags`	Set to `TRUE` to include data quality flags for all applicable variables in the extract definition. This will override the `data_quality_flags` specification for individual variables in the definition. Use `var_spec()` to add data quality flags for specific variables.

Value

An object of class `micro_extract` containing the extract definition.

See Also

`submit_extract()` to submit an extract request for processing.

`save_extract_as_json()` and `define_extract_from_json()` to share an extract definition.

Examples

```
usa_extract <- define_extract_usa(
  description = "2013-2014 ACS Data",
  samples = c("us2013a", "us2014a"),
  variables = c("SEX", "AGE", "YEAR")
)

usa_extract

# Use `var_spec()` to created detailed variable specifications:
usa_extract <- define_extract_usa(
  description = "Example USA extract definition",
  samples = c("us2013a", "us2014a"),
  variables = var_spec(
    "SEX",
    case_selections = "2",
    attached_characteristics = c("mother", "father")
  )
)

# For multiple variables, provide a list of `var_spec` objects and/or
# variable names.
cps_extract <- define_extract_cps(
  description = "Example CPS extract definition",
  samples = c("cps2020_02s", "cps2020_03s"),
  variables = list(
    var_spec("AGE", data_quality_flags = TRUE),
    var_spec("SEX", case_selections = "2"),
    "RACE"
  )
)

cps_extract

# To recycle specifications to many variables, it may be useful to
# create variables prior to defining the extract:
var_names <- c("AGE", "SEX")

my_vars <- purrr::map(
  var_names,
  ~ var_spec(.x, attached_characteristics = "mother")
)

ipumsi_extract <- define_extract_ipumsi(
  description = "Extract definition with predefined variables",
  samples = c("br2010a", "cl2017a"),
  variables = my_vars
)

# Extract specifications can be indexed by name
names(ipumsi_extract$samples)
```

```
names(ipumsi_extract$variables)

ipumsi_extract$variables$AGE

## Not run:
# Use the extract definition to submit an extract request to the API
submit_extract(usa_extract)

## End(Not run)
```

`define_extract_nhgis` *Define an IPUMS NHGIS extract request*

Description

Define the parameters of an IPUMS NHGIS extract request to be submitted via the IPUMS API.

Use `get_metadata_nhgis()` to browse and identify data sources for use in NHGIS extract definitions. For general information, see the NHGIS data source overview and the FAQ.

Learn more about the IPUMS API in `vignette("ipums-api")` and NHGIS extract definitions in `vignette("ipums-api-nhgis")`.

Usage

```
define_extract_nhgis(
  description = "",
  datasets = NULL,
  time_series_tables = NULL,
  shapefiles = NULL,
  geographic_extents = NULL,
  breakdown_and_data_type_layout = NULL,
  tst_layout = NULL,
  data_format = NULL
)
```

Arguments

`description`	Description of the extract.
`datasets`	List of dataset specifications for any datasets to include in the extract request. Use `ds_spec()` to create a `ds_spec` object containing a dataset specification. See examples.
`time_series_tables`	List of time series table specifications for any time series tables to include in the extract request. Use `tst_spec()` to create a `tst_spec` object containing a time series table specification. See examples.
`shapefiles`	Names of any shapefiles to include in the extract request.

`geographic_extents`
: Vector of geographic extents to use for all of the `datasets` in the extract definition (for instance, to obtain data within a particular state). Use `"*"` to select all available extents.

  Required when any of the `datasets` included in the extract definition include `geog_levels` that require extent selection. See `get_metadata_nhgis()` to determine if a geographic level requires extent selection. At the time of writing, NHGIS supports extent selection only for blocks and block groups.

`breakdown_and_data_type_layout`
: The desired layout of any `datasets` that have multiple data types or breakdown values.

  - `"single_file"` (default) keeps all data types and breakdown values in one file
  - `"separate_files"` splits each data type or breakdown value into its own file

  Required if any `datasets` included in the extract definition consist of multiple data types (for instance, estimates and margins of error) or have multiple breakdown values specified. See `get_metadata_nhgis()` to determine whether a requested dataset has multiple data types.

`tst_layout`
: The desired layout of all `time_series_tables` included in the extract definition.

  - `"time_by_column_layout"` (wide format, default): rows correspond to geographic units, columns correspond to different times in the time series
  - `"time_by_row_layout"` (long format): rows correspond to a single geographic unit at a single point in time
  - `"time_by_file_layout"`: data for different times are provided in separate files

  Required when an extract definition includes any `time_series_tables`.

`data_format`
: The desired format of the extract data file.

  - `"csv_no_header"` (default) includes only a minimal header in the first row
  - `"csv_header"` includes a second, more descriptive header row.
  - `"fixed_width"` provides data in a fixed width format

  Note that by default, `read_nhgis()` removes the additional header row in `"csv_header"` files.

  Required when an extract definition includes any `datasets` or `time_series_tables`.

Details

An NHGIS extract definition must include at least one dataset, time series table, or shapefile specification.

Create an NHGIS dataset specification with `ds_spec()`. Each dataset must be associated with a selection of `data_tables` and `geog_levels`. Some datasets also support the selection of `years` and `breakdown_values`.

Create an NHGIS time series table specification with `tst_spec()`. Each time series table must be associated with a selection of `geog_levels` and may optionally be associated with a selection of `years`.

See examples or `vignette("ipums-api-nhgis")` for more details about specifying datasets and time series tables in an NHGIS extract definition.

Value

An object of class `nhgis_extract` containing the extract definition.

See Also

`get_metadata_nhgis()` to find data to include in an extract definition.

`submit_extract()` to submit an extract request for processing.

`save_extract_as_json()` and `define_extract_from_json()` to share an extract definition.

Examples

```
# Extract definition for tables from an NHGIS dataset
# Use `ds_spec()` to create an NHGIS dataset specification
nhgis_extract <- define_extract_nhgis(
  description = "Example NHGIS extract",
  datasets = ds_spec(
    "1990_STF3",
    data_tables = "NP57",
    geog_levels = c("county", "tract")
  )
)

nhgis_extract

# Use `tst_spec()` to create an NHGIS time series table specification
define_extract_nhgis(
  description = "Example NHGIS extract",
  time_series_tables = tst_spec("CL8", geog_levels = "county"),
  tst_layout = "time_by_row_layout"
)

# To request multiple datasets, provide a list of `ds_spec` objects
define_extract_nhgis(
  description = "Extract definition with multiple datasets",
  datasets = list(
    ds_spec("2014_2018_ACS5a", "B01001", c("state", "county")),
    ds_spec("2015_2019_ACS5a", "B01001", c("state", "county"))
  )
)

# If you need to specify the same table or geographic level for
# many datasets, you may want to make a set of datasets before defining
# your extract request:
dataset_names <- c("2014_2018_ACS5a", "2015_2019_ACS5a")

dataset_spec <- purrr::map(
  dataset_names,
  ~ ds_spec(
```

```
    .x,
    data_tables = "B01001",
    geog_levels = c("state", "county")
  )
)

define_extract_nhgis(
  description = "Extract definition with multiple datasets",
  datasets = dataset_spec
)

# You can request datasets, time series tables, and shapefiles in the same
# definition:
define_extract_nhgis(
  description = "Extract with datasets and time series tables",
  datasets = ds_spec("1990_STF1", c("NP1", "NP2"), "county"),
  time_series_tables = tst_spec("CL6", "state"),
  shapefiles = "us_county_1990_tl2008"
)

# Extract specifications can be indexed by name
names(nhgis_extract$datasets)

nhgis_extract$datasets[["1990_STF3"]]

## Not run:
# Use the extract definition to submit an extract request to the API
submit_extract(nhgis_extract)

## End(Not run)
```

`download_extract` *Download a completed IPUMS data extract*

Description

Download an IPUMS data extract via the IPUMS API and write to disk.

Learn more about the IPUMS API in `vignette("ipums-api")`.

Usage

```
download_extract(
  extract,
  download_dir = getwd(),
  overwrite = FALSE,
  progress = TRUE,
  api_key = Sys.getenv("IPUMS_API_KEY")
)
```

Arguments

extract	One of: • An ipums_extract object • The data collection and extract number formatted as a string of the form "collection:number" or as a vector of the form c("collection", number) • An extract number to be associated with your default IPUMS collection. See set_ipums_default_collection() For a list of codes used to refer to each collection, see ipums_data_collections().
download_dir	Path to the directory where the files should be written. Defaults to current working directory.
overwrite	If TRUE, overwrite any conflicting files that already exist in download_dir. Defaults to FALSE.
progress	If TRUE, output progress bar showing the status of the download request. Defaults to TRUE.
api_key	API key associated with your user account. Defaults to the value of the IPUMS_API_KEY environment variable. See set_ipums_api_key().

Details

For NHGIS extracts, data files and GIS files (shapefiles) will be saved in separate .zip archives. download_extract() will return a character vector including the file paths to all downloaded files.

For microdata extracts, only the file path to the downloaded .xml DDI file will be returned, as it is sufficient for reading the data provided in the associated .gz data file.

Value

The path(s) to the files required to read the data requested in the extract, invisibly.

For NHGIS, paths will be named with either "data" (for tabular data files) or "shape" (for spatial data files) to indicate the type of data the file contains.

See Also

read_ipums_micro() or read_nhgis() to read tabular data from an IPUMS extract.

read_ipums_sf() to read spatial data from an IPUMS extract.

ipums_list_files() to list files in an IPUMS extract.

Examples

```
usa_extract <- define_extract_usa(
  description = "2013-2014 ACS Data",
  samples = c("us2013a", "us2014a"),
  variables = c("SEX", "AGE", "YEAR")
)

## Not run:
submitted_extract <- submit_extract(usa_extract)
```

```
downloadable_extract <- wait_for_extract(submitted_extract)

# For microdata, the path to the DDI .xml codebook file is provided.
usa_xml_file <- download_extract(downloadable_extract)

# Load with a `read_ipums_micro_*()` function
usa_data <- read_ipums_micro(usa_xml_file)

# You can also download previous extracts with their collection and number:
nhgis_files <- download_extract("nhgis:1")

# NHGIS extracts return a path to both the tabular and spatial data files,
# as applicable.
nhgis_data <- read_nhgis(data = nhgis_files["data"])

# Load NHGIS spatial data
nhgis_geog <- read_ipums_sf(data = nhgis_files["shape"])

## End(Not run)
```

`get_extract_history` *Browse definitions of previously submitted extract requests*

Description

Retrieve definitions of an arbitrary number of previously submitted extract requests for a given IPUMS collection, starting from the most recent extract request.

To check the status of a particular extract request, use `get_extract_info()`.

Learn more about the IPUMS API in `vignette("ipums-api")`.

Usage

```
get_extract_history(
  collection = NULL,
  how_many = 10,
  delay = 0,
  api_key = Sys.getenv("IPUMS_API_KEY")
)
```

Arguments

`collection`	Character string of the IPUMS collection for which to retrieve extract history. Defaults to the current default collection, if it exists. See `set_ipums_default_collection()`. For a list of codes used to refer to each collection, see `ipums_data_collections()`.
`how_many`	The number of extract requests for which to retrieve information. Defaults to the 10 most recent extracts.

delay	Number of seconds to delay between successive API requests, if multiple requests are needed to retrieve all records. A delay is highly unlikely to be necessary and is intended only as a fallback in the event that you cannot retrieve your extract history without exceeding the API rate limit.
api_key	API key associated with your user account. Defaults to the value of the IPUMS_API_KEY environment variable. See set_ipums_api_key().

Value

A list of ipums_extract objects

See Also

get_extract_info() to get the current status of a specific extract request.

Examples

```
## Not run:
# Get information for most recent extract requests.
# By default gets the most recent 10 extracts
get_extract_history("usa")

# Return only the most recent 3 extract definitions
get_extract_history("cps", how_many = 3)

# To get the most recent extract (for instance, if you have forgotten its
# extract number), use `get_last_extract_info()`
get_last_extract_info("nhgis")

## End(Not run)

# To browse your extract history by particular criteria, you can
# loop through the extract objects. We'll create a sample list of 2 extracts:
extract1 <- define_extract_usa(
  description = "2013 ACS",
  samples = "us2013a",
  variables = var_spec(
    "SEX",
    case_selections = "2",
    data_quality_flags = TRUE
  )
)

extract2 <- define_extract_usa(
  description = "2014 ACS",
  samples = "us2014a",
  variables = list(
    var_spec("RACE"),
    var_spec(
      "SEX",
      case_selections = "1",
```

```
      data_quality_flags = FALSE
    )
  )
)

extracts <- list(extract1, extract2)

# `purrr::keep()`` is particularly useful for filtering:
purrr::keep(extracts, ~ "RACE" %in% names(.x$variables))

purrr::keep(extracts, ~ grepl("2014 ACS", .x$description))

# You can also filter on variable-specific criteria
purrr::keep(extracts, ~ isTRUE(.x$variables[["SEX"]]$data_quality_flags))

# To filter based on all variables in an extract, you'll need to
# create a nested loop. For instance, to find all extracts that have
# any variables with data_quality_flags:
purrr::keep(
  extracts,
  function(extract) {
    any(purrr::map_lgl(
      names(extract$variables),
      function(var) isTRUE(extract$variables[[var]]$data_quality_flags)
    ))
  }
)

# To peruse your extract history without filtering, `purrr::map()` is more
# useful
purrr::map(extracts, ~ names(.x$variables))

purrr::map(extracts, ~ names(.x$samples))

purrr::map(extracts, ~ .x$variables[["RACE"]]$case_selections)

# Once you have identified a past extract, you can easily download or
# resubmit it
## Not run:
extracts <- get_extract_history("nhgis")

extract <- purrr::keep(
  extracts,
  ~ "CW3" %in% names(.x$time_series_tables)
)

download_extract(extract[[1]])

## End(Not run)
```

`get_extract_info` *Retrieve the definition and latest status of an extract request*

Description

Retrieve the latest status of an extract request.

`get_last_extract_info()` is a convenience function to retrieve the most recent extract for a given collection.

To browse definitions of your previously submitted extract requests, see `get_extract_history()`.

Learn more about the IPUMS API in `vignette("ipums-api")`.

Usage

```
get_extract_info(extract, api_key = Sys.getenv("IPUMS_API_KEY"))

get_last_extract_info(collection = NULL, api_key = Sys.getenv("IPUMS_API_KEY"))
```

Arguments

`extract`	One of: • An `ipums_extract` object • The data collection and extract number formatted as a string of the form `"collection:number"` or as a vector of the form `c("collection", number)` • An extract number to be associated with your default IPUMS collection. See `set_ipums_default_collection()` For a list of codes used to refer to each collection, see `ipums_data_collections()`.
`api_key`	API key associated with your user account. Defaults to the value of the `IPUMS_API_KEY` environment variable. See `set_ipums_api_key()`.
`collection`	Character string of the IPUMS collection for which to retrieve extract history. Defaults to the current default collection, if it exists. See `set_ipums_default_collection()`. For a list of codes used to refer to each collection, see `ipums_data_collections()`.

Value

An `ipums_extract` object.

See Also

`get_extract_history()` to browse past extract definitions

`wait_for_extract()` to wait for an extract to finish processing.

`download_extract()` to download an extract's data files.

`save_extract_as_json()` and `define_extract_from_json()` to share an extract definition.

Examples

```
my_extract <- define_extract_usa(
  description = "2013-2014 ACS Data",
  samples = c("us2013a", "us2014a"),
  variables = c("SEX", "AGE", "YEAR")
)

## Not run:
submitted_extract <- submit_extract(my_extract)

# Get latest info for the request associated with a given `ipums_extract`
# object:
updated_extract <- get_extract_info(submitted_extract)

updated_extract$status

# Or specify the extract collection and number:
get_extract_info("usa:1")
get_extract_info(c("usa", 1))

# If you have a default collection, you can use the extract number alone:
set_ipums_default_collection("nhgis")
get_extract_info(1)

# To get the most recent extract (for instance, if you have forgotten its
# extract number), use `get_last_extract_info()`
get_last_extract_info("nhgis")

## End(Not run)
```

`get_metadata_nhgis` *List available data sources from IPUMS NHGIS*

Description

Retrieve information about available NHGIS data sources, including datasets, data tables (summary tables), time series tables, and shapefiles (GIS files).

To retrieve summary metadata for all available data sources of a particular type, use the `type` argument. To retrieve detailed metadata for a single data source, use the `dataset`, `data_table`, or `time_series_table` argument. See the *metadata availability* section below for information on the metadata provided for each data type.

For general information, see the NHGIS data source overview and the FAQ.

Learn more about the IPUMS API in `vignette("ipums-api")` and NHGIS extract definitions in `vignette("ipums-api-nhgis")`.

Usage

```
get_metadata_nhgis(
  type = NULL,
  dataset = NULL,
  data_table = NULL,
  time_series_table = NULL,
  delay = 0,
  api_key = Sys.getenv("IPUMS_API_KEY")
)
```

Arguments

type	One of "datasets", "data_tables", "time_series_tables", or "shapefiles" indicating the type of summary metadata to retrieve. Leave NULL if requesting metadata for a single dataset, data_table, or time_series_table.
dataset	Name of an individual dataset for which to retrieve metadata.
data_table	Name of an individual data table for which to retrieve metadata. If provided, an associated dataset must also be specified.
time_series_table	Name of an individual time series table for which to retrieve metadata.
delay	Number of seconds to delay between successive API requests, if multiple requests are needed to retrieve all records. A delay is highly unlikely to be necessary and is intended only as a fallback in the event that you cannot retrieve all metadata records without exceeding the API rate limit. Only used if type is provided.
api_key	API key associated with your user account. Defaults to the value of the IPUMS_API_KEY environment variable. See set_ipums_api_key().

Value

If type is provided, a tibble of summary metadata for all data sources of the provided type. Otherwise, a named list of metadata for the specified dataset, data_table, or time_series_table.

Metadata availability

The following sections summarize the metadata fields provided for each data type. Summary metadata include a subset of the fields provided for individual data sources.

Datasets::

- name: The unique identifier for the dataset. This is the value that is used to refer to the dataset when interacting with the IPUMS API.
- group: The group of datasets to which the dataset belongs. For instance, 5 separate datasets are part of the "2015 American Community Survey" group.
- description: A short description of the dataset.
- sequence: Order in which the dataset will appear in the metadata API and extracts.

- `has_multiple_data_types`: Logical value indicating whether multiple data types exist for this dataset. For example, ACS datasets include both estimates and margins of error.
- `data_tables`: A `tibble` containing names, codes, and descriptions for all data tables available for the dataset.
- `geog_levels`: A `tibble` containing names, descriptions, and extent information for the geographic levels available for the dataset. The `has_geog_extent_selection` field contains logical values indicating whether extent selection is allowed (and required) for the associated geographic level. See `geographic_instances` below.
- `breakdowns`: A `tibble` containing names, types, descriptions, and breakdown values for all breakdowns available for the dataset.
- `years`: A vector of years for which the dataset is available. This field is only present if a dataset is available for multiple years. Note that ACS datasets are not considered to be available for multiple years.
- `geographic_instances`: A `tibble` containing names and descriptions for all valid geographic extents for the dataset. This field is only present if at least one of the dataset's `geog_levels` allows geographic extent selection.

Data tables::

- `name`: The unique identifier for the data table within its dataset. This is the value that is used to refer to the data table when interacting with the IPUMS API.
- `description`: A short description of the data table.
- `universe`: The statistical population measured by this data table (e.g. persons, families, occupied housing units, etc.)
- `nhgis_code`: The code identifying the data table in the extract. Variables in the extract data will include column names prefixed with this code.
- `sequence`: Order in which the data table will appear in the metadata API and extracts.
- `dataset_name`: Name of the dataset to which this data table belongs.
- `n_variables`: Number of variables included in this data table.
- `variables`: A `tibble` containing variable descriptions and codes for the variables included in the data table

Time series tables::

- `name`: The unique identifier for the time series table. This is the value that is used to refer to the time series table when interacting with the IPUMS API.
- `description`: A short description of the time series table.
- `geographic_integration`: The method by which the time series table aligns geographic units across time. `"Nominal"` integration indicates that geographic units are aligned by name (disregarding changes in unit boundaries). `"Standardized"` integration indicates that data from multiple time points are standardized to the indicated year's census units. For more information, click here.
- `sequence`: Order in which the time series table will appear in the metadata API and extracts.
- `time_series`: A `tibble` containing names and descriptions for the individual time series available for the time series table.
- `years`: A `tibble` containing information on the available data years for the time series table.
- `geog_levels`: A `tibble` containing names and descriptions for the geographic levels available for the time series table.

Shapefiles::

- `name`**:** The unique identifier for the shapefile. This is the value that is used to refer to the shapefile when interacting with the IPUMS API.
- `year`**:** The survey year in which the shapefile's represented areas were used for tabulations, which may be different than the vintage of the represented areas. For more information, click here.
- `geographic_level`**:** The geographic level of the shapefile.
- `extent`**:** The geographic extent covered by the shapefile.
- `basis`**:** The derivation source of the shapefile.
- `sequence`**:** Order in which the shapefile will appear in the metadata API and extracts.

See Also

`define_extract_nhgis()` to create an IPUMS NHGIS extract definition.

Examples

```
## Not run:
library(dplyr)

# Get summary metadata for all available sources of a given data type
get_metadata_nhgis("datasets")

# Filter to identify data sources of interest by their metadata values
all_tsts <- get_metadata_nhgis("time_series_tables")

tsts <- all_tsts %>%
  filter(
    grepl("Children", description),
    grepl("Families", description),
    geographic_integration == "Standardized to 2010"
  )

tsts$name

# Get detailed metadata for a single source with its associated argument:
cs5_meta <- get_metadata_nhgis(time_series_table = "CS5")
cs5_meta$geog_levels

# Use the available values when defining an NHGIS extract request
define_extract_nhgis(
  time_series_tables = tst_spec("CS5", geog_levels = "state")
)

# Detailed metadata is also provided for datasets and data tables
get_metadata_nhgis(dataset = "1990_STF1")
get_metadata_nhgis(data_table = "NP1", dataset = "1990_STF1")

# Iterate over data sources to retrieve detailed metadata for several
# records. For instance, to get variable metadata for a set of data tables:
tables <- c("NP1", "NP2", "NP10")
```

```
var_meta <- purrr::map(
  tables,
  function(dt) {
    dt_meta <- get_metadata_nhgis(dataset = "1990_STF1", data_table = dt)

    # This ensures you avoid hitting rate limit for large numbers of tables
    Sys.sleep(1)

    dt_meta$variables
  }
)

## End(Not run)
```

`ipums_bind_rows` *Bind multiple data frames by row, preserving labelled attributes*

Description

Analogous to `dplyr::bind_rows()`, but preserves the labelled attributes provided with IPUMS data.

Usage

```
ipums_bind_rows(..., .id = NULL)
```

Arguments

`...`	Data frames or `tibbles` to combine. Each argument can be a data frame or a list of data frames. When binding, columns are matched by name. Missing columns will be filled with `NA`.
`.id`	The name of an optional identifier column. Provide a string to create an output column that identifies each input. The column will use names if available, otherwise it will use positions.

Value

Returns the same type as the first input. Either a `data.frame`, `tbl_df`, or `grouped_df`

Examples

```
file <- ipums_example("nhgis0712_csv.zip")

d1 <- read_nhgis(
  file,
  file_select = 1,
  verbose = FALSE
)
```

```
d2 <- read_nhgis(
  file,
  file_select = 2,
  verbose = FALSE
)

# Variables have associated label attributes:
ipums_var_label(d1$PMSAA)

# Preserve labels when binding data sources:
d <- ipums_bind_rows(d1, d2)
ipums_var_label(d$PMSAA)

# dplyr `bind_rows()` drops labels:
d <- dplyr::bind_rows(d1, d2)
ipums_var_label(d$PMSAA)
```

`ipums_collect` *Collect data into R session with IPUMS attributes*

Description

Convenience wrapper around dplyr's `collect()` and `set_ipums_var_attributes()`. Use this to attach variable labels when collecting data from a database.

Usage

```
ipums_collect(data, ddi, var_attrs = c("val_labels", "var_label", "var_desc"))
```

Arguments

`data`	A dplyr `tbl` object (generally a `tbl_lazy` object stored in a database).
`ddi`	An ipums_ddi object created with `read_ipums_ddi()`.
`var_attrs`	Variable attributes to add to the output. Defaults to all available attributes. See `set_ipums_var_attributes()` for more details.

Value

A local `tibble` with the requested attributes attached.

ipums_data_collections *List IPUMS data collections*

Description

List IPUMS data collections with their corresponding codes used by the IPUMS API. Note that some data collections do not yet have API support.

Currently, ipumsr supports extract definitions for the following collections:

- IPUMS USA (`"usa"`)
- IPUMS CPS (`"cps"`)
- IPUMS International (`"ipumsi"`)
- IPUMS NHGIS (`"nhgis"`)

Learn more about the IPUMS API in `vignette("ipums-api")`.

Usage

```
ipums_data_collections()
```

Value

A `tibble` with four columns containing the full collection name, the type of data the collection provides, the collection code used by the IPUMS API, and the status of API support for the collection.

Examples

```
ipums_data_collections()
```

ipums_example *Get path to IPUMS example datasets*

Description

Construct file path to example extracts included with ipumsr. These data are used in package examples and can be used to experiment with ipumsr functionality.

Usage

```
ipums_example(path = NULL)
```

Arguments

`path`	Name of file. If `NULL`, all available example files will be listed.

Value

The path to a specific example file or a vector of all available files.

Examples

```
# List all available example files
ipums_example()

# Get path to a specific example file
file <- ipums_example("cps_00157.xml")

read_ipums_micro(file)
```

---

ipums_extract-class ipums_extract *class*

Description

The `ipums_extract` class provides a data structure for storing the extract definition and status of an IPUMS data extract request. Both submitted and unsubmitted extract requests are stored in `ipums_extract` objects.

`ipums_extract` objects are further divided into microdata and aggregate data classes, and will also include a collection-specific extract subclass to accommodate differences in extract options and content across collections.

Currently supported collections are:

- IPUMS microdata (`"micro_extract"`)
  - IPUMS USA (`"usa_extract"`)
  - IPUMS CPS (`"cps_extract"`)
  - IPUMS International (`"ipumsi_extract"`)
- IPUMS aggregate data (`"agg_extract"`)
  - IPUMS NHGIS (`"nhgis_extract"`)

Learn more about the IPUMS API in `vignette("ipums-api")`.

Properties

Objects of class `ipums_extract` have:

- A `class` attribute of the form `c("{collection}_extract", "{collection_type}_extract", "ipums_extract")`. For instance, `c("cps_extract", "micro_extract", "ipums_extract")`.
- A base type of `"list"`.
- A `names` attribute that is a character vector the same length as the underlying list.

All `ipums_extract` objects will include several core fields identifying the extract and its status:

- `collection`: the collection for the extract request.
- `description`: the description of the extract request.
- `submitted`: logical indicating whether the extract request has been submitted to the IPUMS API for processing.
- `download_links`: links to the downloadable data, if the extract request was completed at the time it was last checked.
- `number`: the number of the extract request. With `collection`, this uniquely identifies an extract request for a given user.
- `status`: status of the extract request at the time it was last checked. One of `"unsubmitted"`, `"queued"`, `"started"`, `"produced"`, `"canceled"`, `"failed"`, or `"completed"`.

Creating an extract

- Create an `ipums_extract` object from scratch with the appropriate `define_extract_*()` function. These functions take the form `define_extract_{collection}`.
- Use `get_extract_info()` to get the latest status of a submitted extract request.
- Use `get_extract_history()` to obtain the extract definitions of previously-submitted extract requests.

Submitting an extract

- Use `submit_extract()` to submit an extract request for processing through the IPUMS API.
- Use `wait_for_extract()` to periodically check the status of a submitted extract request until it is ready to download.
- Use `is_extract_ready()` to manually check whether a submitted extract request is ready to download.

Downloading an extract

- Download the data contained in a completed extract with `download_extract()`.

Saving an extract

- Save an extract to a JSON-formatted file with `save_extract_as_json()`.
- Create an `ipums_extract` object from a saved JSON-formatted definition with `define_extract_from_json()`.

`ipums_file_info` *Get file information for an IPUMS extract*

Description

Get information about the IPUMS project, date, notes, conditions, and citation requirements for an extract based on an ipums_ddi object.

`ipums_conditions()` is a convenience function that provides conditions and citation information for a recently loaded dataset.

Usage

```
ipums_file_info(object, type = NULL)

ipums_conditions(object = NULL)
```

Arguments

object	An ipums_ddi object. For ipums_conditions(), leave NULL to display conditions for most recently loaded dataset.
type	Type of file information to display. If NULL, loads all types. Otherwise, one of "ipums_project", "extract_date", "extract_notes", "conditions" or "citation".

Value

For ipums_file_info(), if type = NULL, a named list of metadata information. Otherwise, a string containing the requested information.

Examples

```
ddi <- read_ipums_ddi(ipums_example("cps_00157.xml"))

ipums_file_info(ddi)
```

---

ipums_list_files *List files contained within a zipped IPUMS extract*

Description

Identify the files that can be read from an IPUMS extract.

Usage

```
ipums_list_files(
  file,
  file_select = NULL,
  types = NULL,
  data_layer = deprecated(),
  shape_layer = deprecated(),
  raster_layer = deprecated()
)
```

Arguments

`file`	Path to a .zip archive containing the IPUMS extract to be examined.
`file_select`	If the path in `file` contains multiple files, a tidyselect selection identifying the files to be included in the output. Only files that match the provided expression will be included. While less useful, this can also be provided as a string specifying an exact file name or an integer to match files by index position.
`types`	One or more of `"data"` or `"shape"` indicating the type of files to include in the output. `"data"` refers to tabular data sources, while `"shape"` refers to spatial data sources. The use of `"raster"` has been deprecated and will be removed in a future release.
`data_layer, shape_layer, raster_layer`	**[Deprecated]** Please use `file_select` instead.

Value

A `tibble` containing the types and names of the available files.

See Also

`read_ipums_micro()` or `read_nhgis()` to read tabular data from an IPUMS extract.

`read_ipums_sf()` to read spatial data from an IPUMS extract.

Examples

```
nhgis_file <- ipums_example("nhgis0712_csv.zip")

# 2 available files in this extract
ipums_list_files(nhgis_file)

# Look for files that match a particular pattern:
ipums_list_files(nhgis_file, file_select = matches("ds136"))
```

---

ipums_shape_join *Join tabular data to geographic boundaries*

Description

These functions are analogous to dplyr's joins, except that:

- They operate on a data frame and an `sf` object
- They retain the variable attributes provided in IPUMS files and loaded by ipumsr data-reading functions
- They handle minor incompatibilities between attributes in spatial and tabular data that emerge in some IPUMS files

Usage

```
ipums_shape_left_join(
  data,
  shape_data,
  by,
  suffix = c("", "SHAPE"),
  verbose = TRUE
)

ipums_shape_right_join(
  data,
  shape_data,
  by,
  suffix = c("", "SHAPE"),
  verbose = TRUE
)

ipums_shape_inner_join(
  data,
  shape_data,
  by,
  suffix = c("", "SHAPE"),
  verbose = TRUE
)

ipums_shape_full_join(
  data,
  shape_data,
  by,
  suffix = c("", "SHAPE"),
  verbose = TRUE
)
```

Arguments

data	A tibble or data frame. Typically, this will contain data that has been aggregated to a specific geographic level.
shape_data	An sf object loaded with read_ipums_sf().
by	Character vector of variables to join by. See dplyr::left_join() for syntax.
suffix	If there are non-joined duplicate variables in the two data sources, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2. Defaults to adding the "SHAPE" suffix to duplicated variables in shape_file.
verbose	If TRUE, display information about any geometries that were unmatched during the join.

Value

An sf object containing the joined data

Examples

```
data <- read_nhgis(
  ipums_example("nhgis0972_csv.zip"),
  verbose = FALSE
)

sf_data <- read_ipums_sf(ipums_example("nhgis0972_shape_small.zip"))
joined_data <- ipums_shape_inner_join(data, sf_data, by = "GISJOIN")

colnames(joined_data)
```

---

ipums_var_info *Get contextual information about variables in an IPUMS data source*

Description

Summarize the variable metadata for the variables found in an ipums_ddi object or data frame. Provides descriptions of variable content (var_label and var_desc) as well as labels of particular values for each variable (val_labels).

ipums_var_info() produces a tibble summary of multiple variables at once.

ipums_var_label(), ipums_var_desc(), and ipums_val_labels() provide specific metadata for a single variable.

Usage

```
ipums_var_info(object, vars = NULL)

ipums_var_label(object, var = NULL)

ipums_var_desc(object, var = NULL)

ipums_val_labels(object, var = NULL)
```

Arguments

object	An ipums_ddi object, a data frame containing variable metadata (as produced by most ipumsr data-reading functions), or a haven::labelled() vector from a single column in such a data frame.
vars, var	A tidyselect selection identifying the variable(s) to include in the output. Only ipums_var_info() allows for the selection of multiple variables.

Details

For `ipums_var_info()`, if the provided `object` is a `haven::labelled()` vector (i.e. a single column from a data frame), the summary output will include the variable label, variable description, and value labels, if applicable.

If it is a data frame, the same information will be provided for all variables present in the data or to those indicated in `vars`.

If it is an ipums_ddi object, the summary will also include information used when reading the data from disk, including start/end positions for columns in the fixed-width file, implied decimals, and variable types.

Providing an `ipums_ddi` object is the most robust way to access variable metadata, as many data processing operations will remove these attributes from data frame-like objects.

Value

For `ipums_var_info()`, a `tibble` containing variable information.

Otherwise, a length-1 character vector with the requested variable information.

See Also

`read_ipums_ddi()` or `read_nhgis_codebook()` to read IPUMS metadata files.

Examples

```
ddi <- read_ipums_ddi(ipums_example("cps_00157.xml"))

# Info for all variables in a data source
ipums_var_info(ddi)

# Metadata for individual variables
ipums_var_desc(ddi, MONTH)

ipums_var_label(ddi, MONTH)

ipums_val_labels(ddi, MONTH)

# NHGIS also supports variable-level metadata, though many fields
# are not relevant and remain blank:
cb <- read_nhgis_codebook(ipums_example("nhgis0972_csv.zip"))

ipums_var_info(cb)
```

`ipums_view` *View a static webpage with variable metadata from an IPUMS extract*

Description

For a given ipums_ddi object or data frame, display metadata about its contents in the RStudio viewer pane. This includes extract-level information as well as metadata for the variables included in the input object.

It is also possible to save the output to an external HTML file without launching the RStudio viewer.

Usage

```
ipums_view(x, out_file = NULL, launch = TRUE)
```

Arguments

x	An ipums_ddi object or a data frame with IPUMS attributes attached. Note that file-level information (e.g. extract notes) is only available when x is an ipums_ddi object.
out_file	Optional location to save the output HTML file. If NULL, makes a temporary file.
launch	Logical indicating whether to launch the HTML file in the RStudio viewer pane. If TRUE, RStudio and rstudioapi must be available.

Details

ipums_view() requires that the htmltools, shiny, and DT packages are installed. If launch = TRUE, RStudio and the rstudioapi package must also be available.

Note that if launch = FALSE and out_file is unspecified, the output file will be written to a temporary directory. Some operating systems may be unable to open the HTML file from the temporary directory; we suggest that you manually specify the out_file location in this case.

Value

The file path to the output HTML file (invisibly, if launch = TRUE)

Examples

```
ddi <- read_ipums_ddi(ipums_example("cps_00157.xml"))

## Not run:
ipums_view(ddi)
ipums_view(ddi, "codebook.html", launch = FALSE)

## End(Not run)
```

ipums_website *Launch a browser window to an IPUMS metadata page*

Description

Launch the documentation webpage for a given IPUMS project and variable. The project can be provided in the form of an ipums_ddi object or can be manually specified.

This provides access to more extensive variable metadata than may be contained within an ipums_ddi object itself.

Note that some IPUMS projects (e.g. IPUMS NHGIS) do not have variable-specific pages. In these cases, ipums_website() will launch the project's main data selection page.

Usage

```
ipums_website(
  x,
  var = NULL,
  launch = TRUE,
  verbose = TRUE,
  homepage_if_missing = FALSE,
  project = deprecated(),
  var_label = deprecated()
)
```

Arguments

x	An ipums_ddi object or the name of an IPUMS project. See ipums_data_collections() for supported projects.
var	Name of the variable to load. If NULL, provides the URL to the project's main data selection site.
launch	If TRUE, launch a browser window to the metadata webpage. Otherwise, return the URL for the webpage.
verbose	If TRUE, produce warnings when invalid URL specifications are detected.
homepage_if_missing	If TRUE, return the IPUMS homepage if the IPUMS project in x is not recognized.
project	**[Deprecated]** Please use x instead.
var_label	**[Deprecated]** Variable label for the provided var. This is typically obtained from the input ipums_ddi object and is unlikely to be needed.

Details

If launch = TRUE, you will need a valid registration for the specified project to successfully launch the webpage.

Not all IPUMS variables are found at webpages that exactly match the variable names that are included in completed extract files (and `ipums_ddi` objects). Therefore, there may be some projects and variables for which `ipums_website()` will launch the page for a different variable or an invalid page.

Value

The URL to the IPUMS webpage for the indicated project and variable (invisibly if `launch = TRUE`)

Examples

```
ddi <- read_ipums_ddi(ipums_example("cps_00157.xml"))

## Not run:
# Launch webpage for particular variable
ipums_website(ddi, "MONTH")

## End(Not run)

# Can also specify an IPUMS project instead of an `ipums_ddi` object
ipums_website("IPUMS CPS", var = "RECTYPE", launch = FALSE)

# Shorthand project names from `ipums_data_collections()` are also accepted:
ipums_website("ipumsi", var = "YEAR", launch = FALSE)
```

---

lbl *Make a label placeholder object*

Description

Define a new label/value pair. For use in functions like `lbl_relabel()` and `lbl_add()`.

Usage

```
lbl(...)
```

Arguments

`...`	Either one or two arguments specifying the label (`.lbl`) and value (`.val`) to use in the new label pair. If arguments are named, they must be named `.val` and/or `.lbl`. If a single unnamed value is passed, it is used as the `.lbl` for the new label. If two unnamed values are passed, they are used as the `.val` and `.lbl`, respectively.

Details

Several `lbl_*()` functions include arguments that can be passed a function of `.val` and/or `.lbl`. These refer to the existing values and labels in the input vector, respectively.

Use `.val` to refer to the *values* in the vector's value labels. Use `.lbl` to refer to the *label names* in the vector's value labels.

Note that not all `lbl_*()` functions support both of these arguments.

Value

A `label_placeholder` object

See Also

Other lbl_helpers: `lbl_add()`, `lbl_clean()`, `lbl_define()`, `lbl_na_if()`, `lbl_relabel()`, `zap_ipums_attributes()`

Examples

```
# Label placeholder with no associated value
lbl("New label")

# Label placeholder with a value/label pair
lbl(10, "New label")

# Use placeholders as inputs to other label handlers
x <- haven::labelled(
  c(100, 200, 105, 990, 999, 230),
  c(`Unknown` = 990, NIU = 999)
)

x <- lbl_add(
  x,
  lbl(100, "$100"),
  lbl(105, "$105"),
  lbl(200, "$200"),
  lbl(230, "$230")
)

lbl_relabel(x, lbl(9999, "Missing") ~ .val > 900)
```

---

lbl_add *Add labels for unlabelled values*

Description

Add labels for values that don't already have them in a `labelled` vector.

Usage

```
lbl_add(x, ...)

lbl_add_vals(x, labeller = as.character, vals = NULL)
```

Arguments

x	A labelled vector
...	Arbitrary number of label placeholders created with lbl() indicating the value/label pairs to add.
labeller	A function that takes values being added as an argument and returns the labels to associate with those values. By default, uses the values themselves after converting to character.
vals	Vector of values to be labelled. If NULL, labels all unlabelled values that exist in the data.

Value

A labelled vector

See Also

Other lbl_helpers: lbl_clean(), lbl_define(), lbl_na_if(), lbl_relabel(), lbl(), zap_ipums_attributes()

Examples

```
x <- haven::labelled(
  c(100, 200, 105, 990, 999, 230),
  c(`Unknown` = 990, NIU = 999)
)

# Add new labels manually
lbl_add(
  x,
  lbl(100, "$100"),
  lbl(105, "$105"),
  lbl(200, "$200"),
  lbl(230, "$230")
)

# Add labels for all unlabelled values
lbl_add_vals(x)

# Update label names while adding
lbl_add_vals(x, labeller = ~ paste0("$", .))

# Add labels for select values
lbl_add_vals(x, vals = c(100, 200))
```

lbl_clean *Clean unused labels*

Description

Remove labels that do not appear in the data. When converting labelled values to a factor, this avoids the creation of additional factor levels.

Usage

```
lbl_clean(x)
```

Arguments

x	A labelled vector

Value

A labelled vector

See Also

Other lbl_helpers: lbl_add(), lbl_define(), lbl_na_if(), lbl_relabel(), lbl(), zap_ipums_attributes()

Examples

```
x <- haven::labelled(
  c(1, 2, 3, 1, 2, 3, 1, 2, 3),
  c(Q1 = 1, Q2 = 2, Q3 = 3, Q4 = 4)
)

lbl_clean(x)

# Compare the factor levels of the normal and cleaned labels after coercion
as_factor(lbl_clean(x))

as_factor(x)
```

lbl_define *Define labels for an unlabelled vector*

Description

Create a labelled vector from an unlabelled vector using lbl_relabel() syntax, allowing for the grouping of multiple values into a single label. Values not assigned a label remain unlabelled.

Usage

```
lbl_define(x, ...)
```

Arguments

x	An unlabelled vector
...	Arbitrary number of two-sided formulas. The left hand side should be a label placeholder created with `lbl()`. The right hand side should be a function taking `.val` that evaluates to `TRUE` for all cases that should receive the label specified on the left hand side. Can be provided as an anonymous function or formula. See Details section.

Details

Several `lbl_*()` functions include arguments that can be passed a function of `.val` and/or `.lbl`. These refer to the existing values and labels in the input vector, respectively.

Use `.val` to refer to the *values* in the vector's value labels. Use `.lbl` to refer to the *label names* in the vector's value labels.

Note that not all `lbl_*()` functions support both of these arguments.

Value

A `labelled` vector

See Also

Other lbl_helpers: `lbl_add()`, `lbl_clean()`, `lbl_na_if()`, `lbl_relabel()`, `lbl()`, `zap_ipums_attributes()`

Examples

```
age <- c(10, 12, 16, 18, 20, 22, 25, 27)

# Group age values into two label groups.
# Values not captured by the right hand side functions remain unlabelled
lbl_define(
  age,
  lbl(1, "Pre-college age") ~ .val < 18,
  lbl(2, "College age") ~ .val >= 18 & .val <= 22
)
```

`lbl_na_if` *Convert labelled data values to NA*

Description

Convert data values in a `labelled` vector to `NA` based on the value labels associated with that vector. Ignores values that do not have a label.

Usage

```
lbl_na_if(x, .predicate)
```

Arguments

`x`	A `labelled` vector
`.predicate`	A function taking `.val` and `.lbl` arguments that returns `TRUE` for all values that should be converted to `NA`. Can be provided as an anonymous function or formula. See Details section.

Details

Several `lbl_*()` functions include arguments that can be passed a function of `.val` and/or `.lbl`. These refer to the existing values and labels in the input vector, respectively.

Use `.val` to refer to the *values* in the vector's value labels. Use `.lbl` to refer to the *label names* in the vector's value labels.

Note that not all `lbl_*()` functions support both of these arguments.

Value

A `labelled` vector

See Also

Other lbl_helpers: `lbl_add()`, `lbl_clean()`, `lbl_define()`, `lbl_relabel()`, `lbl()`, `zap_ipums_attributes()`

Examples

```
x <- haven::labelled(
  c(10, 10, 11, 20, 30, 99, 30, 10),
  c(Yes = 10, `Yes - Logically Assigned` = 11, No = 20, Maybe = 30, NIU = 99)
)

# Convert labelled values greater than 90 to `NA`
lbl_na_if(x, function(.val, .lbl) .val >= 90)

# Can use purrr-style notation
lbl_na_if(x, ~ .lbl %in% c("Maybe"))
```

```
# Or refer to named function
na_function <- function(.val, .lbl) .val >= 90
lbl_na_if(x, na_function)
```

`lbl_relabel` *Modify value labels for a labelled vector*

Description

Update the mapping between values and labels in a `labelled` vector. These functions allow you to simultaneously update data values and the existing value labels. Modifying data values directly does not result in updated value labels.

Use `lbl_relabel()` to manually specify new value/label mappings. This allows for the addition of new labels.

Use `lbl_collapse()` to collapse detailed labels into more general categories. Values can be grouped together and associated with individual labels that already exist in the `labelled` vector.

Unlabelled values will be converted to `NA`.

Usage

```
lbl_relabel(x, ...)

lbl_collapse(x, .fun)
```

Arguments

`x`	A `labelled` vector
`...`	Arbitrary number of two-sided formulas. The left hand side should be a label placeholder created with `lbl()` or a value that already exists in the data. The right hand side should be a function taking `.val` and `.lbl` arguments that evaluates to `TRUE` for all cases that should receive the label specified on the left hand side. Can be provided as an anonymous function or formula. See Details section.
`.fun`	A function taking `.val` and `.lbl` arguments that returns the value associated with an existing label in the vector. Input values to this function will be relabeled with the label of the function's output value. Can be provided as an anonymous function or formula. See Details section.

Details

Several `lbl_*()` functions include arguments that can be passed a function of `.val` and/or `.lbl`. These refer to the existing values and labels in the input vector, respectively.

Use `.val` to refer to the *values* in the vector's value labels. Use `.lbl` to refer to the *label names* in the vector's value labels.

Note that not all `lbl_*()` functions support both of these arguments.

Value

A labelled vector

See Also

Other lbl_helpers: lbl_add(), lbl_clean(), lbl_define(), lbl_na_if(), lbl(), zap_ipums_attributes()

Examples

```
x <- haven::labelled(
  c(10, 10, 11, 20, 21, 30, 99, 30, 10),
  c(
    Yes = 10, `Yes - Logically Assigned` = 11,
    No = 20, Unlikely = 21, Maybe = 30, NIU = 99
  )
)

# Convert cases with value 11 to value 10 and associate with 10's label
lbl_relabel(x, 10 ~ .val == 11)
lbl_relabel(x, lbl("Yes") ~ .val == 11)

# To relabel using new value/label pairs, use `lbl()` to define a new pair
lbl_relabel(
  x,
  lbl(10, "Yes/Yes-ish") ~ .val %in% c(10, 11),
  lbl(90, "???") ~ .val == 99 | .lbl == "Maybe"
)

# Collapse labels to create new label groups
lbl_collapse(x, ~ (.val %/% 10) * 10)

# These are equivalent
lbl_collapse(x, ~ ifelse(.val == 10, 11, .val))
lbl_relabel(x, 11 ~ .val == 10)
```

---

read_ipums_ddi *Read metadata about an IPUMS microdata extract from a DDI codebook (.xml) file*

Description

Reads the metadata about an IPUMS extract from a DDI codebook into an ipums_ddi object.

These metadata contains parsing instructions for the associated fixed-width data file, contextual labels for variables and values in the data, and general extract information.

See *Downloading IPUMS files* below for information about downloading IPUMS DDI codebook files.

Usage

```
read_ipums_ddi(
  ddi_file,
  lower_vars = FALSE,
  file_select = deprecated(),
  data_layer = deprecated()
)
```

Arguments

`ddi_file`	Path to a DDI .xml file downloaded from IPUMS. See *Downloading IPUMS files* below.
`lower_vars`	Logical indicating whether to convert variable names to lowercase. Defaults to `FALSE` for consistency with IPUMS conventions.
`data_layer, file_select`	**[Deprecated]** Reading DDI files contained in a .zip archive has been deprecated. Please provide the full path to the .xml file to be loaded in `ddi_file`.

Value

An ipums_ddi object with metadata information.

Downloading IPUMS files

The DDI codebook (.xml) file provided with IPUMS microdata extracts can be downloaded through the IPUMS extract interface or (for some collections) within R using the IPUMS API.

If using the IPUMS extract interface:

- Download the DDI codebook by right clicking on the **DDI** link in the **Codebook** column of the extract interface and selecting **Save as...** (on Safari, you may have to select **Download Linked File As...**). Be sure that the codebook is downloaded in .xml format.

If using the IPUMS API:

- For supported collections, use `download_extract()` to download a completed extract via the IPUMS API. This automatically downloads both the DDI codebook and the data file from the extract and returns the path to the codebook file.

See Also

`read_ipums_micro()`, `read_ipums_micro_chunked()` and `read_ipums_micro_yield()` to read data from IPUMS microdata extracts.

`ipums_var_info()` and `ipums_file_info()` to view metadata about an ipums_ddi object.

`ipums_list_files()` to list files in an IPUMS extract.

Examples

```
# Example codebook file
ddi_file <- ipums_example("cps_00157.xml")

# Load data into an `ipums_ddi` obj
ddi <- read_ipums_ddi(ddi_file)

# Use the object to load its associated data
cps <- read_ipums_micro(ddi)

head(cps)

# Or get metadata information directly
ipums_var_info(ddi)

ipums_file_info(ddi)[1:2]

# If variable metadata have been lost from a data source, reattach from
# its corresponding `ipums_ddi` object:
cps <- zap_ipums_attributes(cps)

ipums_var_label(cps$STATEFIP)

cps <- set_ipums_var_attributes(cps, ddi$var_info)

ipums_var_label(cps$STATEFIP)
```

---

read_ipums_micro *Read data from an IPUMS microdata extract*

Description

Read a microdata dataset downloaded from the IPUMS extract system.

Two files are required to load IPUMS microdata extracts:

- A DDI codebook file (.xml) used to parse the extract's data file
- A data file (either .dat.gz or .csv.gz)

See *Downloading IPUMS files* below for more information about downloading these files.

`read_ipums_micro()` and `read_ipums_micro_list()` differ in their handling of extracts that contain multiple record types. See *Data structures* below.

Note that Stata, SAS, and SPSS file formats are not supported by ipumsr readers. Convert your extract to fixed-width or CSV format, or see haven for help loading these files.

Usage

```
read_ipums_micro(
  ddi,
  vars = NULL,
  n_max = Inf,
  data_file = NULL,
  verbose = TRUE,
  var_attrs = c("val_labels", "var_label", "var_desc"),
  lower_vars = FALSE
)

read_ipums_micro_list(
  ddi,
  vars = NULL,
  n_max = Inf,
  data_file = NULL,
  verbose = TRUE,
  var_attrs = c("val_labels", "var_label", "var_desc"),
  lower_vars = FALSE
)
```

Arguments

ddi	Either a path to a DDI .xml file downloaded from IPUMS, or an ipums_ddi object parsed by `read_ipums_ddi()`. See *Downloading IPUMS files* below.
vars	Names of variables to include in the output. Accepts a vector of names or a tidyselect selection. If `NULL`, includes all variables in the file. For hierarchical data, the `RECTYPE` variable is always included even if unspecified.
n_max	The maximum number of lines to read. For `read_ipums_micro_list()`, this applies before splitting records into list components.
data_file	Path to the data (.gz) file associated with the provided `ddi` file. By default, looks for the data file in the same directory as the DDI file. If the data file has been moved, specify its location here.
verbose	Logical indicating whether to display IPUMS conditions and progress information.
var_attrs	Variable attributes from the DDI to add to the columns of the output data. Defaults to all available attributes. See `set_ipums_var_attributes()` for more details.
lower_vars	If reading a DDI from a file, a logical indicating whether to convert variable names to lowercase. Defaults to `FALSE` for consistency with IPUMS conventions. This argument will be ignored if argument `ddi` is an ipums_ddi object. Use `read_ipums_ddi()` to convert variable names to lowercase when reading a DDI file. If `lower_vars = TRUE` and `vars` is specified, `vars` should reference the lowercase column names.

Value

`read_ipums_micro()` returns a single `tibble` object.

`read_ipums_micro_list()` returns a list of `tibble` objects with one entry for each record type.

Data structures

Files from IPUMS projects that contain data for multiple types of records (e.g. household records and person records) may be either rectangular or hierarchical.

Rectangular data are transformed such that each row of data represents only one type of record. For instance, each row will represent a person record, and all household-level information for that person will be included in the same row.

Hierarchical data have records of different types interspersed in a single file. For instance, a household record will be included in its own row followed by the person records associated with that household.

Hierarchical data can be read in two different formats:

- `read_ipums_micro()` reads data into a `tibble` where each row represents a single record, regardless of record type. Variables that do not apply to a particular record type will be filled with `NA` in rows of that record type. For instance, a person-specific variable will be missing in all rows associated with household records.
- `read_ipums_micro_list()` reads data into a list of `tibble` objects, where each list element contains only one record type. Each list element is named with its corresponding record type.

Downloading IPUMS files

You must download both the DDI codebook and the data file from the IPUMS extract system to load the data into R. `read_ipums_micro_*()` functions assume that the data file and codebook share a common base file name and are present in the same directory. If this is not the case, provide a separate path to the data file with the `data_file` argument.

If using the IPUMS extract interface:

- Download the data file by clicking **Download .dat** under **Download Data**.
- Download the DDI codebook by right clicking on the **DDI** link in the **Codebook** column of the extract interface and selecting **Save as...** (on Safari, you may have to select **Download Linked File as...**). Be sure that the codebook is downloaded in .xml format.

If using the IPUMS API:

- For supported collections, use `download_extract()` to download a completed extract via the IPUMS API. This automatically downloads both the DDI codebook and the data file from the extract and returns the path to the codebook file.

See Also

`read_ipums_micro_chunked()` and `read_ipums_micro_yield()` to read data from large IPUMS microdata extracts in chunks.

`read_ipums_ddi()` to read metadata associated with an IPUMS microdata extract.

`read_ipums_sf()` to read spatial data from an IPUMS extract.

`ipums_list_files()` to list files in an IPUMS extract.

Examples

```
# Codebook for rectangular example file
cps_rect_ddi_file <- ipums_example("cps_00157.xml")

# Load data based on codebook file info
cps <- read_ipums_micro(cps_rect_ddi_file)

head(cps)

# Can also load data from a pre-existing `ipums_ddi` object
# (This may be useful to retain codebook metadata even if lost from data
# during processing)
ddi <- read_ipums_ddi(cps_rect_ddi_file)
cps <- read_ipums_micro(ddi, verbose = FALSE)

# Codebook for hierarchical example file
cps_hier_ddi_file <- ipums_example("cps_00159.xml")

# Read in "long" format to get a single data frame
read_ipums_micro(cps_hier_ddi_file, verbose = FALSE)

# Read in "list" format and you get a list of multiple data frames
cps_list <- read_ipums_micro_list(cps_hier_ddi_file)

head(cps_list$PERSON)

head(cps_list$HOUSEHOLD)

# Use the `%<-%` operator from zeallot to unpack into separate objects
c(household, person) %<-% read_ipums_micro_list(cps_hier_ddi_file)

head(person)

head(household)
```

---

read_ipums_micro_chunked

Read data from an IPUMS microdata extract by chunk

Description

Read a microdata dataset downloaded from the IPUMS extract system in chunks.

Use these functions to read a file that is too large to store in memory at a single time. The file is processed in chunks of a given size, with a provided callback function applied to each chunk.

Two files are required to load IPUMS microdata extracts:

- A DDI codebook file (.xml) used to parse the extract's data file
- A data file (either .dat.gz or .csv.gz)

See *Downloading IPUMS files* below for more information about downloading these files.

`read_ipums_micro_chunked()` and `read_ipums_micro_list_chunked()` differ in their handling of extracts that contain multiple record types. See *Data structures* below.

Note that Stata, SAS, and SPSS file formats are not supported by ipumsr readers. Convert your extract to fixed-width or CSV format, or see haven for help loading these files.

Usage

```
read_ipums_micro_chunked(
  ddi,
  callback,
  chunk_size = 10000,
  vars = NULL,
  data_file = NULL,
  verbose = TRUE,
  var_attrs = c("val_labels", "var_label", "var_desc"),
  lower_vars = FALSE
)

read_ipums_micro_list_chunked(
  ddi,
  callback,
  chunk_size = 10000,
  vars = NULL,
  data_file = NULL,
  verbose = TRUE,
  var_attrs = c("val_labels", "var_label", "var_desc"),
  lower_vars = FALSE
)
```

Arguments

`ddi`	Either a path to a DDI .xml file downloaded from IPUMS, or an ipums_ddi object parsed by `read_ipums_ddi()`. See *Downloading IPUMS files* below.
`callback`	An ipums_callback object, or a function that will be converted to an `IpumsSideEffectCallback` object. Callback functions should include both data (`x`) and position (`pos`) arguments. See examples.
`chunk_size`	Integer number of observations to read per chunk. Higher values use more RAM, but typically result in faster processing. Defaults to 10,000.
`vars`	Names of variables to include in the output. Accepts a vector of names or a tidyselect selection. If `NULL`, includes all variables in the file. For hierarchical data, the `RECTYPE` variable is always included even if unspecified.

`data_file` Path to the data (.gz) file associated with the provided `ddi` file. By default, looks for the data file in the same directory as the DDI file. If the data file has been moved, specify its location here.

`verbose` Logical indicating whether to display IPUMS conditions and progress information.

`var_attrs` Variable attributes from the DDI to add to the columns of the output data. Defaults to all available attributes. See `set_ipums_var_attributes()` for more details.

`lower_vars` If reading a DDI from a file, a logical indicating whether to convert variable names to lowercase. Defaults to `FALSE` for consistency with IPUMS conventions.

This argument will be ignored if argument `ddi` is an ipums_ddi object. Use `read_ipums_ddi()` to convert variable names to lowercase when reading a DDI file.

Note that if reading in chunks from a .csv or .csv.gz file, the callback function will be called *before* variable names are converted to lowercase, and thus should reference uppercase variable names.

Value

Depends on the provided callback object. See ipums_callback.

Data structures

Files from IPUMS projects that contain data for multiple types of records (e.g. household records and person records) may be either rectangular or hierarchical.

Rectangular data are transformed such that each row of data represents only one type of record. For instance, each row will represent a person record, and all household-level information for that person will be included in the same row.

Hierarchical data have records of different types interspersed in a single file. For instance, a household record will be included in its own row followed by the person records associated with that household.

Hierarchical data can be read in two different formats:

- `read_ipums_micro_chunked()` reads each chunk of data into a `tibble` where each row represents a single record, regardless of record type. Variables that do not apply to a particular record type will be filled with `NA` in rows of that record type. For instance, a person-specific variable will be missing in all rows associated with household records. The provided `callback` function should therefore operate on a `tibble` object.
- `read_ipums_micro_list_chunked()` reads each chunk of data into a list of `tibble` objects, where each list element contains only one record type. Each list element is named with its corresponding record type. The provided `callback` function should therefore operate on a list object. In this case, the chunk size references the total number of rows *across* record types, rather than in each record type.

Downloading IPUMS files

You must download both the DDI codebook and the data file from the IPUMS extract system to load the data into R. `read_ipums_micro_*()` functions assume that the data file and codebook share a common base file name and are present in the same directory. If this is not the case, provide a separate path to the data file with the `data_file` argument.

If using the IPUMS extract interface:

- Download the data file by clicking **Download .dat** under **Download Data**.
- Download the DDI codebook by right clicking on the **DDI** link in the **Codebook** column of the extract interface and selecting **Save as...** (on Safari, you may have to select **Download Linked File as...**). Be sure that the codebook is downloaded in .xml format.

If using the IPUMS API:

- For supported collections, use `download_extract()` to download a completed extract via the IPUMS API. This automatically downloads both the DDI codebook and the data file from the extract and returns the path to the codebook file.

See Also

`read_ipums_micro_yield()` for more flexible handling of large IPUMS microdata files.

`read_ipums_micro()` to read data from an IPUMS microdata extract.

`read_ipums_ddi()` to read metadata associated with an IPUMS microdata extract.

`read_ipums_sf()` to read spatial data from an IPUMS extract.

`ipums_list_files()` to list files in an IPUMS extract.

Examples

```
suppressMessages(library(dplyr))

# Example codebook file
cps_rect_ddi_file <- ipums_example("cps_00157.xml")

# Function to extract Minnesota cases from CPS example
# (This can also be accomplished by including case selections
# in an extract definition)
#
# Function must take `x` and `pos` to refer to data and row position,
# respectively.
filter_mn <- function(x, pos) {
  x[x$STATEFIP == 27, ]
}

# Initialize callback
filter_mn_callback <- IpumsDataFrameCallback$new(filter_mn)

# Process data in chunks, filtering to MN cases in each chunk
read_ipums_micro_chunked(
  cps_rect_ddi_file,
```

```
  callback = filter_mn_callback,
  chunk_size = 1000,
  verbose = FALSE
)

# Tabulate INCTOT average by state without storing full dataset in memory
read_ipums_micro_chunked(
  cps_rect_ddi_file,
  callback = IpumsDataFrameCallback$new(
    function(x, pos) {
      x %>%
        mutate(
          INCTOT = lbl_na_if(
            INCTOT,
            ~ grepl("Missing|N.I.U.", .lbl)
          )
        ) %>%
        filter(!is.na(INCTOT)) %>%
        group_by(STATEFIP = as_factor(STATEFIP)) %>%
        summarize(INCTOT_SUM = sum(INCTOT), n = n(), .groups = "drop")
    }
  ),
  chunk_size = 1000,
  verbose = FALSE
) %>%
  group_by(STATEFIP) %>%
  summarize(avg_inc = sum(INCTOT_SUM) / sum(n))

# `x` will be a list when using `read_ipums_micro_list_chunked()`
read_ipums_micro_list_chunked(
  ipums_example("cps_00159.xml"),
  callback = IpumsSideEffectCallback$new(function(x, pos) {
    print(
      paste0(
        nrow(x$PERSON), " persons and ",
        nrow(x$HOUSEHOLD), " households in this chunk."
      )
    )
  }),
  chunk_size = 1000,
  verbose = FALSE
)

# Using the biglm package, you can even run a regression without storing
# the full dataset in memory
if (requireNamespace("biglm")) {
  lm_results <- read_ipums_micro_chunked(
    ipums_example("cps_00160.xml"),
    IpumsBiglmCallback$new(
      INCTOT ~ AGE + HEALTH, # Model formula
      function(x, pos) {
        x %>%
          mutate(
```

```
            INCTOT = lbl_na_if(
              INCTOT,
              ~ grepl("Missing|N.I.U.", .lbl)
            ),
            HEALTH = as_factor(HEALTH)
          )
      }
    ),
    chunk_size = 1000,
    verbose = FALSE
  )

  summary(lm_results)
}
```

`read_ipums_micro_yield` *Read data from an IPUMS microdata extract in yields*

Description

Read a microdata dataset downloaded from the IPUMS extract system into an object that can read and operate on a group ("yield") of lines at a time. Use these functions to read a file that is too large to store in memory at a single time. They represent a more flexible implementation of `read_ipums_micro_chunked()` using R6.

Two files are required to load IPUMS microdata extracts:

- A DDI codebook file (.xml) used to parse the extract's data file
- A data file (either .dat.gz or .csv.gz)

See *Downloading IPUMS files* below for more information about downloading these files.

`read_ipums_micro_yield()` and `read_ipums_micro_list_yield()` differ in their handling of extracts that contain multiple record types. See *Data structures* below.

Note that these functions only support fixed-width (.dat) data files.

Usage

```
read_ipums_micro_yield(
  ddi,
  vars = NULL,
  data_file = NULL,
  verbose = TRUE,
  var_attrs = c("val_labels", "var_label", "var_desc"),
  lower_vars = FALSE
)

read_ipums_micro_list_yield(
```

```
  ddi,
  vars = NULL,
  data_file = NULL,
  verbose = TRUE,
  var_attrs = c("val_labels", "var_label", "var_desc"),
  lower_vars = FALSE
)
```

Arguments

ddi	Either a path to a DDI .xml file downloaded from IPUMS, or an ipums_ddi object parsed by read_ipums_ddi(). See *Downloading IPUMS files* below.
vars	Names of variables to include in the output. Accepts a vector of names or a tidyselect selection. If NULL, includes all variables in the file. For hierarchical data, the RECTYPE variable is always included even if unspecified.
data_file	Path to the data (.gz) file associated with the provided ddi file. By default, looks for the data file in the same directory as the DDI file. If the data file has been moved, specify its location here.
verbose	Logical indicating whether to display IPUMS conditions and progress information.
var_attrs	Variable attributes from the DDI to add to the columns of the output data. Defaults to all available attributes. See set_ipums_var_attributes() for more details.
lower_vars	If reading a DDI from a file, a logical indicating whether to convert variable names to lowercase. Defaults to FALSE for consistency with IPUMS conventions. This argument will be ignored if argument ddi is an ipums_ddi object. Use read_ipums_ddi() to convert variable names to lowercase when reading a DDI file. If lower_vars = TRUE and vars is specified, vars should reference the lowercase column names.

Value

A HipYield R6 object (see Details section)

Methods summary:

These functions return a HipYield R6 object with the following methods:

- yield(n = 10000) reads the next "yield" from the data.

  For read_ipums_micro_yield(), returns a tibble with up to n rows.

  For read_ipums_micro_list_yield(), returns a list of tibbles with a total of up to n rows across list elements.

  If fewer than n rows are left in the data, returns all remaining rows. If no rows are left in the data, returns NULL.

- `reset()` resets the data so that the next yield will read data from the start.
- `is_done()` returns a logical indicating whether all rows in the file have been read.
- `cur_pos` contains the next row number that will be read (1-indexed).

Data structures

Files from IPUMS projects that contain data for multiple types of records (e.g. household records and person records) may be either rectangular or hierarchical.

Rectangular data are transformed such that each row of data represents only one type of record. For instance, each row will represent a person record, and all household-level information for that person will be included in the same row.

Hierarchical data have records of different types interspersed in a single file. For instance, a household record will be included in its own row followed by the person records associated with that household.

Hierarchical data can be read in two different formats:

- `read_ipums_micro_yield()` produces an object that yields data as a `tibble` whose rows represent single records, regardless of record type. Variables that do not apply to a particular record type will be filled with `NA` in rows of that record type. For instance, a person-specific variable will be missing in all rows associated with household records.
- `read_ipums_micro_list_yield()` produces an object that yields data as a list of `tibble` objects, where each list element contains only one record type. Each list element is named with its corresponding record type. In this case, when using `yield()`, `n` refers to the total number of rows *across* record types, rather than in each record type.

Downloading IPUMS files

You must download both the DDI codebook and the data file from the IPUMS extract system to load the data into R. `read_ipums_micro_*()` functions assume that the data file and codebook share a common base file name and are present in the same directory. If this is not the case, provide a separate path to the data file with the `data_file` argument.

If using the IPUMS extract interface:

- Download the data file by clicking **Download .dat** under **Download Data**.
- Download the DDI codebook by right clicking on the **DDI** link in the **Codebook** column of the extract interface and selecting **Save as...** (on Safari, you may have to select **Download Linked File as...**). Be sure that the codebook is downloaded in .xml format.

If using the IPUMS API:

- For supported collections, use `download_extract()` to download a completed extract via the IPUMS API. This automatically downloads both the DDI codebook and the data file from the extract and returns the path to the codebook file.

See Also

read_ipums_micro_chunked() to read data from large IPUMS microdata extracts in chunks.

read_ipums_micro() to read data from an IPUMS microdata extract.

read_ipums_ddi() to read metadata associated with an IPUMS microdata extract.

read_ipums_sf() to read spatial data from an IPUMS extract.

ipums_list_files() to list files in an IPUMS extract.

Examples

```
# Create an IpumsLongYield object
long_yield <- read_ipums_micro_yield(ipums_example("cps_00157.xml"))

# Yield the first 10 rows of the data
long_yield$yield(10)

# Yield the next 20 rows of the data
long_yield$yield(20)

# Check the current position after yielding 30 rows
long_yield$cur_pos

# Reset to the beginning of the file
long_yield$reset()

# Use a loop to flexibly process the data in pieces. Count all Minnesotans:
total_mn <- 0

while (!long_yield$is_done()) {
  cur_data <- long_yield$yield(1000)
  total_mn <- total_mn + sum(as_factor(cur_data$STATEFIP) == "Minnesota")
}

total_mn

# Can also read hierarchical data as list:
list_yield <- read_ipums_micro_list_yield(ipums_example("cps_00159.xml"))

# Yield size is based on total rows for all list elements
list_yield$yield(10)
```

---

read_ipums_sf *Read spatial data from an IPUMS extract*

Description

Read a spatial data file (also referred to as a GIS file or shapefile) from an IPUMS extract into an sf object from the sf package.

Usage

```
read_ipums_sf(
  shape_file,
  file_select = NULL,
  vars = NULL,
  encoding = NULL,
  bind_multiple = FALSE,
  add_layer_var = NULL,
  verbose = FALSE,
  shape_layer = deprecated()
)
```

Arguments

shape_file	Path to a single .shp file or a .zip archive containing at least one .shp file. See Details section.
file_select	If shape_file is a .zip archive that contains multiple files, an expression identifying the files to load. Accepts a character string specifying the file name, a tidyselect selection, or index position. If multiple files are selected, bind_multiple must be equal to TRUE.
vars	Names of variables to include in the output. Accepts a character vector of names or a tidyselect selection. If NULL, includes all variables in the file.
encoding	Encoding to use when reading the shape file. If NULL, defaults to "latin1" unless the file includes a .cpg metadata file with encoding information. The default value should generally be appropriate.
bind_multiple	If TRUE and shape_file contains multiple .shp files, row-bind the files into a single sf object. Useful when shape_file contains multiple files that represent the same geographic units for different extents (e.g. block-level data for multiple states).
add_layer_var	If TRUE, add a variable to the output data indicating the file that each row originates from. Defaults to FALSE unless bind_multiple = TRUE and multiple files exist in shape_file. The column name will always be prefixed with "layer", but will be adjusted to avoid name conflicts if another column named "layer" already exists in the data.
verbose	If TRUE report additional progress information on load.
shape_layer	**[Deprecated]** Please use file_select instead.

Details

Some IPUMS products provide shapefiles in a "nested" .zip archive. That is, each shapefile (including a .shp as well as accompanying files) is compressed in its own archive, and the collection of all shapefiles provided in an extract is also compressed into a single .zip archive.

read_ipums_sf() is designed to handle this structure. However, if any files are altered such that an internal .zip archive contains *multiple* shapefiles, this function will throw an error. If this is the case, you may need to manually unzip the downloaded file before loading it into R.

Value

An sf object

See Also

`read_ipums_micro()` or `read_nhgis()` to read tabular data from an IPUMS extract.

`ipums_list_files()` to list files in an IPUMS extract.

Examples

```
# Example shapefile from NHGIS
shape_ex1 <- ipums_example("nhgis0972_shape_small.zip")
data_ex1 <- read_nhgis(ipums_example("nhgis0972_csv.zip"), verbose = FALSE)

sf_data <- read_ipums_sf(shape_ex1)

sf_data

# To combine spatial data with tabular data without losing the attributes
# included in the tabular data, use an ipums shape join:
ipums_shape_full_join(data_ex1, sf_data, by = "GISJOIN")

shape_ex2 <- ipums_example("nhgis0712_shape_small.zip")

# Shapefiles are provided in .zip archives that may contain multiple
# files. Select a single file with `file_select`:
read_ipums_sf(shape_ex2, file_select = matches("us_pmsa_1990"))

# Or row-bind files with `bind_multiple`. This may be useful for files of
# the same geographic level that cover different extents)
read_ipums_sf(
  shape_ex2,
  file_select = matches("us_pmsa"),
  bind_multiple = TRUE
)
```

---

read_nhgis *Read tabular data from an NHGIS extract*

Description

Read a csv or fixed-width (.dat) file downloaded from the NHGIS extract system.

To read spatial data from an NHGIS extract, use `read_ipums_sf()`.

Usage

```
read_nhgis(
  data_file,
  file_select = NULL,
  vars = NULL,
  col_types = NULL,
  n_max = Inf,
  guess_max = min(n_max, 1000),
  do_file = NULL,
  var_attrs = c("val_labels", "var_label", "var_desc"),
  remove_extra_header = TRUE,
  verbose = TRUE,
  data_layer = deprecated()
)
```

Arguments

`data_file`	Path to a .zip archive containing an NHGIS extract or a single file from an NHGIS extract.
`file_select`	If `data_file` is a .zip archive that contains multiple files, an expression identifying the file to load. Accepts a character vector specifying the file name, a tidyselect selection, or an index position. This must uniquely identify a file.
`vars`	Names of variables to include in the output. Accepts a vector of names or a tidyselect selection. If `NULL`, includes all variables in the file.
`col_types`	One of `NULL`, a `cols()` specification or a string. If `NULL`, all column types will be inferred from the values in the first `guess_max` rows of each column. Alternatively, you can use a compact string representation to specify column types: • c = character • i = integer • n = number • d = double • l = logical • f = factor • D = date • T = date time • t = time • ? = guess • _ or - = skip See `read_delim()` for more details.
`n_max`	Maximum number of lines to read.
`guess_max`	For .csv files, maximum number of lines to use for guessing column types. Will never use more than the number of lines read.
`do_file`	For fixed-width files, path to the .do file associated with the provided `data_file`. The .do file contains the parsing instructions for the data file.

By default, looks in the same path as `data_file` for a .do file with the same name. See Details section below.

`var_attrs` Variable attributes to add from the codebook (.txt) file included in the extract. Defaults to all available attributes.

See `set_ipums_var_attributes()` for more details.

`remove_extra_header`

If `TRUE`, remove the additional descriptive header row included in some NHGIS .csv files.

This header row is not usually needed as it contains similar information to that included in the `"label"` attribute of each data column (if `var_attrs` includes `"var_label"`).

`verbose` Logical controlling whether to display output when loading data. If `TRUE`, displays IPUMS conditions, a progress bar, and column types. Otherwise, all are suppressed.

Will be overridden by `readr.show_progress` and `readr.show_col_types` options, if they are set.

`data_layer` **[Deprecated]** Please use `file_select` instead.

Details

The .do file that is included when downloading an NHGIS fixed-width extract contains the necessary metadata (e.g. column positions and implicit decimals) to correctly parse the data file. `read_nhgis()` uses this information to parse and recode the fixed-width data appropriately.

If you no longer have access to the .do file, consider resubmitting the extract that produced the data. You can also change the desired data format to produce a .csv file, which does not require additional metadata files to be loaded.

For more about resubmitting an existing extract via the IPUMS API, see `vignette("ipums-api", package = "ipumsr")`.

Value

A `tibble` containing the data found in `data_file`

See Also

`read_ipums_sf()` to read spatial data from an IPUMS extract.

`read_nhgis_codebook()` to read metadata about an IPUMS NHGIS extract.

`ipums_list_files()` to list files in an IPUMS extract.

Examples

```
# Example files
csv_file <- ipums_example("nhgis0972_csv.zip")
fw_file <- ipums_example("nhgis0730_fixed.zip")

# Provide the .zip archive directly to load the data inside:
read_nhgis(csv_file)
```

```
# For extracts that contain multiple files, use `file_select` to specify
# a single file to load. This accepts a tidyselect expression:
read_nhgis(fw_file, file_select = matches("ds239"), verbose = FALSE)

# Or an index position:
read_nhgis(fw_file, file_select = 2, verbose = FALSE)

# For CSV files, column types are inferred from the data. You can
# manually specify column types with `col_types`. This may be useful for
# geographic codes, which should typically be interpreted as character values
read_nhgis(csv_file, col_types = list(MSA_CMSAA = "c"), verbose = FALSE)

# Fixed-width files are parsed with the correct column positions
# and column types automatically:
read_nhgis(fw_file, file_select = contains("ts"), verbose = FALSE)

# You can also read in a subset of the data file:
read_nhgis(
  csv_file,
  n_max = 15,
  vars = c(GISJOIN, YEAR, D6Z002),
  verbose = FALSE
)
```

`read_nhgis_codebook` *Read metadata from an NHGIS codebook (.txt) file*

Description

[Experimental]

Read the variable metadata contained in the .txt codebook file included with NHGIS extracts into an ipums_ddi object.

Because NHGIS variable metadata do not adhere to all the standards of microdata DDI files, some of the `ipums_ddi` fields will not be populated.

This function is marked as experimental while we determine whether there may be a more robust way to standardize codebook and DDI reading across IPUMS collections.

Usage

```
read_nhgis_codebook(cb_file, file_select = NULL, raw = FALSE)
```

Arguments

`cb_file`	Path to a .zip archive containing an NHGIS extract or to an NHGIS codebook (.txt) file.

`file_select` If `cb_file` is a .zip archive or directory that contains multiple codebook files, an expression identifying the file to read. Accepts a character string specifying the file name, a tidyselect selection, or an index position of the file. Ignored if `cb_file` is the path to a single codebook file.

`raw` If `TRUE`, return a character vector containing the lines of `cb_file` rather than an `ipums_ddi` object. Defaults to `FALSE`.

Value

If `raw = FALSE`, an `ipums_ddi` object with information on the variables contained in the data for the extract associated with the given `cb_file`.

If `raw = TRUE`, a character vector with one element for each line of the given `cb_file`.

See Also

`read_nhgis()` to read tabular data from an IPUMS NHGIS extract.

`read_ipums_sf()` to read spatial data from an IPUMS extract.

`ipums_list_files()` to list files in an IPUMS extract.

Examples

```
# Example file
nhgis_file <- ipums_example("nhgis0972_csv.zip")

# Read codebook as an `ipums_ddi` object:
codebook <- read_nhgis_codebook(nhgis_file)

# Variable-level metadata about the contents of the data file:
ipums_var_info(codebook)

ipums_var_label(codebook, "PMSA")

# If variable metadata have been lost from a data source, reattach from
# the corresponding `ipums_ddi` object:
nhgis_data <- read_nhgis(nhgis_file, verbose = FALSE)

nhgis_data <- zap_ipums_attributes(nhgis_data)
ipums_var_label(nhgis_data$PMSA)

nhgis_data <- set_ipums_var_attributes(nhgis_data, codebook$var_info)
ipums_var_label(nhgis_data$PMSA)

# You can also load the codebook in raw format to display in the console
codebook_raw <- read_nhgis_codebook(nhgis_file, raw = TRUE)

# Use `cat` for human-readable output
cat(codebook_raw[1:20], sep = "\n")
```

save_extract_as_json *Store an extract definition in JSON format*

Description

Read and write an ipums_extract object to a JSON file that contains the extract definition specifications.

Use these functions to store a copy of an extract definition outside of your R environment and/or share an extract definition with another registered IPUMS user.

Learn more about the IPUMS API in vignette("ipums-api").

Usage

```
save_extract_as_json(extract, file, overwrite = FALSE)

define_extract_from_json(extract_json)
```

Arguments

extract	An ipums_extract object.
file	File path to which to write the JSON-formatted extract definition.
overwrite	If TRUE, overwrite file if it already exists. Defaults to FALSE.
extract_json	Path to a file containing a JSON-formatted extract definition.

Value

An ipums_extract object.

API Version Compatibility

As of v0.6.0, ipumsr only supports IPUMS API version 2. If you have stored an extract definition made using version beta or version 1 of the IPUMS API, you will not be able to load it using define_extract_from_json(). The API version for the request should be stored in the saved JSON file. (If there is no "api_version" or "version" field in the JSON file, the request was likely made under version beta or version 1.)

If the extract definition was originally made under your user account and you know its corresponding extract number, use get_extract_info() to obtain a definition compliant with IPUMS API version 2. You can then save this definition to JSON with save_extract_as_json().

Otherwise, you will need to update the JSON file to be compliant with IPUMS API version 2. In general, this should only require renaming all JSON fields written in snake_case to camelCase. For instance, "data_tables" would become "dataTables", "data_format" would become "dataFormat", and so on. You will also need to change the "api_version" field to "version" and set it equal to 2. If you are unable to create a valid extract by modifying the file, you may have to recreate the definition manually using the appropriate define_extract_*() function.

See the IPUMS developer documentation for more details on API versioning and breaking changes introduced in version 2.

See Also

define_extract_*() to define an extract request manually.

get_extract_info() to obtain a past extract to save.

submit_extract() to submit an extract request for processing.

add_to_extract() and remove_from_extract() to revise an extract definition.

Examples

```
my_extract <- define_extract_usa(
  description = "2013-2014 ACS Data",
  samples = c("us2013a", "us2014a"),
  variables = c("SEX", "AGE", "YEAR")
)

extract_json_path <- file.path(tempdir(), "usa_extract.json")
save_extract_as_json(my_extract, file = extract_json_path)

copy_of_my_extract <- define_extract_from_json(extract_json_path)

identical(my_extract, copy_of_my_extract)

file.remove(extract_json_path)
```

---

set_ipums_api_key *Set your IPUMS API key*

Description

Set your IPUMS API key as the value associated with the IPUMS_API_KEY environment variable.

The key can be stored for the duration of your session or for future sessions. If saved for future sessions, it is added to the .Renviron file in your home directory. If you choose to save your key to .Renviron, this function will create a backup copy of the file before modifying.

This function is modeled after the census_api_key() function from tidycensus.

Learn more about the IPUMS API in vignette("ipums-api").

Usage

```
set_ipums_api_key(api_key, save = overwrite, overwrite = FALSE, unset = FALSE)
```

Arguments

api_key	API key associated with your user account.
save	If TRUE, save the key for use in future sessions by adding it to the .Renviron file in your home directory. Defaults to FALSE, unless overwrite = TRUE.
overwrite	If TRUE, overwrite any existing value of IPUMS_API_KEY in the .Renviron file with the provided api_key. Defaults to FALSE.

unset — If TRUE, remove the existing value of IPUMS_API_KEY from the environment and the .Renviron file in your home directory.

Value

The value of api_key, invisibly.

See Also

set_ipums_default_collection() to set a default collection.

---

set_ipums_default_collection

Set your default IPUMS collection

Description

Set the default IPUMS collection as the value associated with the IPUMS_DEFAULT_COLLECTION environment variable. If this environment variable exists, IPUMS API functions that require a collection specification will use the value of IPUMS_DEFAULT_COLLECTION, unless another collection is indicated.

The default collection can be stored for the duration of your session or for future sessions. If saved for future sessions, it is added to the .Renviron file in your home directory. If you choose to save your key to .Renviron, this function will create a backup copy of the file before modifying.

This function is modeled after the census_api_key() function from tidycensus.

Learn more about the IPUMS API in vignette("ipums-api").

Usage

```
set_ipums_default_collection(
  collection = NULL,
  save = overwrite,
  overwrite = FALSE,
  unset = FALSE
)
```

Arguments

collection — Character string of the collection to set as your default collection. The collection must currently be supported by the IPUMS API.
For a list of codes used to refer to each collection, see ipums_data_collections().

save — If TRUE, save the default collection for use in future sessions by adding it to the .Renviron file in your home directory. Defaults to FALSE, unless overwrite = TRUE.

overwrite — If TRUE, overwrite any existing value of IPUMS_DEFAULT_COLLECTION in the .Renviron file with the provided collection. Defaults to FALSE.

unset — if TRUE, remove the existing value of IPUMS_DEFAULT_COLLECTION from the environment and the .Renviron file in your home directory.

Value

The value of `collection`, invisibly.

See Also

`set_ipums_api_key()` to set an API key.

Examples

```
set_ipums_default_collection("nhgis")

## Not run:
# Extract info will now be retrieved for the default collection:
get_last_extract_info()
get_extract_history()

is_extract_ready(1)
get_extract_info(1)

# Equivalent to:
get_extract_info("nhgis:1")
get_extract_info(c("nhgis", 1))

# Other collections can be specified explicitly
# Doing so does not alter the default collection
is_extract_ready("usa:2")

## End(Not run)

# Remove the variable from the environment and .Renviron, if saved
set_ipums_default_collection(unset = TRUE)
```

---

set_ipums_var_attributes

Add IPUMS variable attributes to a data frame

Description

Add variable attributes from an ipums_ddi object to a data frame. These provide contextual information about the variables and values contained in the data columns.

Most ipumsr data-reading functions automatically add these attributes. However, some data processing operations may remove attributes, or you may wish to store data in an external database that does not support these attributes. In these cases, use this function to manually attach this information.

Usage

```
set_ipums_var_attributes(
  data,
  var_info,
  var_attrs = c("val_labels", "var_label", "var_desc")
)
```

Arguments

data	tibble or data frame
var_info	An ipums_ddi object or a data frame containing variable information. Variable information can be obtained by calling ipums_var_info() on an ipums_ddi object.
var_attrs	Variable attributes from the DDI to add to the columns of the output data. Defaults to all available attributes.

Details

Attribute val_labels adds the haven_labelled class and the corresponding value labels for applicable variables. For more about the haven_labelled class, see vignette("semantics", package = "haven").

Attribute var_label adds a short summary of the variable's contents to the "label" attribute. This label is viewable in the RStudio Viewer.

Attribute var_desc adds a longer description of the variable's contents to the "var_desc" attribute, when available.

Variable information is attached to the data by column name. If column names in data do not match those found in var_info, attributes will not be added.

Value

data, with variable attributes attached

Examples

```
ddi_file <- ipums_example("cps_00157.xml")

# Load metadata into `ipums_ddi` object
ddi <- read_ipums_ddi(ddi_file)

# Load data
cps <- read_ipums_micro(ddi)

# Data includes variable metadata:
ipums_var_desc(cps$INCTOT)

# Some operations remove attributes, even if they do not alter the data:
cps$INCTOT <- ifelse(TRUE, cps$INCTOT, NA)
ipums_var_desc(cps$INCTOT)
```

```
# We can reattach metadata from the separate `ipums_ddi` object:
cps <- set_ipums_var_attributes(cps, ddi)
ipums_var_desc(cps$INCTOT)
```

`submit_extract` *Submit an extract request via the IPUMS API*

Description

Submit an extract request via the IPUMS API and return an ipums_extract object containing the extract definition with a newly-assigned extract request number.

Learn more about the IPUMS API in `vignette("ipums-api")`.

Usage

```
submit_extract(extract, api_key = Sys.getenv("IPUMS_API_KEY"))
```

Arguments

`extract`	An `ipums_extract` object.
`api_key`	API key associated with your user account. Defaults to the value of the `IPUMS_API_KEY` environment variable. See `set_ipums_api_key()`.

Value

An `ipums_extract` object containing the extract definition and newly-assigned extract number of the submitted extract.

Note that some unspecified extract fields may be populated with default values and therefore change slightly upon submission.

See Also

`wait_for_extract()` to wait for an extract to finish processing.

`get_extract_info()` and `is_extract_ready()` to check the status of an extract request.

`download_extract()` to download an extract's data files.

Examples

```
my_extract <- define_extract_cps(
  description = "2018-2019 CPS Data",
  samples = c("cps2018_05s", "cps2019_05s"),
  variables = c("SEX", "AGE", "YEAR")
)

## Not run:
# Store your submitted extract request to obtain the extract number
```

```
submitted_extract <- submit_extract(my_extract)

submitted_extract$number

# This is useful for checking the extract request status
get_extract_info(submitted_extract)

# You can always get the latest status, even if you forget to store the
# submitted extract request object
submitted_extract <- get_last_extract_info("cps")

# You can also check if submitted extract is ready
is_extract_ready(submitted_extract)

# Or have R check periodically and download when ready
downloadable_extract <- wait_for_extract(submitted_extract)

## End(Not run)
```

`wait_for_extract` *Wait for an extract request to finish processing*

Description

Wait for an extract request to finish by periodically checking its status via the IPUMS API until it is complete.

`is_extract_ready()` is a convenience function to check if an extract is ready to download without committing your R session to waiting for extract completion.

Learn more about the IPUMS API in `vignette("ipums-api")`.

Usage

```
wait_for_extract(
  extract,
  initial_delay_seconds = 0,
  max_delay_seconds = 300,
  timeout_seconds = 10800,
  verbose = TRUE,
  api_key = Sys.getenv("IPUMS_API_KEY")
)

is_extract_ready(extract, api_key = Sys.getenv("IPUMS_API_KEY"))
```

Arguments

`extract` One of:

- An `ipums_extract` object

- The data collection and extract number formatted as a string of the form `"collection:number"` or as a vector of the form `c("collection", number)`
- An extract number to be associated with your default IPUMS collection. See `set_ipums_default_collection()`

For a list of codes used to refer to each collection, see `ipums_data_collections()`.

`initial_delay_seconds`
: Seconds to wait before first status check. The wait time will automatically increase by 10 seconds between each successive check.

`max_delay_seconds`
: Maximum interval to wait between status checks. When the wait interval reaches this value, checks will continue to occur at `max_delay_seconds` intervals until the extract is complete or `timeout_seconds` is reached. Defaults to 300 seconds (5 minutes).

`timeout_seconds`
: Maximum total number of seconds to continue waiting for the extract before throwing an error. Defaults to 10,800 seconds (3 hours).

`verbose`
: If `TRUE`, print status updates to the R console at the beginning of each wait interval and upon extract completion. Defaults to `TRUE`.

`api_key`
: API key associated with your user account. Defaults to the value of the `IPUMS_API_KEY` environment variable. See `set_ipums_api_key()`.

Details

The `status` of a submitted extract will be one of `"queued"`, `"started"`, `"produced"`, `"canceled"`, `"failed"`, or `"completed"`.

To be ready to download, an extract must have a `"completed"` status. However, some requests that are `"completed"` may still be unavailable for download, as extracts expire and are removed from IPUMS servers after a set period of time (72 hours for microdata collections, 2 weeks for IPUMS NHGIS).

Therefore, these functions also check the `download_links` field of the extract request to determine if data are available for download. If an extract has expired (that is, it has completed but its download links are no longer available), these functions will warn that the extract request must be resubmitted.

Value

For `wait_for_extract()`, an `ipums_extract` object containing the extract definition and the URLs from which to download extract files.

For `is_extract_ready()`, a logical value indicating whether the extract is ready to download.

See Also

`download_extract()` to download an extract's data files.

`get_extract_info()` to obtain the definition of a submitted extract request.

Examples

```
my_extract <- define_extract_ipumsi(
  description = "Botswana data",
  samples = c("bw2001a", "bw2011a"),
  variables = c("SEX", "AGE", "YEAR")
)

## Not run:
submitted_extract <- submit_extract(my_extract)

# Wait for a particular extract request to complete by providing its
# associated `ipums_extract` object:
downloadable_extract <- wait_for_extract(submitted_extract)

# Or by specifying the collection and number for the extract request:
downloadable_extract <- wait_for_extract("ipumsi:1")

# If you have a default collection, you can use the extract number alone:
set_ipums_default_collection("ipumsi")

downloadable_extract <- wait_for_extract(1)

# Use `download_extract()` to download the completed extract:
files <- download_extract(downloadable_extract)

# Use `is_extract_ready()` if you don't want to tie up your R session by
# waiting for completion
is_extract_ready("usa:1")

## End(Not run)
```

---

zap_ipums_attributes *Remove label attributes from a data frame or labelled vector*

Description

Remove all label attributes (value labels, variable labels, and variable descriptions) from a data frame or vector.

Usage

```
zap_ipums_attributes(x)
```

Arguments

x	A data frame or labelled vector (for instance, from a data frame column)

Value

An object of the same type as x without "val_labels", "var_label", and "var_desc" attributes.

See Also

Other lbl_helpers: lbl_add(), lbl_clean(), lbl_define(), lbl_na_if(), lbl_relabel(), lbl()

Examples

```
cps <- read_ipums_micro(ipums_example("cps_00157.xml"))

attributes(cps$YEAR)
attributes(zap_ipums_attributes(cps$YEAR))

cps <- zap_ipums_attributes(cps)
attributes(cps$YEAR)
attributes(cps$INCTOT)
```

Index