Package 'ipumsr'

June 5, 2022

Title Read 'IPUMS' Extract Files

Version 0.5.0

Contact ipums@umn.edu

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

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

Description An easy way to import census, survey and geographic data provided by 'IPUMS' into R plus tools to help use the 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 our website <https://www.ipums.org>.

License Mozilla Public License 2.0

Encoding UTF-8

Depends R (>= 3.5.0)

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

RoxygenNote 7.2.0

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

VignetteBuilder knitr

NeedsCompilation no

Author Greg Freedman Ellis [aut], Derek Burk [aut, cre], Joe Grover [ctb], Finn Roberts [ctb], Minnesota Population Center [cph]

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

Repository CRAN

Date/Publication 2022-06-04 22:00:02 UTC

R topics documented:

`add_to_extract` *Add values to an IPUMS USA or CPS extract*

Description

Add new values to any fields of an IPUMS USA or CPS extract object. All fields are optional, and if omitted, will be unchanged. Supplying a value for fields that take a single value, such as `description` and `data_format`, will replace the existing value with the supplied value.

To remove existing values from an extract, see `remove_from_extract()`.

For an overview of ipumsr microdata API functionality, see `vignette("ipums-api", package = "ipumsr")`.

Usage

```
add_to_extract(extract, ...)

## S3 method for class 'usa_extract'
add_to_extract(
  extract,
  description = NULL,
  samples = NULL,
  variables = NULL,
  data_format = NULL,
  validate = TRUE,
  ...
)

## S3 method for class 'cps_extract'
add_to_extract(
  extract,
  description = NULL,
  samples = NULL,
  variables = NULL,
  data_format = NULL,
  validate = TRUE,
  ...
)
```

Arguments

extract	An ipums_extract object.
...	Further arguments passed to methods.
description	Description of the extract.
samples	Character vector of samples to add to the extract, if any. Use the USA sample ID values or the CPS sample ID values.
variables	Character vector of variables to add to the extract, if any.
data_format	The desired format of the extract data file (one of "fixed_width", "csv", "stata", "spss", or "sas9").
validate	Logical value indicating whether to check the modified extract structure for validity. Defaults to TRUE.

Value

A modified IPUMS USA or CPS extract object

Note

If the supplied extract definition comes from a previously submitted extract, this function will reset the definition to an unsubmitted state.

See Also

Other ipums_api: define_extract_cps(), define_extract_from_json(), define_extract_usa(), download_extract(), extract_list_to_tbl(), extract_tbl_to_list(), get_extract_info(), get_last_extract_info(), get_recent_extracts_info, ipums_data_collections(), is_extract_ready(), remove_from_extract(), save_extract_as_json(), set_ipums_api_key(), submit_extract(), wait_for_extract()

Examples

```
usa_extract <- define_extract_usa(
  description = "USA example",
  samples = "us2013a",
  variables = "YEAR"
)

revised_usa_extract <- add_to_extract(
  usa_extract,
  description = "Revised USA extract",
  samples = "us2014a"
)

revised_usa_extract


cps_extract <- define_extract_cps(
  description = "CPS example",
```

```
  samples = "cps2019_03s",
  variables = "YEAR"
)

revised_cps_extract <- add_to_extract(
  cps_extract,
  description = "Revised CPS extract",
  samples = "cps2020_03s"
)

revised_cps_extract
```

`define_extract_cps` *Define an IPUMS CPS extract request*

Description

Define an IPUMS CPS extract request to be submitted via the IPUMS microdata extract API. For an overview of ipumsr microdata API functionality, see `vignette("ipums-api", package = "ipumsr")`.

Usage

```
define_extract_cps(
  description,
  samples,
  variables,
  data_format = c("fixed_width", "csv", "stata", "spss", "sas9"),
  data_structure = "rectangular",
  rectangular_on = "P"
)
```

Arguments

`description`	Description of the extract.
`samples`	Character vector of samples to include in the extract. Samples should be specified using the sample ID values.
`variables`	Character vector of variables to include in the extract.
`data_format`	The desired format of the extract data file (one of "fixed_width", "csv", "stata", "spss", or "sas9").
`data_structure`	Currently, this must be "rectangular", which is also the default. In the future, the API will also support "hierarchical" extracts.
`rectangular_on`	Currently, this must be "P", indicating that the extract will be rectangularized on person records. In the future, the API will also support household-only extracts (`rectangular_on = "H"`).

Value

An object of class `c("cps_extract", "ipums_extract")` containing the extract definition.

See Also

Other ipums_api: `add_to_extract()`, `define_extract_from_json()`, `define_extract_usa()`, `download_extract()`, `extract_list_to_tbl()`, `extract_tbl_to_list()`, `get_extract_info()`, `get_last_extract_info()`, `get_recent_extracts_info`, `ipums_data_collections()`, `is_extract_ready()`, `remove_from_extract()`, `save_extract_as_json()`, `set_ipums_api_key()`, `submit_extract()`, `wait_for_extract()`

Examples

```
my_extract <- define_extract_cps("Example", "cps2020_03s", "YEAR")
```

`define_extract_from_json` *Create an* `ipums_extract` *object from a JSON-formatted definition*

Description

Create an `ipums_extract` object based on an extract definition formatted as JSON. For an overview of ipumsr microdata API functionality, see `vignette("ipums-api", package = "ipumsr")`.

Usage

```
define_extract_from_json(extract_json)
```

Arguments

`extract_json`	The path to a file containing the JSON definition, or a JSON string.

Value

An `ipums_extract` object.

See Also

Other ipums_api: `add_to_extract()`, `define_extract_cps()`, `define_extract_usa()`, `download_extract()`, `extract_list_to_tbl()`, `extract_tbl_to_list()`, `get_extract_info()`, `get_last_extract_info()`, `get_recent_extracts_info`, `ipums_data_collections()`, `is_extract_ready()`, `remove_from_extract()`, `save_extract_as_json()`, `set_ipums_api_key()`, `submit_extract()`, `wait_for_extract()`

Examples

```
my_extract <- define_extract_usa("Example", "us2013a", "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)
```

---

define_extract_usa *Define an IPUMS USA extract request*

Description

Define an IPUMS USA extract request to be submitted via the IPUMS microdata extract API. For an overview of ipumsr microdata API functionality, see `vignette("ipums-api", package = "ipumsr")`.

Usage

```
define_extract_usa(
  description,
  samples,
  variables,
  data_format = c("fixed_width", "csv", "stata", "spss", "sas9"),
  data_structure = "rectangular",
  rectangular_on = "P"
)
```

Arguments

`description`	Description of the extract.
`samples`	Character vector of samples to include in the extract. Samples should be specified using the sample ID values.
`variables`	Character vector of variables to include in the extract.
`data_format`	The desired format of the extract data file (one of "fixed_width", "csv", "stata", "spss", or "sas9").
`data_structure`	Currently, this must be "rectangular", which is also the default. In the future, the API will also support "hierarchical" extracts.
`rectangular_on`	Currently, this must be "P", indicating that the extract will be rectangularized on person records. In the future, the API will also support household-only extracts (`rectangular_on = "H"`).

Value

An object of class `c("usa_extract", "ipums_extract")` containing the extract definition.

See Also

Other ipums_api: `add_to_extract()`, `define_extract_cps()`, `define_extract_from_json()`, `download_extract()`, `extract_list_to_tbl()`, `extract_tbl_to_list()`, `get_extract_info()`, `get_last_extract_info()`, `get_recent_extracts_info`, `ipums_data_collections()`, `is_extract_ready()`, `remove_from_extract()`, `save_extract_as_json()`, `set_ipums_api_key()`, `submit_extract()`, `wait_for_extract()`

Examples

```
my_extract <- define_extract_usa("Example", "us2013a", "YEAR")
```

---

download_extract *Download an IPUMS data extract*

Description

Download an IPUMS data extract via the IPUMS API. For an overview of ipumsr microdata API functionality, see `vignette("ipums-api", package = "ipumsr")`.

Usage

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

Arguments

`extract`	One of: • An `ipums_extract` object • The data collection and extract number formatted as a single string of the form `"collection:number"` • The data collection and extract number formatted as a vector of the form `c("collection", "number")` The extract number does not need to be zero-padded (e.g., use `"usa:1"` or `c("usa", "1")`, not `"usa:00001"` or `c("usa", "00001")`). See Examples section below for examples of each form. For a list of codes used to refer to each collection, see `ipums_data_collections()`.
`download_dir`	In what folder should the downloaded files be saved? Defaults to current working directory.

overwrite	Logical indicating whether to overwrite files that already exist. Defaults to FALSE.
api_key	API key associated with your user account. Defaults to the value of environment variable "IPUMS_API_KEY".

Value

Invisibly, the path to the downloaded .xml DDI file.

See Also

Other ipums_api: add_to_extract(), define_extract_cps(), define_extract_from_json(), define_extract_usa(), extract_list_to_tbl(), extract_tbl_to_list(), get_extract_info(), get_last_extract_info(), get_recent_extracts_info, ipums_data_collections(), is_extract_ready(), remove_from_extract(), save_extract_as_json(), set_ipums_api_key(), submit_extract(), wait_for_extract()

Examples

```
my_extract <- define_extract_usa("Example", "us2013a", "YEAR")

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

# Download extract by supplying an ipums_extract object:
path_to_ddi_file <- download_extract(submitted_extract)

# By supplying the data collection and extract number, as a string:
path_to_ddi_file <- download_extract("usa:1")
# Note that there is no space before or after the colon, and no zero-padding
# of the extract number.

# By supplying the data collection and extract number, as a vector:
path_to_ddi_file <- download_extract(c("usa", "1"))

## End(Not run)
```

---

dplyr_select_style *Select-style helpers from dplyr*

Description

Several arguments in ipumsr allow syntax for selecting variables based on dplyr's select function. See details for more information.

Details

There are 3 broad categories of methods for specifying arguments for these select-style parameters.

- "Character Vector"A character vector of names (such as `c("var1", "var2", "var3")`)
- "'Bare' Vector"A vector of 'bare' names (such as `c(var1, var2, var3)`)
- "Helper Functions"Helper functions from `dplyr::select` such as `starts_with()`, `contains` and others.

Examples

```
# For microdata, use this syntax to load variables
# Load 3 variables by name
cps_file <- ipums_example("cps_00006.xml")
data <- read_ipums_micro(cps_file, vars = c("YEAR", "MONTH", "PERNUM"))

# Load same 3 variables using bare names
data <- read_ipums_micro(cps_file, vars = c(YEAR, MONTH, PERNUM))

# Use helper functions to load all variables that start with "WT"
data <- read_ipums_micro(cps_file, vars = starts_with("WT"))

# Use bare names and helper function to load YEAR, MONTH and all variables with 'INC' in name
data <- read_ipums_micro(cps_file, vars = c(YEAR, MONTH, contains("INC")))

# For geographic extracts, `data_layer` and `shape_layer` arguments use the same conventions
# to select file names from within zip files.
# (This extract only contains 1 type of file, but some have multiple)
csv_file <- ipums_example("nhgis0008_csv.zip")
data <- read_nhgis(
  csv_file,
  data_layer = contains("pmsa")
)
```

---

extract_list_to_tbl *Convert a list of extract definitions to a tibble*

Description

Convert a list of `ipums_extract` objects to a `tibble` in which each row contains the definition of one extract. For an overview of ipumsr microdata API functionality, see `vignette("ipums-api", package = "ipumsr")`.

Usage

```
extract_list_to_tbl(extract_list)
```

Arguments

extract_list	A list of ipums_extract objects.

Value

A tibble with number of rows equal to the length of extract_list, in which each row contains the definition of one extract.

See Also

Other ipums_api: add_to_extract(), define_extract_cps(), define_extract_from_json(), define_extract_usa(), download_extract(), extract_tbl_to_list(), get_extract_info(), get_last_extract_info(), get_recent_extracts_info, ipums_data_collections(), is_extract_ready(), remove_from_extract(), save_extract_as_json(), set_ipums_api_key(), submit_extract(), wait_for_extract()

Examples

```
## Not run:
# Get list of recent extracts
list_of_last_10_extracts <- get_recent_extracts_info_list("usa")

# Print the extract number for extracts that are downloadable:
for (extract in list_of_last_10_extracts) {
  if (is_extract_ready(extract)) print(extract$number)
}

# Convert list of extracts to tibble of extracts to view in a tabular format
extract_list_to_tbl(list_of_last_10_extracts)

## End(Not run)
```

---

extract_tbl_to_list *Convert a tibble of extract definitions to a list*

Description

Convert a tibble (or data.frame) of extract definitions, such as that returned by get_recent_extracts_info_tbl(), to a list of ipums_extract objects. For an overview of ipumsr microdata API functionality, see vignette("ipums-api", package = "ipumsr").

Usage

```
extract_tbl_to_list(extract_tbl, validate = TRUE)
```

Arguments

extract_tbl	A tibble (or data.frame) where each row contains the definition of one extract.
validate	Logical (TRUE or FALSE) value indicating whether to check that each row of extract_tbl contains a valid and complete extract definition. Defaults to TRUE.

Value

A list of length equal to the number of rows of extract_tbl.

See Also

Other ipums_api: add_to_extract(), define_extract_cps(), define_extract_from_json(), define_extract_usa(), download_extract(), extract_list_to_tbl(), get_extract_info(), get_last_extract_info(), get_recent_extracts_info, ipums_data_collections(), is_extract_ready(), remove_from_extract(), save_extract_as_json(), set_ipums_api_key(), submit_extract(), wait_for_extract()

Examples

```
## Not run:
# Get tibble of recent extracts
tbl_of_last_10_extracts <- get_recent_extracts_info_tbl("usa")

# Filter down to extracts with "income" in the description
description_mentions_income <- grepl(
  "[Ii]ncome",
  tbl_of_last_10_extracts$description
)
income_extracts <- tbl_of_last_10_extracts[description_mentions_income, ]

# Convert tibble of extracts to list of extracts
income_extracts <- extract_tbl_to_list(income_extracts)

# Now it's easier to operate on those elements as extract objects:
revised_income_extract <- add_to_extract(
  income_extracts[[1]],
  samples = "us2018a"
)

submitted_revised_income_extract <- submit_extract(revised_income_extract)

## End(Not run)
```

get_extract_info *Get information about a submitted extract*

Description

Get information about a submitted extract via the IPUMS API. For an overview of ipumsr microdata API functionality, see `vignette("ipums-api", package = "ipumsr")`.

Usage

```
get_extract_info(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 single string of the form `"collection:number"` • The data collection and extract number formatted as a vector of the form `c("collection", "number")` The extract number does not need to be zero-padded (e.g., use `"usa:1"` or `c("usa", "1")`, not `"usa:00001"` or `c("usa", "00001")`). See Examples section below for examples of each form. 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 environment variable "IPUMS_API_KEY".

Value

An `ipums_extract` object.

See Also

Other ipums_api: `add_to_extract()`, `define_extract_cps()`, `define_extract_from_json()`, `define_extract_usa()`, `download_extract()`, `extract_list_to_tbl()`, `extract_tbl_to_list()`, `get_last_extract_info()`, `get_recent_extracts_info`, `ipums_data_collections()`, `is_extract_ready()`, `remove_from_extract()`, `save_extract_as_json()`, `set_ipums_api_key()`, `submit_extract()`, `wait_for_extract()`

Examples

```
my_extract <- define_extract_usa("Example", "us2013a", "YEAR")

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

# Get info by supplying an ipums_extract object:
```

```
get_extract_info(submitted_extract)

# Get info by supplying the data collection and extract number, as a string:
get_extract_info("usa:1")
# Note that there is no space before or after the colon, and no zero-padding
# of the extract number.

# Get info by supplying the data collection and extract number, as a vector:
get_extract_info(c("usa", "1"))

## End(Not run)
```

`get_last_extract_info` *Get information on last extract*

Description

Get information on your most recent extract for a given IPUMS data collection, returned as an `ipums_extract` object. For an overview of ipumsr microdata API functionality, see `vignette("ipums-api", package = "ipumsr")`.

Usage

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

Arguments

`collection`	The code for an IPUMS data collection. For a list of the codes used to refer to the data collections, see `ipums_data_collections()`.
`api_key`	API key associated with your user account. Defaults to the value of environment variable "IPUMS_API_KEY".

Value

An `ipums_extract` object containing information on your most recent extract.

See Also

Other ipums_api: `add_to_extract()`, `define_extract_cps()`, `define_extract_from_json()`, `define_extract_usa()`, `download_extract()`, `extract_list_to_tbl()`, `extract_tbl_to_list()`, `get_extract_info()`, `get_recent_extracts_info`, `ipums_data_collections()`, `is_extract_ready()`, `remove_from_extract()`, `save_extract_as_json()`, `set_ipums_api_key()`, `submit_extract()`, `wait_for_extract()`

Examples

```
my_extract <- define_extract_usa("Example", "us2013a", "YEAR")

## Not run:
submit_extract(my_extract)

# Oops, forgot to capture the return object from submit_extract. Grab it with:
submitted_extract <- get_last_extract_info("usa")

# View the extract number
submitted_extract$number

# Check if submitted extract is ready
is_extract_ready(submitted_extract) # returns TRUE or FALSE

# Or have R check periodically until the extract is ready
downloadable_extract <- wait_for_extract(submitted_extract)

## End(Not run)
```

---

get_recent_extracts_info

Get information on recent extracts

Description

Get information on recent extracts for a given IPUMS collection via the IPUMS API, returned either as a list or tibble. For an overview of ipumsr microdata API functionality, see `vignette("ipums-api", package = "ipumsr")`.

Usage

```
get_recent_extracts_info_list(
  collection,
  how_many = 10,
  api_key = Sys.getenv("IPUMS_API_KEY")
)

get_recent_extracts_info_tbl(
  collection,
  how_many = 10,
  api_key = Sys.getenv("IPUMS_API_KEY")
)
```

Arguments

collection	The code for an IPUMS data collection. For a list of the codes used to refer to the data collections, see ipums_data_collections().
how_many	Number of recent extracts for which you'd like information. Defaults to 10 extracts.
api_key	API key associated with your user account. Defaults to the value of environment variable "IPUMS_API_KEY".

Value

For get_recent_extracts_info_list(), a list of extract objects. For get_recent_extracts_info_tbl(), a tibble with information on one extract in each row.

See Also

Other ipums_api: add_to_extract(), define_extract_cps(), define_extract_from_json(), define_extract_usa(), download_extract(), extract_list_to_tbl(), extract_tbl_to_list(), get_extract_info(), get_last_extract_info(), ipums_data_collections(), is_extract_ready(), remove_from_extract(), save_extract_as_json(), set_ipums_api_key(), submit_extract(), wait_for_extract()

Examples

```
## Not run:
# Get list of recent extracts
list_of_last_10_extracts <- get_recent_extracts_info_list("usa")

# Print the extract number for extracts that are downloadable:
for (extract in list_of_last_10_extracts) {
  if (is_extract_ready(extract)) print(extract$number)
}

# Get tibble of recent extracts
tbl_of_last_10_extracts <- get_recent_extracts_info_tbl("usa")

# Filter down to extracts with "income" in the description
description_mentions_income <- grepl(
  "[Ii]ncome",
  tbl_of_last_10_extracts$description
)
income_extracts <- tbl_of_last_10_extracts[description_mentions_income, ]

# Convert tibble of extracts to list of extracts
income_extracts <- extract_tbl_to_list(income_extracts)

# Now it's easier to operate on those elements as extract objects:
revised_income_extract <- add_to_extract(
  income_extracts[[1]],
  samples = "us2018a"
)
```

```
submitted_revised_income_extract <- submit_extract(revised_income_extract)

## End(Not run)
```

`ipums_bind_rows` *Bind rows together, but preserve labelled class attributes*

Description

Bind rows together, but preserve labelled class attributes

Usage

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

Arguments

`...`	Either data.frames or list of data.frames
`.id`	Data frame identifier, when arguments are named (or are named lists of data.frames), will make a new column with this name that has the original names.

Value

A data.frame

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

Description

Convenience wrapper around dplyr `collect` and `set_ipums_var_attributes`.

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`	A DDI object, read with `read_ipums_ddi`.
`var_attrs`	One or more of `val_labels`, `var_label` and `var_desc` describing what kinds of attributes you want to add. If NULL, will not add any attributes.

Value

A local `tbl_df` data.frame with IPUMS attributes attached

ipums_conditions *Get IPUMS citation and conditions*

Description

Gets information about citation and conditions from a DDI.

Usage

```
ipums_conditions(object = NULL)
```

Arguments

object	A DDI object (loaded with read_ipums_ddi). If NULL (the default), will use the conditions from the dataset you loaded most recently.

ipums_data_collections *List IPUMS data collections*

Description

List IPUMS data collections with corresponding codes used by the IPUMS API. Note that some data collections do not yet have API support. For an overview of ipumsr microdata API functionality, see `vignette("ipums-api", package = "ipumsr")`.

Usage

```
ipums_data_collections()
```

Value

A tibble with three columns containing the full collection name, the corresponding code used by the IPUMS API, and the status of API support for the collection.

See Also

Other ipums_api: add_to_extract(), define_extract_cps(), define_extract_from_json(), define_extract_usa(), download_extract(), extract_list_to_tbl(), extract_tbl_to_list(), get_extract_info(), get_last_extract_info(), get_recent_extracts_info, is_extract_ready(), remove_from_extract(), save_extract_as_json(), set_ipums_api_key(), submit_extract(), wait_for_extract()

Examples

```
# Print a tibble of all IPUMS data collections:
ipums_data_collections()
```

ipums_example *Get path to ipums example datasets*

Description

Get access to example extracts.

Usage

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

Arguments

path	Name of file. If 'NULL', the example files will be listed.

Value

The filepath to an example file, or if path is empty, a vector of all available files.

Examples

```
ipums_example() # Lists all available examples
ipums_example("cps_00006.xml") # Gives filepath for a cps DDI
```

ipums_extract-class *ipums_extract class*

Description

The ipums_extract class provides a data structure for storing the definition and status of a submitted or unsubmitted IPUMS data extract, for the purpose of interacting with the IPUMS extract API.

It is a superclass encompassing all of the collection-specific extract classes.

All objects with class ipums_extract will also have a collection-specific subclass (e.g. usa_extract, cps_extract) to accommodate collection-specific differences in extract options and contents, but all these subclasses share similarities as described below.

For an overview of ipumsr microdata API functionality, see vignette("ipums-api", package = "ipumsr").

Properties of ipums_extract

Objects of class ipums_extract have:

- A class attribute of the form c("<collection>_extract", "ipums_extract") (e.g. c("cps_extract", "ipums_extract")).
- A base type of "list".
- A names attribute that is a character vector the same length as the underlying list.

Behavior of ipums_extract

Objects of class ipums_extract:

- Can be created from scratch with a function that has a name of the form define_extract_<collection>() (e.g. define_extract_usa()).
- Can be created from existing extract definitions with functions define_extract_from_json() and get_extract_info().
- Can be submitted for processing with submit_extract(). After submission, you can have your R session periodically check the status of the submitted extract, and wait until it is ready to download, with wait_for_extract(). You can also check whether it is ready to download directly with is_extract_ready().
- Can be revised with add_to_extract() and remove_from_extract().
- Can be saved to a JSON-formatted file with save_extract_as_json().

---

ipums_file_info *Get IPUMS file information*

Description

Get IPUMS metadata information about the data file loaded into R from an ipums_ddi

Usage

```
ipums_file_info(object, type = NULL)
```

Arguments

object	An ipums_ddi object (loaded with read_ipums_ddi).
type	NULL to load all types, or one of "ipums_project", "extract_data", "extract_notes", "conditions" or "citation".

Value

If type is NULL, a list with the ipums_project, extract_date, extract_notes, conditions, and citation. Otherwise a string with the type of information requested in type.

Examples

```
ddi <- read_ipums_ddi(ipums_example("cps_00006.xml"))
ipums_file_info(ddi)
```

`ipums_list_files` *List files available for analysis in an IPUMS extract*

Description

Find which files can be loaded from an IPUMS extract. On Windows, this is generally a zip file (which you can optionally unzip). On macOS, they are generally unzipped for you, so there will be a directory.

Usage

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

ipums_list_data(file, data_layer = NULL)

ipums_list_shape(file, shape_layer = NULL)

ipums_list_raster(file, raster_layer = NULL)
```

Arguments

`file`	An IPUMS extract zip file or directory
`types`	One or more of "data", "shape", or "raster" indicating what type of files to look for.
`data_layer`	dplyr `select`-style notation for the data files to look for
`shape_layer`	dplyr `select`-style notation for the shape files to look for
`raster_layer`	dplyr `select`-style notation for the raster files to look for

Value

A `tbl_df` data.frame containing the files available

Examples

```
nhgis_file <- ipums_example("nhgis0008_csv.zip")
ipums_list_files(nhgis_file) # Only one extract available
```

`ipums_shape_left_join` *Join data to geographic boundaries*

Description

Helpers for joining shape files downloaded from the IPUMS website to data from extracts. Because of historical reasons, the attributes of (like variable type) of variables in the shape files does not always match those in the data 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 dataset, usually one that has been aggregated to a geographic level.
`shape_data`	A shape file (loaded with `read_ipums_sf` or `read_ipums_sp`)

by	A vector of variable names to join on. Like the dplyr join functions, named vectors indicate that the names are different between the data and shape file. shape files to load. Accepts a character vector specifying the file name, or dplyr_select_style conventions. Can load multiple shape files, which will be combined.
suffix	For variables that are found in both, but aren't joined on, a suffix to put on the variables. Defaults to nothing for data variables and "_SHAPE" for variables from the shape file.
verbose	I TRUE, will report information about geometries dropped in the merge.

Value

returns a sf or a SpatialPolygonsDataFrame depending on what was passed in.

Examples

```
# Note that these examples use NHGIS data so that they use the example data provided,
# but the functions read_nhgis_sf/read_nhgis_sp perform this merge for you.

data <- read_nhgis(ipums_example("nhgis0008_csv.zip"))

if (require(sf)) {
  sf <- read_ipums_sf(ipums_example("nhgis0008_shape_small.zip"))
  data_sf <- ipums_shape_inner_join(data, sf, by = "GISJOIN")
}

if (require(sp) && require(rgdal)) {
  sp <- read_ipums_sp(ipums_example("nhgis0008_shape_small.zip"))
  data_sp <- ipums_shape_inner_join(data, sp, by = "GISJOIN")
}

## Not run:
 # Sometimes variable names won't match between datasets (for example in IPUMS international)
  data <- read_ipums_micro("ipumsi_00004.xml")
  shape <- read_ipums_sf("geo2_br1980_2010.zip")
  data_sf <- ipums_shape_inner_join(data, shape, by = c("GEO2" = "GEOLEVEL2"))

## End(Not run)
```

---

ipums_var_info *Get IPUMS variable information*

Description

Get IPUMS metadata information about variables loaded into R. Will try to read the metadata from the loaded datasets, but it is more reliable to load the DDI into a separate object and use it instead.

Usage

```
ipums_var_info(object, vars = NULL)

ipums_var_desc(object, var = NULL)

ipums_var_label(object, var = NULL)

ipums_val_labels(object, var = NULL)
```

Arguments

object	A DDI object (loaded with read_ipums_ddi), a data.frame with ipums metadata attached, or a single column from an ipums data.frame.
vars	dplyr select-style notation for the variables to give information about
var	select-style notation for a single variable

ipums_var_info() loads all available variable information for one or more variables into a data.frame. If object is a vector, it will include the variable label, variable description and value labels. If object is a data.frame, it will include it for all variables (or only those specified by vars). If it is a DDI, it will also include information used to read the data from disk, including start/end position in the fixed-width file, implied decimals and variable type.

ipums_var_desc() loads the variable description for a single variable.

ipums_var_label() loads the short variable label for a single variable.

ipums_val_labels() loads the value labels for a single variable.

Note that many R functions drop attributes that provide this information. In order to make sure that they are available, it is best to keep a copy of the separate from the data your are manipulating using read_ipums_ddi. Then you can refer to the IPUMS documentation in this object.

Value

ipums_var_info returns a tbl_df data frame with variable information, and the other functions return a length 1 character vector.

Examples

```
ddi <- read_ipums_ddi(ipums_example("cps_00006.xml"))

ipums_var_info(ddi)
ipums_var_desc(ddi, MONTH)
ipums_var_label(ddi, MONTH)
ipums_val_labels(ddi, MONTH)
```

ipums_view *View a static webpage with variable information from a IPUMS extract*

Description

Requires that htmltools, shiny and DT are installed.

Usage

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

Arguments

x	A DDI or other object with ipums attributes (such as data loaded from an extract). Note that the file level information (like extract notes) are only available from the DDI.
out_file	Optionally specify a location to save HTML file. NULL the default makes a temporary file.
launch	Logical indicating whether to launch the website.

Value

The filepath to the html (silently if launch is TRUE)

Examples

```
ddi <- read_ipums_ddi(ipums_example("cps_00006.xml"))
## Not run:
ipums_view(ddi)
ipums_view(ddi, "codebook.html", launch = FALSE)

## End(Not run)
```

ipums_website *Launch a browser window to the ipums website*

Description

Takes a DDI (or you can specify a project directly) and a variable name, and makes a best guess at the URL for the variable's page on the IPUMS website. Note that NHGIS and TerraPop do not have accessible pages for variables.

Usage

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

Arguments

x	A DDI or empty (if specifying project)
var	A single variable name in a character vector
project	If not using a DDI (or object with a project attribute) A name of an IPUMS project, one of: "IPUMS-USA", "IPUMS-CPS", "IPUMS-International", "IPUMS-DHS", "ATUS-X", "AHTUS-X", "MTUS-X", "NHIS", "Higher Ed", "NHGIS", or "IPUMS Terra"
launch	If TRUE, launch the website.
verbose	If TRUE, message user if no variable specific websites are available
var_label	Sometimes the variable label is useful for finding the correct URL. Only needed if not passing in the ddi object.
homepage_if_missing	If TRUE, Return homepage if project does not provide variable specific web pages.

Details

Because some variables are constructed during the extract creation process, the URL may not always work unfortunately.

Value

The url to the page on ipums.org (silently if launch is TRUE)

Examples

```
ddi <- read_ipums_ddi(ipums_example("cps_00006.xml"))
ipums_website(ddi, "MONTH", launch = FALSE)

## Not run:
# Launches website
ipums_website(ddi, "MONTH")

## End(Not run)

# Can also specify project instead of using DDI
```

```
ipums_website(var = "RECTYPE", project = "IPUMS-CPS", launch = FALSE)
```

`is_extract_ready` *Is the extract ready to download?*

Description

This function uses the IPUMS API to check whether the given extract is ready to download, returning TRUE for extracts that are ready and FALSE for those that are not. For an overview of ipumsr microdata API functionality, see `vignette("ipums-api", package = "ipumsr")`.

Usage

```
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 single string of the form `"collection:number"` • The data collection and extract number formatted as a vector of the form `c("collection", "number")` The extract number does not need to be zero-padded (e.g., use `"usa:1"` or `c("usa", "1")`, not `"usa:00001"` or `c("usa", "00001")`). See Examples section below for examples of each form. 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 environment variable "IPUMS_API_KEY".

Details

This function checks the "download_links" element of the supplied extract to determine whether the extract files are available to download. The "status" of a submitted extract is one of "queued", "started", "produced", "canceled", "failed", or "completed". Only "completed" extracts can be ready to download, but not all "completed" extracts are ready to download, because extract files are subject to removal from the IPUMS servers 72 hours after they first become available. Completed extracts older than 72 hours will still have a "completed" status, but will return `FALSE` from `is_extract_ready()`, because the extract files are no longer available.

Value

A logical vector of length one.

See Also

Other ipums_api: add_to_extract(), define_extract_cps(), define_extract_from_json(), define_extract_usa(), download_extract(), extract_list_to_tbl(), extract_tbl_to_list(), get_extract_info(), get_last_extract_info(), get_recent_extracts_info, ipums_data_collections(), remove_from_extract(), save_extract_as_json(), set_ipums_api_key(), submit_extract(), wait_for_extract()

Examples

```
my_extract <- define_extract_usa("Example", "us2013a", "YEAR")

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

# Check if extract is ready by supplying an ipums_extract object:
is_extract_ready(submitted_extract)

# By supplying the data collection and extract number, as a string:
is_extract_ready("usa:1")
# Note that there is no space before or after the colon, and no zero-padding
# of the extract number.

# By supplying the data collection and extract number, as a vector:
is_extract_ready(c("usa", "1"))

## End(Not run)
```

---

join_failures *Report on observations dropped by a join*

Description

Helper for learning which observations were dropped from a dataset because they were not joined on.

Usage

```
join_failures(join_results)
```

Arguments

join_results	A dataset that has just been created by a shape join (like ipums_shape_left_join)

Value

returns a list of data.frames, where the first item (shape) is the observations dropped from the shape file and the second (data) is the observations dropped from the data.

lbl *Make a label placeholder object*

Description

Helper to make a placeholder for a label-value pair.

Usage

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

Arguments

...	Either one or two arguments, possibly named .val and .lbl. If a single unnamed value, represents the label, if 2 unnamed values, the first is the value and the second is the label.

Value

A `label_placeholder` object, useful in functions like `lbl_add`

See Also

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

Examples

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

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

lbl_add *Add labels for unlabelled values*

Description

Add labels for values that don't already have them.

Usage

```
lbl_add(x, ...)

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

Arguments

x	A labelled vector
...	Labels formed by lbl indicating the value and label to be added.
labeller	A function that takes a single argument of the values and returns the labels. Defaults to as.character. as_function, so also accepts quosure-style lambda functions. See examples for more details.
vals	Vector of values to be labelled. NULL, the default labels all values that are in the data, but aren't already labelled.

Value

A haven::labelled vector

See Also

Other lbl_helpers: lbl_clean(), lbl_collapse(), 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)
)

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

lbl_add_vals(x)
lbl_add_vals(x, ~paste0("$", .))
lbl_add_vals(x, vals = c(100, 200))
```

---

lbl_clean *Clean unused labels*

Description

Remove labels that do not appear in the data.

Usage

```
lbl_clean(x)
```

Arguments

x	A labelled vector

Value

A haven::labelled vector

See Also

Other lbl_helpers: `lbl_add()`, `lbl_collapse()`, `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)
```

`lbl_collapse` *Collapse labelled values to labels that already exist*

Description

Converts values to a new value based on their label and value in a `labelled` vector. If the newly assigned value does not match an already existing labelled value, the smallest value's label is used. Ignores any value that does not have a label.

Usage

```
lbl_collapse(x, .fun)
```

Arguments

`x`	A `labelled` vector
`.fun`	A function that takes .val and .lbl (the values and labels) and returns the values of the label you want to change it to. It is passed to a function similar to `as_function`, so also accepts quosure-style lambda functions (that use values .val and .lbl). See examples for more information.

Value

A haven::labelled vector

See Also

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

lbl_collapse(x, ~(.val %/% 10) * 10)
# Notice that 90 get's NIU from 99 even though 90 didn't have a label in original

lbl_collapse(x, ~ifelse(.val == 10, 11, .val))
# But here 10 is assigned 11's label

# You can also use the more explicit function notation
lbl_collapse(x, function(.val, .lbl) (.val %/% 10) * 10)

# Or even the name of a function
collapse_function <- function(.val, .lbl) (.val %/% 10) * 10
lbl_collapse(x, "collapse_function")
```

---

lbl_define *Define labels for an unlabelled vector*

Description

Creates a `labelled` vector from an unlabelled atomic vector using `lbl_relabel` syntax, which allows grouping multiple values into a single labelled value. Values not assigned a label will remain unlabelled.

Usage

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

Arguments

x	An unlabelled atomic vector
...	Two-sided formulas where the left hand side is a label placeholder (created with the `lbl` function) and the right hand side is a function that returns a logical vector that indicates which existing values should be assigned that labeled value. The right hand side is passed to a function similar to `as_function`, so also accepts quosure-style lambda functions (that use values .val and .lbl). See examples for more information.

Value

A haven::labelled vector

See Also

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

Examples

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

# Note that values not assigned a new labelled value remain unchanged
lbl_define(
  age,
  lbl(1, "Pre-college age") ~ .val < 18,
  lbl(2, "College age") ~ .val >= 18 & .val <= 22
)
```

---

lbl_na_if *Set labelled values to missing*

Description

Convert values to NA based on their label and value in a labelled vector. Ignores any value that does not have a label.

Usage

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

Arguments

x	A labelled vector
.predicate	A function that takes .val and .lbl (the values and labels) and returns TRUE or FALSE. It is passed to a function similar to as_function, so also accepts quosure-style lambda functions (that use values .val and .lbl). See examples for more information.

Value

A haven::labelled vector

See Also

Other lbl_helpers: lbl_add(), lbl_clean(), lbl_collapse(), 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)
)

lbl_na_if(x, ~.val >= 90)
lbl_na_if(x, ~.lbl %in% c("Maybe"))
lbl_na_if(x, ~.val >= 90 | .lbl %in% c("Maybe"))

# You can also use the more explicit function notation
lbl_na_if(x, function(.val, .lbl) .val >= 90)

# Or even the name of a function
na_function <- function(.val, .lbl) .val >= 90
lbl_na_if(x, "na_function")
```

`lbl_relabel` *Relabel labelled values*

Description

Converts values to a new value (that may or may not exist) based on their label and value in a `labelled` vector. Ignores any value that does not have a label.

Usage

```
lbl_relabel(x, ...)
```

Arguments

`x`	A `labelled` vector
`...`	Two-sided formulas where the left hand side is a label placeholder (created with the `lbl` function) or a value that already exists in the data and the right hand side is a function that returns a logical vector that indicates which labels should be relabeled. The right hand side is passed to a function similar to `as_function`, so also accepts quosure-style lambda functions (that use values .val and .lbl). See examples for more information.

Value

A haven::labelled vector

See Also

Other lbl_helpers: `lbl_add()`, `lbl_clean()`, `lbl_collapse()`, `lbl_define()`, `lbl_na_if()`, `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)
)

lbl_relabel(
  x,
  lbl(10, "Yes/Yes-ish") ~ .val %in% c(10, 11),
  lbl(90, "???") ~ .val == 99 | .lbl == "Maybe"
)

# If relabelling to labels that already exist, don't need to specify both label
# and value:
# If just bare, assumes it is a value:
lbl_relabel(x, 10 ~ .val == 11)
# Use single argument to lbl for the label
lbl_relabel(x, lbl("Yes") ~ .val == 11)
# Or can used named arguments
lbl_relabel(x, lbl(.val = 10) ~ .val == 11)
```

---

read_ipums_codebook *Read metadata from a text codebook in a NHGIS or Terra area-level extract*

Description

Read text formatted codebooks provided by some IPUMS extract systems such as NHGIS and Terra Area-level extracts in a format analogous to the DDIs available for other projects.

Usage

```
read_ipums_codebook(cb_file, data_layer = NULL)
```

Arguments

cb_file	Filepath to the codebook (either the .zip file directly downloaded from the website, or the path to the unzipped .txt file).
data_layer	dplyr select-style notation for uniquely identifying the data layer to load. Required for reading from .zip files for extracts with multiple files.

Value

A ipums_ddi object with information on the variables included in the csv file of a NHGIS extract.

See Also

Other ipums_metadata: read_ipums_ddi()

Examples

```
# Example NHGIS extract
nhgis_file <- ipums_example("nhgis0008_csv.zip")
ddi <- read_ipums_codebook(nhgis_file)
```

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

Description

Reads the metadata about an IPUMS extract from a DDI file into R. Includes information about variable and value labels, terms of usage for the data and positions for the fixed-width file.

Usage

```
read_ipums_ddi(ddi_file, data_layer = NULL, lower_vars = FALSE)
```

Arguments

`ddi_file`	Filepath to DDI xml file
`data_layer`	If ddi_file is an extract with multiple DDIs, dplyr `select`-style notation indicating which .xml data layer to load.
`lower_vars`	Logical indicating whether to convert variable names to lowercase (default is FALSE, in line with IPUMS conventions)

Value

An `ipums_ddi` object with metadata information.

See Also

Other ipums_metadata: `read_ipums_codebook()`

Examples

```
# Example extract DDI
ddi_file <- ipums_example("cps_00006.xml")
ddi <- read_ipums_ddi(ddi_file)
```

`read_ipums_micro` *Read data from an IPUMS extract*

Description

Reads a dataset downloaded from the IPUMS extract system. For IPUMS projects with microdata, it relies on a downloaded DDI codebook and a fixed-width file. Loads the data with value labels (using `labelled` format) and variable labels. See 'Details' for more information on how record types are handled by the ipumsr package.

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 filepath to a DDI xml file downloaded from the website, or a `ipums_ddi` object parsed by `read_ipums_ddi`
`vars`	Names of variables to load. Accepts a character vector of names, or `dplyr_select_style` conventions. For hierarchical data, the rectype id variable will be added even if it is not specified.
`n_max`	The maximum number of records to load.
`data_file`	Specify a directory to look for the data file. If left empty, it will look in the same directory as the DDI file.
`verbose`	Logical, indicating whether to print progress information to console.
`var_attrs`	Variable attributes to add from the DDI, defaults to adding all (val_labels, var_label and var_desc). See `set_ipums_var_attributes` for more details.

lower_vars — Only if reading a DDI from a file, a logical indicating whether to convert variable names to lowercase (default is FALSE, in line with IPUMS conventions). Note that this argument will be ignored if argument ddi is an ipums_ddi object rather than a file path. See read_ipums_ddi for converting variable names to lowercase when reading in the DDI.

Details

Some IPUMS projects have data for multiple types of records (eg Household and Person). When downloading data from many of these projects you have the option for the IPUMS extract system to "rectangularize" the data, meaning that the data is transformed so that each row of data represents only one type of record.

There also is the option to download "hierarchical" extracts, which are a single file with record types mixed in the rows. The ipumsr package offers two methods for importing this data.

read_ipums_micro loads this data into a "long" format where the record types are mixed in the rows, but the variables are NA for the record types that they do not apply to.

read_ipums_micro_list loads the data into a list of data frames objects, where each data frame contains only one record type. The names of the data frames in the list are the text from the record type labels without 'Record' (often 'HOUSEHOLD' for Household and 'PERSON' for Person).

Value

read_ipums_micro returns a single tbl_df data frame, and read_ipums_micro_list returns a list of data frames, named by the Record Type. See 'Details' for more information.

See Also

Other ipums_read: read_ipums_micro_chunked(), read_ipums_micro_yield(), read_ipums_sf(), read_nhgis(), read_terra_area(), read_terra_micro(), read_terra_raster()

Examples

```
# Rectangular example file
cps_rect_ddi_file <- ipums_example("cps_00006.xml")

cps <- read_ipums_micro(cps_rect_ddi_file)
# Or load DDI separately to keep the metadata
ddi <- read_ipums_ddi(cps_rect_ddi_file)
cps <- read_ipums_micro(ddi)

# Hierarchical example file
cps_hier_ddi_file <- ipums_example("cps_00010.xml")

# Read in "long" format and you get 1 data frame
cps_long <- read_ipums_micro(cps_hier_ddi_file)
head(cps_long)

# 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)

# Or you can use the \code{%<-%} operator from zeallot to unpack
c(household, person) %<-% read_ipums_micro_list(cps_hier_ddi_file)
head(person)
head(household)
```

`read_ipums_micro_chunked` *Read data from an IPUMS extract (in chunks)*

Description

Reads a dataset downloaded from the IPUMS extract system, but does so by reading a chunk, then applying your code to that chunk and then continuing, which can allow you to deal with data that is too large to store in your computer's RAM all at once.

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 filepath to a DDI xml file downloaded from the website, or a `ipums_ddi` object parsed by `read_ipums_ddi`
`callback`	An `ipums_callback` object, or a function that will be converted to an IpumsSideEffectCallback object.

chunk_size	An integer indicating how many observations to read in per chunk (defaults to 10,000). Setting this higher uses more RAM, but will usually be faster.
vars	Names of variables to load. Accepts a character vector of names, or dplyr_select_style conventions. For hierarchical data, the rectype id variable will be added even if it is not specified.
data_file	Specify a directory to look for the data file. If left empty, it will look in the same directory as the DDI file.
verbose	Logical, indicating whether to print progress information to console.
var_attrs	Variable attributes to add from the DDI, defaults to adding all (val_labels, var_label and var_desc). See set_ipums_var_attributes for more details.
lower_vars	Only if reading a DDI from a file, a logical indicating whether to convert variable names to lowercase (default is FALSE, in line with IPUMS conventions). Note that this argument will be ignored if argument ddi is an ipums_ddi object rather than a file path. See read_ipums_ddi for converting variable names to lowercase when reading in the DDI. Also 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 callback object

See Also

Other ipums_read: read_ipums_micro_yield(), read_ipums_micro(), read_ipums_sf(), read_nhgis(), read_terra_area(), read_terra_micro(), read_terra_raster()

Examples

```
# Select Minnesotan cases from CPS example (Note you can also accomplish
# this and avoid having to even download a huge file using the "Select Cases"
# functionality of the IPUMS extract system)
mn_only <- read_ipums_micro_chunked(
  ipums_example("cps_00006.xml"),
  IpumsDataFrameCallback$new(function(x, pos) {
    x[x$STATEFIP == 27, ]
  }),
  chunk_size = 1000 # Generally you want this larger, but this example is a small file
)

# Tabulate INCTOT average by state without storing full dataset in memory
library(dplyr)
inc_by_state <- read_ipums_micro_chunked(
  ipums_example("cps_00006.xml"),
  IpumsDataFrameCallback$new(function(x, pos) {
    x %>%
      mutate(
        INCTOT = lbl_na_if(
          INCTOT, ~.lbl %in% c("Missing.", "N.I.U. (Not in Universe).")
```

```
      ) %>%
    filter(!is.na(INCTOT)) %>%
    group_by(STATEFIP = as_factor(STATEFIP)) %>%
    summarize(INCTOT_SUM = sum(INCTOT), n = n(), .groups = "drop")
  }),
  chunk_size = 1000 # Generally you want this larger, but this example is a small file
) %>%
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_00010.xml"),
  IpumsSideEffectCallback$new(function(x, pos) {
   print(paste0(nrow(x$PERSON), " persons and ", nrow(x$HOUSEHOLD), " households in this chunk."))
  }),
  chunk_size = 1000 # Generally you want this larger, but this example is a small file
)

# Using the biglm package, you can even run a regression without storing
# the full dataset in memory
library(dplyr)
if (require(biglm)) {
  lm_results <- read_ipums_micro_chunked(
    ipums_example("cps_00015.xml"),
    IpumsBiglmCallback$new(
      INCTOT ~ AGE + HEALTH, # Simple regression (may not be very useful)
      function(x, pos) {
        x %>%
        mutate(
          INCTOT = lbl_na_if(
            INCTOT, ~.lbl %in% c("Missing.", "N.I.U. (Not in Universe).")
          ),
          HEALTH = as_factor(HEALTH)
        )
    }),
    chunk_size = 1000 # Generally you want this larger, but this example is a small file
  )
  summary(lm_results)
}
```

`read_ipums_micro_yield`

Read data from an IPUMS extract (in yields)

Description

Reads a dataset downloaded from the IPUMS extract system, but does so by returning an object that can read a group of lines at a time. This is a more flexible way to read data in chunks than the functions like `read_ipums_micro_chunked`, allowing you to do things like reading parts of multiple

files at the same time and resetting from the beginning more easily than with the chunked functions. **Note that while other `read_ipums_micro*` functions can read from .csv(.gz) or .dat(.gz) files, these functions can only read from .dat(.gz) 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 filepath to a DDI xml file downloaded from the website, or a `ipums_ddi` object parsed by `read_ipums_ddi`
`vars`	Names of variables to load. Accepts a character vector of names, or `dplyr_select_style` conventions. For hierarchical data, the rectype id variable will be added even if it is not specified.
`data_file`	Specify a directory to look for the data file. If left empty, it will look in the same directory as the DDI file.
`verbose`	Logical, indicating whether to print progress information to console.
`var_attrs`	Variable attributes to add from the DDI, defaults to adding all (val_labels, var_label and var_desc). See `set_ipums_var_attributes` for more details.
`lower_vars`	Only if reading a DDI from a file, a logical indicating whether to convert variable names to lowercase (default is FALSE, in line with IPUMS conventions). Note that this argument will be ignored if argument `ddi` is an `ipums_ddi` object rather than a file path. See `read_ipums_ddi` for converting variable names to lowercase when reading in the DDI.

Details

These functions return an IpumsYield R6 object which have the following methods:

- `yield(n = 10000)` A function to read the next 'yield' from the data, returns a 'tbl_df' (or list of 'tbl_df' for 'hipread_list_yield()') with up to n rows (it will return NULL if no rows are left, or all available ones if less than n are available).

- `reset()` A function to reset the data so that the next yield will read data from the start.
- `is_done()` A function that returns whether the file has been completely read yet or not.
- `cur_pos` A property that contains the next row number that will be read (1-indexed).

Value

A HipYield R6 object (See 'Details' for more information)

Super classes

`hipread::HipYield` -> `hipread::HipLongYield` -> `IpumsLongYield`

Methods

Public methods:

- `IpumsLongYield$new()`
- `IpumsLongYield$yield()`

Method `new()`:

Usage:

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

Method `yield()`:

Usage:

```
IpumsLongYield$yield(n = 10000)
```

Super classes

`hipread::HipYield` -> `hipread::HipListYield` -> `IpumsListYield`

Methods

Public methods:

- `IpumsListYield$new()`
- `IpumsListYield$yield()`

Method `new()`:

Usage:

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

Method `yield()`**:**

Usage:

```
IpumsListYield$yield(n = 10000)
```

See Also

Other ipums_read: `read_ipums_micro_chunked()`, `read_ipums_micro()`, `read_ipums_sf()`, `read_nhgis()`, `read_terra_area()`, `read_terra_micro()`, `read_terra_raster()`

Examples

```
# An example using "long" data
long_yield <- read_ipums_micro_yield(ipums_example("cps_00006.xml"))
# Get first 10 rows
long_yield$yield(10)
# Get 20 more rows now
long_yield$yield(20)
# See what row we're on now
long_yield$cur_pos
# Reset to beginning
long_yield$reset()
# Read the whole thing in chunks and count 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_00006.xml"))
list_yield$yield(10)
```

---

read_ipums_sf *Read boundary files from an IPUMS extract*

Description

Reads the boundary files from an IPUMS extract into R as simple features (sf) objects or SpatialPolygonsDataFrame (sp) objects.

Usage

```
read_ipums_sf(
  shape_file,
  shape_layer = NULL,
  vars = NULL,
  encoding = NULL,
  bind_multiple = TRUE,
  add_layer_var = NULL,
  verbose = TRUE
)

read_ipums_sp(
  shape_file,
  shape_layer = NULL,
  vars = NULL,
  encoding = NULL,
  bind_multiple = TRUE,
  add_layer_var = NULL,
  verbose = TRUE
)
```

Arguments

shape_file	Filepath to one or more .shp files, a .zip file from an IPUMS extract or a path to an unzipped folder.
shape_layer	For .zip extracts with multiple datasets, the name of the shape files to load. Accepts a character vector specifying the file name, or dplyr_select_style conventions. Can load multiple shape files, which will be combined.
vars	Which variables in the shape file's data to keep (NULL the default keeps all)
encoding	The text encoding to use when reading the shape file. Typically the defaults should read the data correctly, but for some extracts you may need to set them manually, but if funny characters appear in your data, you may need to. For microdata projects, the default NULL will look for a .cpg file to determine the encoding and if none is available, it will default to latin1. The NHGIS and the IPUMS Terra functions specify the encoding for those projects (latin1 and UTF-8 respectively).
bind_multiple	If TRUE, will combine multiple shape files found into a single object.
add_layer_var	Whether to add a variable named layer that indicates which shape_layer the data came from. NULL, the default, uses TRUE if more than 1 layer is found, and FALSE otherwise.
verbose	I TRUE, will report progress information

Value

read_ipums_sf returns a sf object and read_ipums_sp returns a SpatialPolygonsDataFrame.

See Also

Other ipums_read: read_ipums_micro_chunked(), read_ipums_micro_yield(), read_ipums_micro(), read_nhgis(), read_terra_area(), read_terra_micro(), read_terra_raster()

Examples

```
shape_file <- ipums_example("nhgis0008_shape_small.zip")
# If sf package is availble, can load as sf object
if (require(sf)) {
  sf_data <- read_ipums_sf(shape_file)
}

# If sp package is available, can load as SpatialPolygonsDataFrame
if (require(sp) && require(rgdal)) {
  sp_data <- read_ipums_sp(shape_file)
}
```

---

read_nhgis *Read data from an NHGIS extract*

Description

Reads a dataset downloaded from the NHGIS extract system. Relies on csv files (with or without the extra header row).

Usage

```
read_nhgis(
  data_file,
  data_layer = NULL,
  verbose = TRUE,
  var_attrs = c("val_labels", "var_label", "var_desc")
)

read_nhgis_sf(
  data_file,
  shape_file,
  data_layer = NULL,
  shape_layer = data_layer,
  shape_encoding = "latin1",
  verbose = TRUE,
  var_attrs = c("val_labels", "var_label", "var_desc")
)

read_nhgis_sp(
  data_file,
  shape_file,
```

```
  data_layer = NULL,
  shape_layer = data_layer,
  shape_encoding = "latin1",
  verbose = TRUE,
  var_attrs = c("val_labels", "var_label", "var_desc")
)
```

Arguments

data_file	Filepath to the data (either the .zip file directly downloaded from the website, the path to the unzipped folder, or the path to the unzipped .csv file directly).
data_layer	For .zip extracts with multiple datasets, the name of the data to load. Accepts a character vector specifying the file name, or dplyr_select_style conventions. Data layer must uniquely identify a dataset.
verbose	Logical, indicating whether to print progress information to console.
var_attrs	Variable attributes to add from the codebook, defaults to adding all (val_labels, var_label and var_desc). See set_ipums_var_attributes for more details.
shape_file	Filepath to the shape files (either the .zip file directly downloaded from the website, or the path to the unzipped folder, or the unzipped .shp file directly).
shape_layer	(Defaults to using the same value as data_layer) Specification of which shape files to load using the same semantics as data_layer. Can load multiple shape files, which will be combined.
shape_encoding	The text encoding to use when reading the shape file. Typically the defaults should read the data correctly, but for some extracts you may need to set them manually, but if funny characters appear in your data, you may need to. Defaults to "latin1" for NHGIS.

Value

read_nhgis returns a tbl_df with only the tabular data, read_nhgis_sf returns a sf object with data and the shapes, and read_nhgis_sp returns a SpatialPolygonsDataFrame with data and shapes.

See Also

Other ipums_read: read_ipums_micro_chunked(), read_ipums_micro_yield(), read_ipums_micro(), read_ipums_sf(), read_terra_area(), read_terra_micro(), read_terra_raster()

Examples

```
csv_file <- ipums_example("nhgis0008_csv.zip")
shape_file <- ipums_example("nhgis0008_shape_small.zip")

data_only <- read_nhgis(csv_file)

# If sf package is availble, can load as sf object
if (require(sf)) {
  sf_data <- read_nhgis_sf(csv_file, shape_file)
```

```
}

# If sp package is available, can load as SpatialPolygonsDataFrame
if (require(rgdal) && require(sp)) {
  sp_data <- read_nhgis_sp(csv_file, shape_file)
}
```

`read_terra_area` *Read data from an IPUMS Terra area extract*

Description

Reads a area-level dataset downloaded from the IPUMS Terra extract system.

Usage

```
read_terra_area(
  data_file,
  data_layer = NULL,
  ddi_file = NULL,
  cb_file = NULL,
  verbose = TRUE,
  var_attrs = c("val_labels", "var_label", "var_desc")
)

read_terra_area_sf(
  data_file,
  shape_file = NULL,
  data_layer = NULL,
  shape_layer = data_layer,
  shape_encoding = "UTF-8",
  ddi_file = NULL,
  cb_file = NULL,
  verbose = TRUE,
  var_attrs = c("val_labels", "var_label", "var_desc")
)

read_terra_area_sp(
  data_file,
  shape_file = NULL,
  data_layer = NULL,
  shape_layer = data_layer,
  shape_encoding = "UTF-8",
  ddi_file = NULL,
  cb_file = NULL,
  verbose = TRUE,
  var_attrs = c("val_labels", "var_label", "var_desc")
)
```

Arguments

data_file	Path to the data file, which can either be the .zip file directly downloaded from the IPUMS Terra website, path to the unzipped folder, or to the csv unzipped from the download.
data_layer	For .zip extracts with multiple datasets, the name of the data to load. Accepts a character vector specifying the file name, or dplyr_select_style conventions. Data layer must uniquely identify a dataset.
ddi_file	(Optional) If the download is unzipped, path to the .xml file which provides usage and citation information for extract.
cb_file	(Optional) If the download is unzipped, path to the .txt file which provides usage and citation information for extract.
verbose	Logical, indicating whether to print progress information to console.
var_attrs	Variable attributes to add from the DDI, defaults to adding all (val_labels, var_label and var_desc). See set_ipums_var_attributes for more details.
shape_file	(Optional) If the download is unzipped, path to the .zip, folder path or .shp file representing the the shape file. If only the data table is needed, can be set to FALSE to indicate not to load the shape file.
shape_layer	(Defaults to using the same value as data_layer) Specification of which shape files to load using the same semantics as data_layer. Can load multiple shape files, which will be combined.
shape_encoding	The text encoding to use when reading the shape file. Typically the defaults should read the data correctly, but for some extracts you may need to set them manually, but if funny characters appear in your data, you may need to. Defaults to "UTF-8" for IPUMS Terra.

Value

read_terra_area returns a tbl_df with the tabular data, read_terra_area_sf returns a sf object with tabular data and shapes, and read_terra_area_sp returns a SpatialPolygonsDataFrame with data and shapes.

See Also

Other ipums_read: read_ipums_micro_chunked(), read_ipums_micro_yield(), read_ipums_micro(), read_ipums_sf(), read_nhgis(), read_terra_micro(), read_terra_raster()

Examples

```
## Not run:
data <- read_terra_area("2553_bundle.zip")

## End(Not run)
```

`read_terra_micro` *Read data from an IPUMS Terra microdata extract*

Description

Reads a microdata dataset downloaded from the IPUMS Terra extract system.

Usage

```
read_terra_micro(
  data_file,
  ddi_file = NULL,
  data_layer = NULL,
  n_max = Inf,
  verbose = TRUE,
  var_attrs = c("val_labels", "var_label", "var_desc")
)
```

Arguments

`data_file`	Path to the data file, which can either be the .zip file directly downloaded from the IPUMS Terra website, a path to the unzipped version of that folder, or to the csv unzipped from the download.
`ddi_file`	(Optional) If the download is unzipped, path to the .xml file which provides usage and citation information for extract.
`data_layer`	For .zip extracts with multiple datasets, the name of the data to load. Accepts a character vector specifying the file name, or `dplyr_select_style` conventions. Data layer must uniquely identify a dataset.
`n_max`	Maximum number of observations to read from the data
`verbose`	Logical, indicating whether to print progress information to console.
`var_attrs`	Variable attributes to add from the DDI, defaults to adding all (val_labels, var_label and var_desc). See `set_ipums_var_attributes` for more details.

Value

`read_terra_micro` returns a `tbl_df` with the tabular data. Use `read_ipums_sf` or `read_ipums_sp` to read shape data out of a microdata Terra extract.

See Also

Other ipums_read: `read_ipums_micro_chunked()`, `read_ipums_micro_yield()`, `read_ipums_micro()`, `read_ipums_sf()`, `read_nhgis()`, `read_terra_area()`, `read_terra_raster()`

Examples

```
## Not run:
data <- read_terra_micro("2553_bundle.zip")

## End(Not run)
```

`read_terra_raster` *Read data from an IPUMS Terra raster extract*

Description

Read a single raster datasets downloaded from the IPUMS Terra extract system using `read_terra_raster`, or read multiple into a list using `read_terra_raster_list`.

Usage

```
read_terra_raster(data_file, data_layer = NULL, verbose = TRUE)

read_terra_raster_list(data_file, data_layer = NULL, verbose = TRUE)
```

Arguments

`data_file`	Filepath to the data (either the .zip file directly downloaded from the website, or the path to the unzipped .tiff file(s)).
`data_layer`	For .zip extracts with multiple raster datasets, the name of the data to load. Accepts a character vector specifying the file name, or `dplyr_select_style` conventions.
`verbose`	Logical, indicating whether to print progress information to console.

Value

For `read_terra_raster` A `raster` object, for `read_terra_raster_list` A list of raster objects.

See Also

Other ipums_read: `read_ipums_micro_chunked()`, `read_ipums_micro_yield()`, `read_ipums_micro()`, `read_ipums_sf()`, `read_nhgis()`, `read_terra_area()`, `read_terra_micro()`

Examples

```
## Not run:
data <- read_terra_raster("2552_bundle.zip", "LCDECIDOPZM2013.tiff")
data <- read_terra_raster_list("2552_bundle.zip", "ZM")

## End(Not run)
```

`remove_from_extract` *Remove values from an IPUMS USA or CPS extract*

Description

Remove existing values from fields of an IPUMS USA or CPS extract object. All fields are optional, and if omitted, will remain unchanged.

To add new values to an extract, see `add_to_extract()`.

For an overview of ipumsr microdata API functionality, see `vignette("ipums-api", package = "ipumsr")`.

Usage

```
remove_from_extract(extract, ...)

## S3 method for class 'usa_extract'
remove_from_extract(
  extract,
  samples = NULL,
  variables = NULL,
  validate = TRUE,
  ...
)

## S3 method for class 'cps_extract'
remove_from_extract(
  extract,
  samples = NULL,
  variables = NULL,
  validate = TRUE,
  ...
)
```

Arguments

`extract`	An `ipums_extract` object.
`...`	Further arguments passed to methods.
`samples`	Character vector of samples to remove from the extract, if any.
`variables`	Character vector of variables to remove from the extract, if any.
`validate`	Logical value indicating whether to check the modified extract structure for validity. Defaults to `TRUE`.

Value

A modified IPUMS USA or CPS extract object

Note

If the supplied extract definition comes from a previously submitted extract, this function will reset the definition to an unsubmitted state.

See Also

Other ipums_api: add_to_extract(), define_extract_cps(), define_extract_from_json(), define_extract_usa(), download_extract(), extract_list_to_tbl(), extract_tbl_to_list(), get_extract_info(), get_last_extract_info(), get_recent_extracts_info, ipums_data_collections(), is_extract_ready(), save_extract_as_json(), set_ipums_api_key(), submit_extract(), wait_for_extract()

Examples

```
usa_extract <- define_extract_usa(
  description = "USA example",
  samples = c("us2013a", "us2014a"),
  variables = "YEAR"
)

revised_usa_extract <- remove_from_extract(
  usa_extract,
  samples = "us2014a"
)

revised_usa_extract


cps_extract <- define_extract_cps(
  description = "CPS example",
  samples = c("cps2019_03s", "cps2020_03s"),
  variables = "YEAR"
)

revised_cps_extract <- remove_from_extract(
  cps_extract,
  samples = "cps2020_03s"
)

revised_cps_extract
```

---

save_extract_as_json *Save an ipums_extract to disk as JSON*

Description

Save an ipums_extract to a JSON-formatted file. For an overview of ipumsr microdata API functionality, see vignette("ipums-api", package = "ipumsr").

Usage

```
save_extract_as_json(extract, file)
```

Arguments

extract	An ipums_extract object.
file	File path at which to write the JSON-formatted extract definition.

Details

Note that this function only saves out the properties of an extract that are required to submit a new extract request, namely, the description, data structure, data format, samples, variables, and collection.

Value

The file path where the extract definition was written, invisibly.

See Also

Other ipums_api: add_to_extract(), define_extract_cps(), define_extract_from_json(), define_extract_usa(), download_extract(), extract_list_to_tbl(), extract_tbl_to_list(), get_extract_info(), get_last_extract_info(), get_recent_extracts_info, ipums_data_collections(), is_extract_ready(), remove_from_extract(), set_ipums_api_key(), submit_extract(), wait_for_extract()

Examples

```
my_extract <- define_extract_usa("Example", "us2013a", "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)
```

---

set_ipums_api_key *Set your IPUMS API key*

Description

Set your IPUMS API key for the duration of your session, or indefinitely by adding it to the file ".Renviron" in your home directory. In either case, this function works by assigning your API key as the value of the environment variable IPUMS_API_KEY. 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 the R package tidycensus.

Usage

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

Arguments

api_key	API key associated with your user account, formatted in quotes.
save	Do you want to save this value for future sessions by adding it to the file ".Renviron" in your home directory? Defaults to FALSE.
overwrite	Do you want to overwrite any existing value of IPUMS_API_KEY in the file ".Renviron"? Defaults to FALSE.

Value

The value of api_key, invisibly.

See Also

Other ipums_api: add_to_extract(), define_extract_cps(), define_extract_from_json(), define_extract_usa(), download_extract(), extract_list_to_tbl(), extract_tbl_to_list(), get_extract_info(), get_last_extract_info(), get_recent_extracts_info, ipums_data_collections(), is_extract_ready(), remove_from_extract(), save_extract_as_json(), submit_extract(), wait_for_extract()

---

set_ipums_var_attributes

Add IPUMS variable attributes to a data.frame

Description

Add variable attributes from an IPUMS DDI to the variables in a data.frame. This function is usually called automatically for you inside of the read_* functions (such as read_ipums_micro or read_nhgis), but they can be useful other times as well. For example, if you store the data in a database, you can store the data without attributes in the database and add them on after loading a subset into a data.frame.

Usage

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

Arguments

data	A data.frame
var_info	An ipums_ddi object or a data.frame with the variable information (equivalent to getting ipums_var_info on a DDI).
var_attrs	One or more of val_labels, var_label and var_desc describing what kinds of attributes you want to add. If NULL, will not add any attributes.

Details

Attribute val_labels adds the haven::labelled class attributes and the corresponding value labels for variables that have value labels.

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

Attribute var_desc Adds a longer summary of the variable's contents to the attribute "var_desc" when available.

Value

A tbl_df data.frame with data and IPUMS attributes

Examples

```
ddi_file <- ipums_example("cps_00006.xml")
ddi <- read_ipums_ddi(ddi_file)
cps <- read_ipums_micro(ddi, var_attrs = NULL) # Don't load with attributes

ipums_var_desc(cps$YEAR) # Not available

# But, we can add on attributes after loading
cps_with_attr <- set_ipums_var_attributes(cps, ddi)
ipums_var_desc(cps_with_attr$YEAR)
```

---

submit_extract *Submit an extract request via the IPUMS API*

Description

Given an ipums_extract object, submit an extract request via the IPUMS API, and return a modified copy of the extract object with the newly-assigned extract number. For an overview of ipumsr microdata API functionality, see vignette("ipums-api", package = "ipumsr").

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 environment variable "IPUMS_API_KEY".

Value

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

See Also

Other ipums_api: add_to_extract(), define_extract_cps(), define_extract_from_json(), define_extract_usa(), download_extract(), extract_list_to_tbl(), extract_tbl_to_list(), get_extract_info(), get_last_extract_info(), get_recent_extracts_info, ipums_data_collections(), is_extract_ready(), remove_from_extract(), save_extract_as_json(), set_ipums_api_key(), wait_for_extract()

Examples

```
my_extract <- define_extract_usa("Example", "us2013a", "YEAR")

## Not run:
# `submit_extract()` returns an ipums_extract object updated to include the
# extract number, so it is often useful to name the return object:
submitted_extract <- submit_extract(my_extract)

# If you didn't capture the return object of submit_extract for your most
# recent extract, you can recover that information with:
submitted_extract <- get_last_extract_info("usa")

# View the extract number
submitted_extract$number

# Check if submitted extract is ready
is_extract_ready(submitted_extract) # returns TRUE or FALSE

# Or have R check periodically until the extract is ready
downloadable_extract <- wait_for_extract(submitted_extract)

## End(Not run)
```

---

wait_for_extract *Wait for extract to finish*

Description

Wait for an extract to finish by periodically checking its status via the IPUMS API and returning when the extract is ready to download. For an overview of ipumsr microdata API functionality, see `vignette("ipums-api", package = "ipumsr")`.

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")
)
```

Arguments

`extract` One of:

- An `ipums_extract` object
- The data collection and extract number formatted as a single string of the form `"collection:number"`
- The data collection and extract number formatted as a vector of the form `c("collection", "number")`

The extract number does not need to be zero-padded (e.g., use `"usa:1"` or `c("usa", "1")`, not `"usa:00001"` or `c("usa", "00001")`). See Examples section below for examples of each form.

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

`initial_delay_seconds`
How many seconds to wait before first status check.

`max_delay_seconds`
Maximum seconds to wait between status checks. The function doubles the wait time after each check, but will cap the wait time at this maximum value (300 seconds, or 5 minutes, by default).

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

`verbose` If `TRUE`, the default, messages will be printed at the beginning of each wait interval with the current wait time, each time the status of the extract is checked, and when the extract is ready to download. Setting this argument to `FALSE` will silence these messages.

`api_key` API key associated with your user account. Defaults to the value of environment variable "IPUMS_API_KEY".

Value

An `ipums_extract` object containing the extract definition and the URLs from which to download extract files.

See Also

Other ipums_api: add_to_extract(), define_extract_cps(), define_extract_from_json(), define_extract_usa(), download_extract(), extract_list_to_tbl(), extract_tbl_to_list(), get_extract_info(), get_last_extract_info(), get_recent_extracts_info, ipums_data_collections(), is_extract_ready(), remove_from_extract(), save_extract_as_json(), set_ipums_api_key(), submit_extract()

Examples

```
my_extract <- define_extract_usa("Example", "us2013a", "YEAR")

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

# Wait for extract by supplying ipums_extract object:
downloadable_extract <- wait_for_extract(submitted_extract)

# By supplying the data collection and extract number, as a string:
downloadable_extract <- wait_for_extract("usa:1")
# Note that there is no space before or after the colon, and no zero-padding
# of the extract number.

# By supplying the data collection and extract number, as a vector:
downloadable_extract <- wait_for_extract(c("usa", "1"))

## End(Not run)
```

---

zap_ipums_attributes *Remove all IPUMS attributes from a variable (or all variables in a data.frame)*

Description

Helper to remove ipums attributes (including value labels from the labelled class, the variable label and the variable description). These attributes can sometimes get in the way of functions like the dplyr join functions so you may want to remove them.

Usage

```
zap_ipums_attributes(x)
```

Arguments

x	A variable or a whole data.frame to remove attributes from

Value

A variable or data.frame

See Also

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

Examples

```
cps <- read_ipums_micro(ipums_example("cps_00006.xml"))
annual_unemployment <- data.frame(YEAR = c(1962, 1963), unemp = c(5.5, 5.7))

# Avoids warning 'Column `YEAR` has different attributes on LHS and RHS of join'
cps$YEAR <- zap_ipums_attributes(cps$YEAR)
cps <- dplyr::left_join(cps, annual_unemployment, by = "YEAR")
```

Index