Package 'DDIwR'

June 21, 2022

Version 0.12

Date 2022-06-21

Title DDI with R

Depends R (>= 3.5.0)

Imports admisc (> 0.28), declared (> 0.16), tools, xml2, haven, readxl, writexl

Description
Useful functions for various DDI (Data Documentation Initiative) related inputs and outputs. Converts data files to and from SPSS, Stata, SAS, R and Excel, including user declared missing values.

License GPL (>= 3)

URL https://github.com/dusadrian/DDIwR

BugReports https://github.com/dusadrian/DDIwR/issues

NeedsCompilation no

Author Adrian Dusa [aut, cre, cph] (<https://orcid.org/0000-0002-3525-9253>)

Maintainer Adrian Dusa <dusa.adrian@unibuc.ro>

Repository CRAN

Date/Publication 2022-06-21 12:40:02 UTC

R topics documented:

About the DDIwR package

Useful functions for various DDI (Data Documentation Initiative) related outputs.

Description

This package provides various functions to read DDI based metadata documentation, and write dedicated setup files for R, SPSS, Stata and SAS to read an associated .csv file containing the raw data, apply labels for variables and values and also deal with the treatment of missing values.

It can also generate a DDI metadata file out of an R information object, which can be used to export directly to the standard statistical packages files (such as SPSS, Stata and SAS, or even Excel), using the versatile package **haven**. For R, the default object to store data and metadata is a `data.frame`, and labelled data are automatically coerced to class `declared`.

The research leading to the initial functions in this package has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 262608 (DwB - Data without Boundaries)

Details

Package:	DDIwR
Type:	Package
Version:	0.12
Date:	2022-06-21
License:	GPL (>= 2)

Author(s)

Authors:
Adrian Dusa
Department of Sociology
University of Bucharest
<dusa.adrian@unibuc.ro>

Maintainer:
Adrian Dusa

convert *Convert a dataset from one statistical software to another*

Description

This function converts (or transfers) between R, Stata, SPSS, SAS, Excel and DDI XML files. Unlike the regular import / export functions from packages **haven** or **rio**, this function uses the DDI standard as an exchange platform and facilitates a consistent conversion of the missing values.

Usage

```
convert(from, to = NULL, declared = TRUE, chartonum = FALSE, recode = TRUE,
        encoding = "UTF-8", csv = NULL, ...)
```

Arguments

`from`	A path to a file, or a data.frame object
`to`	Character, the name of a software package or a path to a specific file
`declared`	Logical, return the resulting dataset as a declared object
`chartonum`	Logical, recode character categorical variables to numerical categorical variables
`recode`	Logical, recode missing values
`encoding`	The character encoding used to read a file
`csv`	Path to the CSV file, if not embedded in XML file containing the DDI Codebook.
`...`	Additional parameters passed to exporting functions, see the Details section

Details

When the argument `to` specifies a certain statistical package (`"R"`, `"Stata"`, `"SPSS"`, `"SAS"`, `"XPT"`) or `"Excel"`, the name of the destination file will be idential to the one in the argument `from`, with an automatically added software specific extension.

SPSS portable file (with the extension `".por"`) can only be read, and SAS Transport files (with the extension `".xpt"`) can be both read and written.

Alternatively, the argument `to` can be specified as a path to a specific file, in which case the software package is determined from its file extension. The following extentions are currently recognized: `.xml` for DDI, `.rds` for R, `.dta` for Stata, `.sav` for SPSS, `.sas7bdat` for SAS, and `.xlsx` for Excel.

Additional parameters can be specified via the three dots argument `...`, that are passed to the respective functions from packages **haven** and **readxl**. For instance the function `write_dta()` has an additional argument called `version` when writing a Stata file.

The most important argument to consider is called `user_na`, part of the function `read_sav()`. Although it is defaulted to `FALSE` in package **haven**, in package **DDIwR** it is used as having the value of `TRUE`. Users who really want to deactivate it should explicitly specify `use_na = FALSE` in function `convert()`.

The same three dots argument is used to pass additional parameters to other functions in this package, for instance `exportDDI()` when converting to a DDI file. Its argument `embed` (activated by default) can be used to control embedding the data in the XML file. Deactivating it will create a CSV file in the same directory, using the same file name as the XML file.

When converting from DDI, if the dataset is not embedded in the XML file, the CSV file is expected to be found in the same directory as the DDI Codebook, and it should have the same file name as the

XML file. Alternatively, the path to the CSV file can be provided via the csv argument. Additional formal parameters of the function read.csv() can be passed via the same three dots ... argument.

The argument chartonum signals recoding character categorical variables, and employs the function recodeCharcat(). This only makes sense when recoding to Stata, which does not allow allocating labels for anything but integer variables.

If the argument to is left to NULL, the data is (invisibly) returned to the R enviroment. Conversion to R, either in the working space or as a data file, will result (by default) in a data frame containing declared labelled variables, as defined in package **declared**.

The current version reads and creates DDI Codebook version 2.5, with future versions to extend the functionality for DDI Lifecycle versions 3.x and link to the future package **DDI4R** for the UML model based version 4. It extends the standard DDI Codebook by offering the possibility to embed a CSV version of the raw data into the XML file containing the Codebook, into a notes child of the fileDscr component. This type of codebook is unique to this package and automatically detected when converting to another statistical software.

Converting the missing values to SAS is not tested, but it relies on the same package **haven** using the ReadStat C library. Should it not work, it is also possible to use a setup file produced by function setupfile() and run the commands manually.

The argument recode controls how missing values are treated. If the input file has SPSS like numeric codes, they will be recoded to extended (a-z) missing types when converting to Stata or SAS. If the input has Stata like extended codes, they will be recoded to SPSS like numeric codes.

The character encoding is usually passed to the corresponding functions from package **haven**. It can be set to NULL to reset at the default in that package.

Author(s)

Adrian Dusa

References

DDI - Data Documentation Initiative, see https://ddialliance.org/

See Also

setupfile, getMetadata, declared, labelled

Examples

```
## Not run:
# Assuming an SPSS file called test.sav is located in the working directory
# the following command will extract the metadata in a DDI Codebook and
# produce a test.xml file in the same directory
convert("test.sav", to = "DDI")

# It is possible to include the data in the XML file, using:
convert("test.sav", to = "DDI", embed = TRUE)

# To produce a Stata file:
convert("test.sav", to = "Stata")
```

```
# To produce an R file:
convert("test.sav", to = "R")

# To produce an Excel file:
convert("test.sav", to = "Excel")

## End(Not run)
```

`exportDDI` *Export to a DDI metadata file*

Description

This function creates a DDI Codebook version 2.5, XML file structure.

Usage

```
exportDDI(
    codebook, file = "", embed = TRUE, OS = "", indent = 4,
    monolang = FALSE, xmlang = "en", xmlns = "", ...
)
```

Arguments

`codebook`	A list object containing the metadata, or a path to a directory where these objects are located, for batch processing
`file`	either a character string naming a file or a connection open for writing. "" indicates output to the console
`embed`	Embed the CSV datafile in the XML file, if present
`OS`	The target operating system, for the eol - end of line character(s)
`indent`	Indent width, in number of spaces
`monolang`	Logical, monolang or multilingual document
`xmlang`	ISO two letter code for the language used in the DDI elements
`xmlns`	Character, namespace for the XML file (ignored if already present in the codebook object)
`...`	Other arguments, mainly for internal use

Details

The information object is essentially a list having two main list components:

- `fileDscr`, if the data is provided in a subcomponent named `datafile`

- `dataDscr`, having as many components as the number of variables in the (meta)data. For each variable, there should a mandatory subcomponent called `label` (that contains the variable's label) and, if the variable is of a categorical type, another subcomponent called `values`.

Additional informations about the variables can be specified as further subcomponents, combining DDI specific data but also other information that might not be covered by DDI:

- `measurement` is the equivalent of the specific DDI attribute `nature` of the `var` element, and it accepts these values: `"nominal"`, `"ordinal"`, `"interval"`, `"ratio"`, `"percent"`, and `"other"`.

- `type` is useful for multiple reasons. A first one, if the variable is numerical, is to differentiate between `discrete` and `contin` values of the attribute `intrvl` from the same DDI element `var`. Another reason is to help identifying pure string variables (containing text), when the subcomponent `type` is equal to `"char"`. It is also used for the subelement `varFormat` of the element `var`. Finally, another reason is to differentiate between pure categorical (`"cat"`) and pure numerical (`"num"`) variables, as well as mixed ones, among which `"numcat"` referring to a numerical variable with very few values (such as the number of children), for which it is possible to also produce a table of frequencies along the numerical summaries. There are also categorical variables that can be interpreted as numeric (`"catnum"`), such as a Likert type response scale with 7 values, where numerical summaries are also routinely performed along with the usual table of frequencies.

- `missing` is an important subcomponent, indicating which of the values in the variable are going to be treated as missing values, and it is going to be exported as the attribute `missing` of the DDI subelement `catgry`.

There are many more possible attributes and DDI elements to be added in the information object, future versions of this function will likely expand.

For the moment, only DDI codebook version 2.5 is exported, and DDI Lifecycle is planned for future releases.

Argument `xmlang` expects a two letter ISO country coding, for instance `"en"` to indicate English, or `"ro"` to indicate Romanian etc.

If the document is monolang, this argument is placed a single time for the entire document, in the attributes of the `codeBook` element. For multilingual documents, it is placed in the attributes of various other (sub)elements, for instance `abstract` as an obvious one, or the study title, name of the distributing institution, variable labels etc.

The argument `OS` can be either:
`"windows"` (default), or `"Windows"`, `"Win"`, `"win"`,
`"MacOS"`, `"Darwin"`, `"Apple"`, `"Mac"`, `"mac"`,
`"Linux"`, `"linux"`.

The end of line separator changes only when the target OS is different from the running OS.

The argument `indent` controls how many spaces will be used in the XML file, to indent the different subelements.

A small number of required DDI specific elements and attributes have generic default values but they may be specified using the three dots `...` argument. For the current version, these are: `IDNo`, `titl`, `agency`, `URI` (for the `holdings` element), `distrbtr`, `abstract` and `level` (for the `otherMat` element).

Value

An XML file containing a DDI version 2.5 metadata.

See Also

https://ddialliance.org/Specification/DDI-Codebook/2.5/XMLSchema/field_level_documentation.html

Examples

```
codeBook <- list(dataDscr = list(
ID = list(
    label = "Questionnaire ID",
    type = "num",
    measurement = "interval"
),
V1 = list(
    label = "Label for the first variable",
    labels = c(
        "No"             =  0,
        "Yes"            =  1,
        "Not applicable" = -97,
        "Not answered"   = -99),
    na_values = c(-99, -97),
    type = "cat",
    measurement = "nominal"
),
V2 = list(
    label = "Label for the second variable",
    labels = c(
        "Very little"    =  1,
        "Little"         =  2,
        "So, so"         =  3,
        "Much"           =  4,
        "Very much"      =  5,
        "Don't know"     = -98),
    na_values = c(-98),
    type = "cat",
    measurement = "ordinal"
),
V3 = list(
    label = "Label for the third variable",
    labels = c(
        "First answer"   = "A",
        "Second answer"  = "B",
        "Don't know"     = -98),
    na_values = c(-98),
    type = "cat",
    measurement = "nominal"
),
V4 = list(
    label = "Number of children",
    labels = c(
        "Don't know"     = -98,
        "Not answered"   = -99),
    na_values = c(-99, -98),
```

```
    type = "numcat",
    measurement = "ratio"
),
V5 = list(
    label = "Political party reference",
    type = "char",
    txt = "When the respondent indicated his political party reference, his/her
        open response was recoded on a scale of 1-99 with parties with a
        left-wing orientation coded on the low end of the scale and parties with
        a right-wing orientation coded on the high end of the scale. Categories
        90-99 were reserved miscellaneous responses."
)))

## Not run:
exportDDI(codeBook, file = "codebook.xml")

# using a namespace
exportDDI(codeBook, file = "codebook.xml", xmlns = "ddi")

## End(Not run)
```

`getMetadata` *Extract metadata information*

Description

Extract a list containing the variable labels, value labels and any available information about missing values.

Usage

```
getMetadata(x, save = FALSE, declared = TRUE, OS = "Windows", encoding = "UTF-8", ...)
```

Arguments

`x`	A path to a file, or a data frame object
`save`	Boolean, save an .R file in the same directory
`declared`	Logical, embed the data as a declared object
`OS`	The target operating system, for the eol - end of line separator, if saving the file
`encoding`	The character encoding used to read a file
`...`	Additional arguments for this function (internal uses only)

Details

This function reads an XML file containing a DDI codebook version 2.5, or an SPSS or Stata file and returns a list containing the variable labels, value labels, plus some other useful information.

It additionally attempts to automatically detect a type for each variable:

`cat`:	categorical variable using numeric values
`catchar`:	categorical variable using character values
`catnum`:	categorical variable for which numerical summaries can be calculated (ex. a 0...10 Likert response scale)
`num`:	numerical
`numcat`:	numerical variable with very few values (ex. number of children) for which a table of frequencies is possible in addition to frequencies

By default, this function extracts the metadata into an R list object, but when the argument `save` is activated, the argument `OS` (case insensitive) can be either:
`"Windows"` (default), or `"Win"`,
`"MacOS"`, `"Darwin"`, `"Apple"`, `"Mac"`,
`"Linux"`.

The end of line separator changes only when the target OS is different from the running OS.

For the moment, only DDI version 2.5 (Codebook) is supported, but DDI Lifecycle is planned to be implemented.

Value

An R list roughly equivalent to a DDI codebook, containing all variables, their corresponding variable labels and value labels, and (if applicable) missing values if imported and found.

Author(s)

Adrian Dusa

Examples

```
x <- data.frame(
    A = declared(
        c(1:5, -92),
        labels = c(Good = 1, Bad = 5, NR = -92),
        na_values = -92
    ),
    C = declared(
        c(1, -91, 3:5, -92),
        labels = c(DK = -91, NR = -92),
        na_values = c(-91, -92)
    )
)

getMetadata(x)$dataDscr
```

recodeCharcat *Recode character categorical variables*

Description

Recodes a character categorical variables to a numerical categorical variable.

Usage

```
recodeCharcat(x, ...)
```

Arguments

x	A character categorical variable
...	Other internal arguments.

Details

For this function, a categorical variable is something else than a base factor. It should be an object of class "declared", or an object of class "haven_labelled_spss", with a specific attribute called "labels" that stores the value labels.

Value

A numeric categorical variable of the same class as the input.

Examples

```
x <- declared(
    c(letters[1:5], -91),
    labels = c(Good = "a", Bad = "e", NR = -91),
    na_values = -91
)

recodeCharcat(x)
```

recodeValues *Recode missing values*

Description

A function to recode all missing values to either SPSS or Stata types, uniformly (re)using the same codes across all variables.

Usage

```
recodeValues(dataset, to = c("SPSS", "Stata"), dictionary = NULL, ...)
```

Arguments

dataset	A data frame
to	Software to recode missing values for
dictionary	A named vector, with corresponding SPSS values and Stata codes.
...	Other internal arguments.

Details

When a dictionary is not provided, it is automatically constructed from the available data and metadata, using negative numbers starting from -91 and up to 27 letters starting with "a".

If the dataset contains mixed variables with SPSS and Stata style missing values, unless otherwise specified in a dictionary it uses other codes than the existing ones.

For the SPSS type of missing values, the resulting variables are coerced to a declared labelled format.

Unlike SPSS, Stata does not allow labels for character values. Both cannot be transported from SPSS to Stata, it is either one or another. If labels are more important to preserve than original values (especially the information about the missing values), the argument `chartonum` replaces all character values with suitable, non-overlapping numbers and adjusts the labels accordingly.

If no labels are found in the metadata, the original values are preserved.

Value

A data frame with all missing values recoded consistently.

Examples

```
x <- data.frame(
    A = declared(
        c(1:5, -92),
        labels = c(Good = 1, Bad = 5, NR = -92),
        na_values = -92
    ),
    B = labelled(
        c(1:5, tagged_na('a')),
        labels = c(DK = tagged_na('a'))
    ),
    C = declared(
        c(1, -91, 3:5, -92),
        labels = c(DK = -91, NR = -92),
        na_values = c(-91, -92)
    )
)

#         A     B       C
```

```
# 1       1     1       1
# 2       2     2 NA(-91)
# 3       3     3       3
# 4       4     4       4
# 5       5     5       5
# 6 NA(-92) NA(a) NA(-92)


xrec <- recodeValues(x, to = "Stata")

#       A     B     C
# 1     1     1     1
# 2     2     2 NA(b)
# 3     3     3     3
# 4     4     4     4
# 5     5     5     5
# 6 NA(c) NA(a) NA(c)


attr(xrec, "dictionary")
#   b   c
# -91 -92


recodeValues(x, to = "Stata", dictionary = c(a = -91, b = -92))
#       A     B     C
# 1     1     1     1
# 2     2     2 NA(a)
# 3     3     3     3
# 4     4     4     4
# 5     5     5     5
# 6 NA(b) NA(a) NA(b)


recodeValues(x, to = "SPSS")
#         A       B       C
# 1       1       1       1
# 2       2       2 NA(-91)
# 3       3       3       3
# 4       4       4       4
# 5       5       5       5
# 6 NA(-92) NA(-93) NA(-92)


recodeValues(x, to = "SPSS", dictionary = c(a = -91))
#         A       B       C
# 1       1       1       1
# 2       2       2 NA(-91)
# 3       3       3       3
# 4       4       4       4
# 5       5       5       5
# 6 NA(-92) NA(-91) NA(-92)
```

`setupfile` *Create setup files for SPSS, Stata, SAS and R*

Description

This function creates a setup file, based on a list of variable and value labels.

Usage

```
setupfile(codeBook, file = "", type = "all", csv = "", recode = TRUE, OS = "", ...)
```

Arguments

`codeBook`	A list object containing the metadata, or a path to a directory where these objects are located, for batch processing
`file`	Character, the (path to the) setup file to be created
`type`	The type of setup file, can be: "SPSS", "Stata", "SAS", "R", or "all" (default)
`csv`	The original dataset, used to create the setup file commands, or a path to the directory where the .csv files are located, for batch processing
`recode`	Logical, recode missing values to extended .a-.z range
`OS`	The target operating system, for the eol - end of line character(s)
`...`	Other arguments, see Details below

Details

When the a path to a metadata directory is specified for the argument `codebook`, then next argument `file` is silently ignored and all created setup files are saved in a directory called "Setup Files" that (if not already found) is created in the working directory.

The argument `file` expects the name of the final setup file being saved on the disk. If not specified, the name of the object provided for the `codebook` argument will be used as a filename.

If `file` is specified, the argument `type` is automatically determined from the file's extension, otherwise when `type = "all"`, the function produces one setup file for each supported type.

If batch processing multiple files, the function will inspect all files in the provided directory, and retain only those with the extension `.R` or `.r` or DDI versions with the extension `.xml` or `.XML` (it will subsequently generate an error if the .R files do not contain an object list, or if the `.xml` files do not contain a DDI structured metadata file).

If the metadata directory contains a subdirectory called `"data"` or `"Data"`, it will match the name of the metadata file with the name of the `.csv` file (their names have to be *exactly* the same, regardless of their extension).

The `csv` argument can provide a data frame object produced by reading the `.csv` file, or a path to the directory where the `.csv` files are located. If the user doesn't provide something for this argument, the function will check the existence of a subdirectory called `data` in the directory where the metadata files are located.

In batch mode, the code starts with the argument `delim = ","`, but if the `.csv` file is delimited differently it will also try hard to find other delimiters that will match the variable names in the metadata file. At the initial version 0.1-0, the automatically detected delimiters include `";"` and `"\t"`.

The argument `OS` (case insensitive) can be either:
`"Windows"` (default), or `"Win"`,
`"MacOS"`, `"Darwin"`, `"Apple"`, `"Mac"`,
`"Linux"`.

The end of line character(s) changes only when the target OS is different from the running OS.

Value

A setup file to complement the imported raw dataset.

Examples

```
codeBook <- list(dataDscr = list(
ID = list(
    label = "Questionnaire ID",
    type = "num",
    measurement = "interval"
),
V1 = list(
    label = "Label for the first variable",
    labels = c(
        "No"             =  0,
        "Yes"            =  1,
        "Not applicable" = -97,
        "Not answered"   = -99),
    na_values = c(-99, -97),
    type = "cat",
    measurement = "nominal"
),
V2 = list(
    label = "Label for the second variable",
    labels = c(
        "Very little"    =  1,
        "Little"         =  2,
        "So, so"         =  3,
        "Much"           =  4,
        "Very much"      =  5,
        "Don't know"     = -98),
    na_values = c(-98),
    type = "cat",
    measurement = "ordinal"
),
V3 = list(
    label = "Label for the third variable",
    labels = c(
```

```
        "First answer"   = "A",
        "Second answer"  = "B",
        "Don't know"     = -98),
    na_values = c(-98),
    type = "cat",
    measurement = "nominal"
),
V4 = list(
    label = "Number of children",
    labels = c(
        "Don't know"     = -98,
        "Not answered"   = -99),
    na_values = c(-99, -98),
    type = "numcat",
    measurement = "ratio"
)))


## Not run:
# IMPORTANT:
# make sure to set the working directory to a directory with read/write permissions
# setwd("/path/to/read/write/directory")


setupfile(codeBook)


# if the csv data file is available
setupfile(codeBook, csv="/path/to/csv/file.csv")


# generating a specific type of setup file
setupfile(codeBook, file = "codeBook.do") # type = "Stata" is unnecessary


# other types of possible utilizations, using paths to specific files
# an XML file containing a DDI metadata object

setupfile("/path/to/the/metadata/file.xml", csv="/path/to/csv/file.csv")


# or in batch mode, specifying entire directories
setupfile("/path/to/the/metadata/directory", csv="/path/to/csv/directory")

## End(Not run)
```

Index