Package: DDIwR (via r-universe)

September 7, 2024

Title DDI with R

Version 0.18.4

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

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

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

License GPL (>= 3)

Depends R (>= 3.5.0)

Imports admisc (> 0.33), base64enc, declared (> 0.23), digest, tools, xml2, haven, readxl, writexl

Suggests knitr, labelled, rmarkdown, spelling, testthat (>= 3.0.0), tibble

Language en-US

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Config/testthat/edition 3

Repository https://dusadrian.r-universe.dev

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

RemoteRef HEAD

RemoteSha 986050858a1471626f3d1bb82cbcf92634ca809f

Contents

`convert` *Converts 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 other 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 identical 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, but not written.

The argument to can also 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, .xpt 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(). Defaulted to FALSE in package **haven**, in package **DDIwR** it is used as having the value of TRUE, and it can be deactivated by explicitly specifying user_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. One of 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 DDI .xml file generates unique IDs for all variables, if not already present in the attributes. These IDs are useful for newer versions of the DDI Codebook, for referencing purposes.

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 2.6, 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 serialized version of the R dataset into the XML file containing the Codebook, within a notes child of the fileDscr component. This type of generated codebook is unique to this package and automatically detected when converting to another statistical software. This will likely be replaced with a time insensitive text version.

Converting to SAS is experimental, and it relies on the same package **haven** that uses the ReadStat C library. The safest way to convert, which at the same time consistently converts the missing values, is to export the data to a CSV file and create a setup file produced by function setupfile() and run the commands manually.

Converting data from SAS is possible, however reading the metadata is also experimental (the current version of **haven** only partially imports the metadata). Either specify the path to the catalog

file using the argument `catalog_file` from the function `read_sas()`, or have the catalog file in the same directory as the data set, with the same file name and the extension `.sas7bcat`

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.

Converting to SPSS works with numerical and character labelled vectors, with or without labels. Date/Time variables are partially supported by package **haven**: either having such a variable with no labels and missing values, or if labels and missing values are declared the variable is automatically coerced to numeric, and users may have to make the proper settings in SPSS.

Value

An invisible R data frame, when the argument `to` is NULL.

Author(s)

Adrian Dusa

References

DDI - Data Documentation Initiative, see the DDI Alliance website.

See Also

`setupfile`, `getMetadata`, `declared`

Examples

```
## Not run:
# Assuming an SPSS file called test.sav is located in the working directory
# The following command imports the file into the R environment:
test <- convert("test.sav")

# 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")

# The data may be saved separately from the DDI file, using:
convert("test.sav", to = "DDI", embed = FALSE)

# 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)
```

DDI-children *Add/remove/change one or more children or attributes from a DDI Codebook attribute.*

Description

addChildren() adds one or more children to a standard DDI Codebook element (see makeElement), anyChildren() checks if an element has any children at all, hasChildren() checks if the element has specific children, indexChildren() returns the positions of the children among all containing children, and getChildren() extracts them. For attributes and content, there are dedicated functions to add*(), remove*() and change*().

Usage

```
addChildren(children, to, overwrite = TRUE, ...)

anyChildren(element)

getChildren(xpath, from, ...)

hasChildren(element, name)

indexChildren(element, name)

removeChildren(name, from, overwrite = TRUE, ...)

addContent(content, to, overwrite = TRUE)

changeContent(content, to, overwrite = TRUE)

removeContent(from, overwrite = TRUE)

addAttributes(attrs, to, overwrite = TRUE)

anyAttributes(element)

changeAttributes(attrs, from, overwrite = TRUE)

hasAttributes(element, name)

removeAttributes(name, from, overwrite = TRUE)
```

Arguments

children	A standard element of class "DDI", or a list of such elements.
to	A standard element of class "DDI".
overwrite	Logical, overwrite the original object in the parent frame.
...	Other arguments, mainly for internal use.
element	A standard element of class "DDI".
xpath	Character, a path to a DDI Codebook element.
from	A standard element of class "DDI".
name	Character, name(s) of specific child element / attribute.
content	Character, the text content of a DDI element.
attrs	A list of specific attribute names and values.

Details

Although an XML list generally allows for multiple contents, sometimes spread between the children elements, it is preferable to maintain a single content (eventually separated with carriage return characters for separate lines).

Arguments are unique, and can be changed by simply referring to their names.

Elements, however, can be repeated. For instance element `var` to describe variables, within the `dataDscr` (data description) sub-element in the `codeBook`. There are as many such `var` elements as the number of variables in the dataset, in which case it is not possible to change a specific `var` element by referring to its name. For this purpose, it is useful to extract the positions of all `var` elements to iterate through, which is the purpose of the function `indexChildren()`.

Future versions will allow deep manipulations of child elements using the `xpath` argument.

If more than one children, they should be grouped into a list.

Value

An invisible standard DDI element. Functions `any*()` and `has*()` return a logical (vector).

Author(s)

Adrian Dusa

---

exportDDI *Export a DDI Codebook to an XML file.*

Description

Create a DDI Codebook version 2.6, XML file structure.

Usage

```
exportDDI(codeBook, file = "", OS = "", indent = 2, ...)
```

Arguments

codeBook	A standard element of class "DDI".
file	either a character string naming a file or a connection open for writing. "" indicates output to the console
OS	The target operating system, for the eol - end of line character(s)
indent	Indent width, in number of spaces
...	Other arguments, mainly for internal use

Details

#' The information object is a codeBook DDI element having at least two main children:

- fileDscr, with the data provided as a sub-component named datafile
- dataDscr, having as many components as the number of variables in the (meta)data.

For the moment, only DDI codebook version 2.6 is exported, and DDI Lifecycle is planned for future releases.

A small number of required DDI specific elements and attributes have generic default values, if not otherwise specified in the codeBook list object. For the current version, these are: monolang, xmlang, IDNo, titl, agency, URI (for the holdings element), distrbtr, abstract and level (for the otherMat element).

The codeBook object is exported as provided, and it is the user's responsibility to test its validity against the XML schema. Most of these arguments help create the mandatory element stdyDscr, which cannot be harvested from the dataset. If this element is not already present, providing any of these arguments via the three dots ... gate, signal an automatic creation and inclusion with the values provided.

Argument xmlang expects a two letter ISO country coding, for instance "en" to indicate English, or "ro" to indicate Romanian etc. The original DDI Codebook attribute is called xml:lang, which for obvious reasons had to be renamed into this R function.

A logical argument monolang signal if the document is monolingual, in which case the attribute xmlang is placed a single time for the entire document in the codeBook element. For multilingual documents, xmlang should be placed in the attributes of various other (child) elements, for instance abstract, 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 sub-elements.

Value

An XML file containing a DDI version 2.6 metadata.

Author(s)

Adrian Dusa

See Also

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

Examples

```
## 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, encoding = "UTF-8", ignore = NULL, ...)
```

Arguments

x	A path to a file, or a data frame object
encoding	The character encoding used to read a file
ignore	Character, ignore DDI elements when reading from an XML file
...	Additional arguments for this function (internal use only)

Details

This function extracts the metadata from an R dataset, or alternatively it can read an XML file containing a DDI codebook version 2.6, or an SPSS or Stata file and returns a list containing the variable labels, value labels and information about the missing values.

If the input is a dataset, it will extract the variable level metadata (labels, missing values etc.). From a DDI XML file, it will import all metadata elements, the most expensive being the data description.

For the moment, only DDI 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)
```

---

makeCategories *Create the* catgry *elements for a particular variable*

Description

Utility function to create the catgry elements, as well as all necessary sub-elements (e.g. catValu, labl, varFormat) along with their associated XML attributes.

Usage

```
makeCategories(metadata)
```

Arguments

metadata	A list of two or three components: labels, na_values and/or na_range

Value

A list of standard catgry DDI elements.

Author(s)

Adrian Dusa

makeDataNotes *Create a* notes *element for the dataset.*

Description

Create the notes element to embed a serialized, gzip-ed version of the data in the fileDscr section of the codeBook.

Usage

```
makeDataNotes(data)
```

Arguments

data	An R dataframe.

Value

A standard notes DDI element.

Author(s)

Adrian Dusa

makeElement *Make a DDI Codebook element*

Description

Creates a standard DDI element.

Usage

```
makeElement(
  name,
  children = NULL,
  attributes = NULL,
  content = NULL,
  fill = FALSE,
  ...
)
```

Arguments

name	Character, a DDI Codebook element name.
children	A list of standard DDI codebook elements.
attributes	A vector of named values.
content	Character scalar.
fill	Logical, fill the element with arbitrary values for its mandatory children and attributes
...	Other arguments, see Details.

Details

The structure of a DDI element in R follows the usual structure of an XML node, as returned by the function `as_list()` from package **xml2**, with one additional (first) component named ".extra" to accommodate any other information that is not part of the DDI element.

In the DDI Codebook, most elements and their attributes are optional, but some are mandatory. In case of attributes, some become mandatory only if the element itself is present. The mandatory elements need to be present in the final version of the Codebook, to pass the validation against the XML schema.

By activating the argument `fill`, this function creates DDI elements containing all mandatory (sub)elements and (their) attributes, filled with arbitrary values that can be changed later on. Some recommended elements are also filled, as expected by the CESSDA Data Catalogue profile for DDI Codebook.

By default, the Codebook is assumed to have a single language for all elements. The argument `monolang` can be deactivated through the `"..."` gate, in which situation the appropriate elements will receive a default argument `xmlang = "en"`. For other languages, that argument can also be provided through the `"..."` gate.

One such DDI Codebook element is the `stdyDscr` (Study Description), with the associated mandatory children, for instance title, ID number, distributor, citation, abstract etc.

Value

A standard list element of class `"DDI"` with reserved component names.

Author(s)

Adrian Dusa

See Also

`addChildren` `getChildren` `showDetails`

Examples

```
stdyDscr <- makeElement("stdyDscr", fill = TRUE)

# easier to extract with:
getChildren("stdyDscr/citation/titlStmt/titl", from = stdyDscr)
```

`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.

Author(s)

Adrian Dusa

Examples

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

recodeCharcat(x)
```

`recodeMissings` *Consistent recoding of (extended) 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

```
recodeMissings(
  dataset,
  to = c("SPSS", "Stata", "SAS"),
  dictionary = NULL,
  start = -91,
  ...
)
```

Arguments

`dataset`	A data frame
`to`	Software to recode missing values for
`dictionary`	A named vector, with corresponding Stata missing codes to SPSS missing values
`start`	A named vector, with corresponding Stata missing codes to SPSS missing values
`...`	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.

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
    ),
    B = labelled(
        c(1:5, haven::tagged_na('a')),
        labels = c(DK = haven::tagged_na('a'))
    ),
    C = declared(
        c(1, -91, 3:5, -92),
        labels = c(DK = -91, NR = -92),
        na_values = c(-91, -92)
    )
)

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

attr(xrec, "dictionary")

dictionary <- data.frame(
    old = c(-91, -92, "a"),
    new = c("c", "d", "c")
)
recodeMissings(x, to = "Stata", dictionary = dictionary)

recodeMissings(x, to = "SPSS")

dictionary$new <- c(-97, -98, -97)

recodeMissings(x, to = "SPSS", dictionary = dictionary)

recodeMissings(x, to = "SPSS", start = 991)

recodeMissings(x, to = "SPSS", start = -8)
```

---

searchFor *Search for key words*

Description

Search function to return elements that contain a certain word of regular expression pattern.

Usage

```
searchFor(
  word,
  where = c("everywhere", "description", "examples", "attributes")
)
```

Arguments

word	Character, either a word or a regular expression.
where	Character, in which sections to search for.

Value

Character vector of DDI element names.

Author(s)

Adrian Dusa

---

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

Description

Creates a setup file, based on a list of variable and value labels.

Usage

```
setupfile(
  obj,
  file = "",
  type = "all",
  csv = NULL,
  recode = TRUE,
  OS = "",
  stringnum = TRUE,
  ...
)
```

Arguments

obj	A data frame, or a list object containing the metadata, or a path to a data file or to a directory where such 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)
stringnum	Logical, recode string variables to numeric
...	Other arguments, see Details below

Details

When a path to a metadata directory is specified for the argument obj, 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 obj 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.

Author(s)

Adrian Dusa

Examples

```
## 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" also works


# 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)
```

---

showDetails *Describe what a DDI element is*

Description

Describe what a DDI element is

Usage

```
showDetails(x, ...)

showDescription(x, ...)

showAttributes(x, name = NULL, ...)

globalAttributes()
```

```
showExamples(x, ...)

showRelations(x, ...)

showLineages(x, ...)
```

Arguments

x	Character, a DDI Codebook element name.
...	Other arguments, mainly for internal use.
name	Character, print only a specific element (name)

Details

All arguments having predefined values such as "(Y | N) : N" are mandatory if the element is used

Author(s)

Adrian Dusa

Examples

```
showDetails("codeBook")

showAttributes("catgry")

showExamples("abstract")

showLineages("titl")
```

---

testValid *Validate a DDI element.*

Description

Attempts a minimal validation of a DDI Codebook element, by searching for mandatory elements and attributes.

Usage

```
testValid(element, monolang = TRUE)
```

Arguments

element	A standard element of class "DDI".
monolang	Logical, the codebook file is monolingual

Details

This function currently attempts a minimal check for the absolute most mandatory elements, such as the `stdyDscr`. An absolute bare version of this element, filled with arbitrary default values, can be produced with the function `makeElement()`, activating its attribute `fill`. It also checks for chained expectations, that is element X is mandatory only if the parent element is present.

Future versions will implement more functionality for recommended elements and attributes, with the intention to provide a 1:1 validation as offered by the "CESSDA Metadata Validator".

To ease the validation of the DDI Codebook XML files, the argument `monolang` is activated by default. This means a single attribute `xmlang` in the main `codeBook` element. For multi-language codebooks, an error is flagged if this argument is missing where appropriate.

Value

A character vector of validation problems found.

Author(s)

Adrian Dusa

See Also

`makeElement`

`updateCodebook` *Update Codebook.*

Description

Update an XML file containing a DDI Codebook.

Usage

```
updateCodebook(xmlfile, with, ...)
```

Arguments

`xmlfile`	A path to a DDI Codebook XML document.
`with`	An R object containing a root `codeBook` element.
`...`	Other internal arguments.

Details

This function replaces entire Codebook sections. Any such section present in the R object will replace the corresponding section from the XML document.

Author(s)

Adrian Dusa

Index