

Package: DDIwR (via r-universe)

October 15, 2026

Title DDI with R

Version 0.22.1

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 (>= 4.1.0)

Imports admisc (> 0.36), base64enc, declared (> 0.25), digest, tools, xml2, readxl, writexl

Suggests spelling, testthat (>= 3.0.0)

Language en-US

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libxml2-dev zlib1g-dev

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

Date/Publication 2026-10-15 06:10:37 UTC

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

RemoteRef HEAD

RemoteSha c4ec7adebb1913402d7d77500f6aa265f87f02c4

Contents

buildDictionary	2
convert	3
DDI-children	7
exportCodebook	9
getCodebook	11
makeCategories	12
makeDataNotes	13
makeElement	14
recodeCharcat	15
recodeMissings	16
searchFor	18
setupfile	19
showDetails	21
testValid	22
updateCodebook	23
updateSchema	24
Index	25

buildDictionary	<i>Build a dictionary of missing-value recodes</i>
-----------------	--

Description

Build a dataset-level dictionary of missing values and their recoded targets for internal SPSS-style normalization or export-oriented Stata/SAS mapping.

Usage

```
buildDictionary(dataset, to = c("SPSS", "Stata", "SAS"), start = -91)
```

Arguments

- | | |
|---------|--|
| dataset | A data frame. |
| to | Target software for the recoding dictionary. |
| start | Starting code for generated SPSS-style numeric missings. |

Value

A data frame with columns such as label, old, new, count, and n_variables.

 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,
  charnum = FALSE,
  recode = TRUE,
  encoding = "UTF-8",
  csv = NULL,
  n_max = -1L,
  skip = 0L,
  ...
)
```

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
charnum	Logical, recode character categorical variables to numerical categorical variables
recode	Logical, recode missing values
encoding	The character encoding used to read a file
csv	Complex argument, see the Details section
n_max	Integer, maximum number of rows to import from SPSS, Stata or SAS files
skip	Integer, number of rows to skip when importing from SPSS, Stata or SAS files
...	Additional options specific to the source or destination format, including language when importing translated DDI labels; 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.

SPSS system files (.sav and .zsav) preserve custom dataset and variable attributes. They are stored in the R attribute `spss.attributes` on the data frame and on each column, as a named list of character vectors. For example, `attr(data, "spss.attributes") <- list(source = "Archive")` and `attr(data$answer, "spss.attributes") <- list(foo = "bar")` are written as SPSS dataset and variable attributes, respectively. A character vector with several elements represents an attribute array; a single element represents a scalar. Custom names such as `label` remain separate from standard R metadata. Names must be valid SPSS attribute names of at most 64 UTF-8 bytes, unique ignoring case within their scope. Values must be non-missing strings without line feeds, which the SPSS attribute format cannot represent.

Stata dataset and variable characteristics are preserved in .dta files through the R attribute `stata.characteristics` on the data frame and each column. Use a named list of single, non-missing strings, for example `attr(data, "stata.characteristics") <- list(Source = "Archive")` and `attr(data$answer, "stata.characteristics") <- list(Foo = "bar")`. Characteristic names are case sensitive and may contain at most 32 characters. Values may contain line feeds and trailing spaces, and may use at most 67783 bytes in the target encoding (67784 including the terminating zero byte). Stata 14 and later use UTF-8. Earlier versions write characteristic values in Windows-1252 and require ASCII characteristic names; unrepresentable values are rejected. When importing these older files with `convert()`, specify `encoding = "windows-1252"` (or their actual source encoding). Dataset and variable note characteristics, including `note0`, are retained. Raw dataset note characteristics take precedence over the R notes convenience attribute when writing, preserving note numbering and contents. If no raw note characteristics are present, dataset notes are written as Stata note characteristics.

Stata export converts R Date columns to days since 1960-01-01 and POSIXct columns to milliseconds since that date. Integer storage is used only when the converted values fit its range and precision. System and extended missing values are preserved separately from the date/time values.

The argument `can` can also be specified as a path to a specific file, in which case the software package is determined from its file extension. Home-directory paths beginning with `~` are supported. DDI exports include the required study description with generic defaults. Supply `titl`, `IDNo`, `agency`, `distrbtr`, `URI` and `abstract` through `...` to describe the actual study. XML exports are validated against the bundled DDI Codebook 2.6 schema before writing. Invalid XML causes an error and leaves destination files unchanged. The following extensions 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 formal arguments `n_max` and `skip` are used only when importing foreign data files through the direct ReadStat readers, namely SPSS (.sav, .zsav, .por), Stata (.dta) and SAS (.sas7bdat, .xpt). They can be used to read only the first `n_max` rows or to import large files in batches by skipping the first `skip` rows.

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 `exportCodebook()` when writing 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. 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. Alternatively, the `csv` argument can also be an R data frame.

A multilingual DDI Codebook is imported in one language at a time. The DDI-specific language option is supplied via `...`, for example `convert("codebook.xml", language = "en")`. When it is omitted, the effective `xml:lang` of the first variable label is used. Supplying it selects that translation throughout the variable and category labels. Language declarations inherited from a parent DDI element are honored. An unqualified label is used as a neutral fallback, but a label in a different language is not substituted.

When converting to DDI, if the argument `embed` is set to `FALSE`, users have the option to save the data in a separate CSV file (the default) or not to save the data at all, by setting `csv` to `FALSE`.

The DDI .xml file generates unique IDs for all variables, if not already present in the attributes. These IDs are useful for referencing purposes, in newer versions of the DDI Codebook.

The argument `char tonum` 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 environment. 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 DDI Codebook versions 1.2.2, 2.5 and 2.6, and creates 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.

Native SAS import uses `ReadStat` directly. SAS value-label definitions are read from a companion format catalog. Supply its path via `catalog_file`, or place it alongside the `.sas7bdat` file with the same base name and the extension `.sas7bcat` for automatic discovery. When a catalog does not declare an encoding, it inherits the effective data-file encoding. Use `encoding` and `catalog_encoding` to override their encodings independently; a catalog override does not affect data-file decoding.

Catalog labels for .A through .Z use numeric keys -91 through -116 in the `R labels` attribute. A label for `._` retains an underscore-tagged NA key, distinct from the ordinary NA key for system missing. These keys also apply to dates and timestamps and are preserved in RDS storage.

The native `SAS7BDAT` and `XPT` writers preserve missing values but do not emit value-label definitions or a format catalog. An existing catalog can be reused with `SAS7BDAT` only while its format references and code mappings remain unchanged. A standalone exported data file does not preserve value labels. Creating or modifying SAS format definitions requires a separate catalog or SAS setup commands; see `setupfile()`.

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.

Missing values are harmonized across the entire dataset by default when exporting to Stata or SAS. This automatically builds a dataset-level missing value dictionary when possible. Use `harmonize = FALSE` via the three dots argument to deactivate this behavior. The alias `harmonise = FALSE` is also accepted. If the dataset-level dictionary exceeds 26 codes, conversion falls back to separate dictionaries for each variable. A variable with more than 26 distinct numeric missing codes cannot be represented this way and export fails. SAS's additional `._` tag is preserved as an NA with the name `"_"` in the column's `na_index` attribute, for numeric columns, dates and timestamps. It does not consume a letter-tag dictionary entry; the A-Z tags retain their existing numeric codes from -91 to -116.

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. The native date/time adapters write SPSS dates and timestamps as seconds since 1582-10-14, SAS dates as days since 1960-01-01, and SAS timestamps as seconds since 1960-01-01. Integer-backed R temporal columns use double storage when exporting to SPSS or SAS to avoid overflow. SPSS user-missing codes and ranges remain separate from date/time values; they are retained when importing with `user_na = TRUE` and discarded with `user_na = FALSE`.

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](#), [getCodebook](#), [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)

# Format-specific options are supplied through the three dots. For example,
# import the English labels from a multilingual DDI Codebook using:
```

```

test <- convert("test.xml", language = "en")

# 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	<i>Add/remove/change one or more children or attributes from a DDI Codebook attribute.</i>
--------------	--

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)

makePath(xpath, from, overwrite = TRUE, ...)

moveChild(xpath, from, to, overwrite = TRUE, ...)

replaceChild(xpath, with, 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", or an xpath string pointing to a target element.
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, an xpath to a DDI Codebook element. Indexed segments are supported using square brackets, e.g. <code>codeBook/dataDscr/var[3]</code> . Missing indexes default to the first matching element.
from	A standard element of class "DDI", or an xpath string pointing to a target element.
name	Character, name(s) of specific child element / attribute.
content	Character, the text content of a DDI element.
with	A standard element of class "DDI", or an xpath string pointing to a target 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). XPath resolution accepts indexed segments using `[n]`. When an index is missing, the first matching element is selected.

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

Elements can be repeated. For example, `dataDscr` contains one `var` element per dataset variable. When multiple `var` elements exist, referring only to the name is ambiguous. Use indexed xpaths like `var[3]` to target a specific instance, or use `indexChildren()` to list all positions for iteration.

If more than one children, they should be grouped into a list. Functions `addContent`, `changeContent`, `removeContent`, `addAttributes`, `changeAttributes`, and `removeAttributes` accept either a

standard DDI element or a character xpath. When an xpath is provided, the target element is resolved and replaced in the root element. Child occurrence limits and choices are evaluated in the immediate parent. Required children may be added later while building an element; use `testValid()` to check completeness. Sequence slots are ordered according to the schema, while insertion order within repeated choices is preserved.

Value

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

Author(s)

Adrian Dusa

exportCodebook	<i>Export a DDI Codebook to an XML file.</i>
----------------	--

Description

Create a DDI Codebook version 2.6, XML file structure.

Usage

```
exportCodebook(
  codeBook,
  to = "",
  OS = "",
  indent = 2,
  ...,
  session = NULL,
  revision = NULL
)
```

Arguments

codeBook	A standard element of class "DDI".
to	either a character string naming a file or a connection open for writing ("" indicates output to the console) Home-directory paths beginning with ~ are expanded before writing.
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
session	Optional session returned by <code>metadataSweepCapture()</code> . Its complete variable description replaces <code>dataDscr</code> in <code>codeBook</code> , reusing cached unweighted analysis. Study and document metadata remain supplied by <code>codeBook</code> . Do not also supply data or variables. <code>embed = TRUE</code> includes captured data with stable variable IDs; <code>csv = TRUE</code> requests a companion CSV. Weighted export uses the existing weighted calculation path.

revision Current host revision, required when session is supplied.

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), distribtr, abstract and level (for the otherMat element).

The finished XML is always validated against the bundled, pinned DDI Codebook 2.6 schema and its local dependencies before writing. Invalid or incomplete documents, or unavailable schemas, cause an error. Validation failure leaves the destination and any companion CSV unchanged, and writes nothing to an output connection. No network access is needed for validation. 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. Formatting and target line endings also apply when writing to a connection.

Value

An XML file containing a DDI version 2.6 metadata.

Author(s)

Adrian Dusa

See Also

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

Examples

```
## Not run:
exportCodebook(codeBook, to = "codebook.xml")

# using a namespace
exportCodebook(codeBook, to = "codebook.xml", xmlns = "ddi")

## End(Not run)
```

getCodebook	<i>Extract metadata information</i>
-------------	-------------------------------------

Description

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

Usage

```
getCodebook(from = NULL, encoding = "auto", ignore = NULL, ...)
```

Arguments

from	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 1.2.2, 2.5 or 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.

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 few enough values (ex. number of children)
for which a table of frequencies is possible in addition to frequencies

Apart from utf8, other encodings might be necessary when reading from SPSS or DDI XML files, for instance latin1 or windows-1252, and it also accepts bytes for multi-byte encodings. To use the one specified in the file, set encoding = NULL. The default is encoding = "auto", which tries to detect the encoding automatically.

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)
  )
)

getCodebook(from = 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, lab1, 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	<i>Create a notes element for the dataset.</i>
---------------	--

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.

The complete list of elements for which default values are added is: "IDNo", "titl", "titlStmt", "dis-
trbtr", "distStmt", "holdings", "citation", "abstract", "stdyInfo", "stdyDscr", "prodDate", "software",
"prodStmt", "docDscr" and "otherMat".

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("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

<code>x</code>	A character categorical variable
<code>...</code>	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" 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

Package **DDIwR** uses numeric declared missing codes as its internal R representation. Recoding to "Stata" or "SAS" is therefore best viewed as an export-oriented temporary representation, not as a preferred internal storage strategy for declared vectors in R. SAS's `._` tag is retained separately as `"_"` in `na_index` when it is absent from the recoding dictionary. Columns with missing metadata alone are accepted, even without variable or value labels.

When no dictionary is provided, export-oriented recoding can be performed either by scanning the entire dataset for a harmonized mapping, or by recoding each variable independently. The package now defaults to the faster per-variable strategy in export preparation. Supplying a dictionary keeps the harmonized cross-dataset behavior.

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 `char tonum` 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, 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)
#   )
# )
```

```

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

# attr(xrec, "dictionary")

# Supply a dictionary to harmonize missing meanings across variables
# 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 or regular expression pattern.

Usage

```

searchFor(
  x,
  where = c("everywhere", "title", "description", "attributes", "examples"),
  ...
)

```

Arguments

x	Character, either word(s) or a regular expression.
where	Character, in which section(s) to search for.
...	Other arguments to be passed to the grepl() function.

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 labelled data frame, a DDI codeBook list returned by getCodebook() , or a path to a metadata file or directory 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

A codeBook object is an R list containing DDI metadata, such as variable labels and value labels. It can be created from a labelled data frame with [getCodebook\(\)](#). A data frame can also be passed directly to `setupfile()`. The generated commands have a configuration section where input and output paths must be set before running them in the target software.

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

```
# A small dataset with variable and value labels
survey <- data.frame(
  age = declared(
    c(28, 35, 42),
    label = "Age in years"
  ),
  device = declared(
    c(1, 2, 1),
    label = "Device used",
    labels = c(Desktop = 1, Mobile = 2)
  )
)
```

```

# Use a temporary directory for the files created in this example
temp_dir <- tempdir()
csv_file <- file.path(temp_dir, "survey.csv")
admisc::export(undeclare(survey, drop = TRUE), file = csv_file)

# codeBook is an R object containing DDI metadata extracted from survey.
# It describes the variables and their labels; the observations are in the CSV.
codeBook <- getCodebook(survey)

# Create Stata commands to import the CSV and apply the metadata
# Edit the generated file's configuration paths before running it in Stata.
setupfile(codeBook, file = file.path(temp_dir, "survey.do"), csv = csv_file)

# A labelled data frame can also be supplied directly
setupfile(survey, file = file.path(temp_dir, "survey.sps"))

# The file extension selects the software: .do, .sps, .sas or .R.
# A setup file contains import commands, not the observations themselves.

## Not run:
# Alternatively, read metadata from an existing DDI XML file
setupfile("/path/to/metadata.xml", file = "survey.do",
  csv = "/path/to/survey.csv")

# Batch processing: read metadata and CSV files from matching directories
setupfile("/path/to/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	<i>Validate a DDI element.</i>
-----------	--------------------------------

Description

Checks parent-specific child occurrences, sequences, choices and mandatory attributes throughout a DDI Codebook element.

Usage

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

Arguments

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

Details

Each existing instance is checked separately, including repeated and recursively nested elements. Required children are checked only when their parent exists. Repeated choices may use different alternatives. An incomplete element can be built incrementally with `addChildren()`; `testValid()` also checks the required minimum occurrences and child order.

Validation follows the curated DDI child model. It does not expand imported markup groups or validate all XML datatypes, ID references or mixed text. Use XML Schema validation on exported XML for those additional checks.

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 list of class validation, with character vectors mandatory and optional. The latter contains missing required attributes of optional children; both components report validation problems, not recommendations.

Author(s)

Adrian Dusa

See Also

[makeElement](#)

updateCodebook	<i>Update Codebook.</i>
----------------	-------------------------

Description

Update an XML file containing a DDI Codebook.

Usage

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

Arguments

- | | |
|----------------------|--|
| <code>xmlfile</code> | A path to a DDI Codebook XML document. |
| <code>with</code> | An R object containing a root <code>codeBook</code> element. |
| <code>...</code> | 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. Sections are matched as XML elements, including in compact or prefixed XML, and inserted in schema order. Repeated sections are replaced as a group. The document namespace, root attributes and untouched sections are preserved. Both inputs must have a codeBook root. Invalid root inputs are rejected before the file is written. The finished document must conform to the bundled DDI Codebook 2.6 schema. Validation is mandatory and offline; failure leaves the original file unchanged. Documents using another DDI namespace are rejected rather than silently converted to 2.6.

Author(s)

Adrian Dusa

updateSchema

Update the internal DDI Codebook schema object.

Description

Extract structural facts from a Codebook XSD while preserving the existing R naming conventions and curated descriptive metadata.

Usage

```
updateSchema(xsd = NULL, return = FALSE)
```

Arguments

xsd	Path or URL to codebook.xsd. Defaults to the pinned 2.6 release.
return	Return the proposed list without changing the in-memory cache.

Details

The extractor follows type inheritance, restrictions, local elements and attribute groups. Each element receives a parent-specific contentModel containing sequence/choice particles and numeric min/max occurrence bounds. PHRASE, FORM, imported markup groups and wildcards remain outside the simplified child model. Existing descriptions, examples, recommendations and deprecation flags are preserved; new records use schema documentation.

Inspect `updateSchema(xsd, return = TRUE)` before applying a different schema. Structural facts are refreshed, including removed/prohibited attributes. Legacy element-wide optional/repeatable flags are retained for existing records; validation uses contentModel. No source files are overwritten.

Author(s)

Adrian Dusa

Index

addAttributes (DDI-children), 7
addChildren, 15
addChildren (DDI-children), 7
addContent (DDI-children), 7
anyAttributes (DDI-children), 7
anyChildren (DDI-children), 7

buildDictionary, 2

changeAttributes (DDI-children), 7
changeContent (DDI-children), 7
convert, 3

DDI-children, 7
declared, 6

exportCodebook, 9

getChildren, 15
getChildren (DDI-children), 7
getCodebook, 6, 11
getCodebook(), 19
globalAttributes (showDetails), 21

hasAttributes (DDI-children), 7
hasChildren (DDI-children), 7

indexChildren (DDI-children), 7

makeCategories, 12
makeDataNotes, 13
makeElement, 7, 14, 23
makePath (DDI-children), 7
moveChild (DDI-children), 7

read.csv, 5
read_sav, 4
recodeCharcat, 5, 15
recodeMissings, 16
removeAttributes (DDI-children), 7
removeChildren (DDI-children), 7

removeContent (DDI-children), 7
replaceChild (DDI-children), 7

searchFor, 18
setupfile, 5, 6, 19
showAttributes (showDetails), 21
showDescription (showDetails), 21
showDetails, 15, 21
showExamples (showDetails), 21
showLineages (showDetails), 21
showRelations (showDetails), 21

testValid, 22

updateCodebook, 23
updateSchema, 24

write_dta, 4