# Package: DDIwR (via r-universe)

October 8, 2026

**Title** DDI with R

**Version** 0.22

**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

**Language** en-US

**Encoding** UTF-8

**NeedsCompilation** yes

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

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

**Repository** https://cran.r-universe.dev

**Date/Publication** 2026-10-08 09:09:34 UTC

**RemoteUrl** https://github.com/cran/DDIwR

**RemoteRef** HEAD

**RemoteSha** a15f33bb915a7a16c8da2144c579b6d593633e1e

## Contents

buildDictionary . . . . . 2
convert . . . . . 3
DDI-children . . . . . 6

exportCodebook . . . . . 8
getCodebook . . . . . 10
makeCategories . . . . . 11
makeDataNotes . . . . . 12
makeElement . . . . . 13
metadataSweepCapture . . . . . 14
metadataSweepImport . . . . . 16
recodeCharcat . . . . . 17
recodeMissings . . . . . 18
searchFor . . . . . 20
setupfile . . . . . 21
showDetails . . . . . 23
testValid . . . . . 24
updateCodebook . . . . . 25
updateSchema . . . . . 25

**Index** **27**

---

`buildDictionary` *Build a dictionary of missing-value recodes*

---

## 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,
  chartonum = 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 |
| `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` | 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.

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 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 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 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.

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.

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.

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`, `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` *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)
```

```
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. codeBook/dataDscr/var[3]. 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` *Export a DDI Codebook to an XML file.*

---

## 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) |
| `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 metadataSweepCapture(). Its complete variable description replaces dataDscr in codeBook, reusing cached unweighted analysis. Study and document metadata remain supplied by codeBook. Do not also supply data or variables. embed = TRUE includes captured data with stable variable IDs; csv = TRUE 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), 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/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 *Extract metadata information*

---

### 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, 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.

The complete list of elements for which default values are added is: "IDNo", "titl", "titlStmt", "distrbtr", "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)
```

---

# metadataSweepCapture *Reusable metadata sweep sessions*

---

## Description

Capture metadata once and request ordered subsets for progressive displays, DialogR variable selectors, or DDI Codebook generation.

## Usage

```
metadataSweepCapture(from, revision)

metadataSweepRead(
  session,
  revision,
  columns = NULL,
  fields = NULL,
  xml_options = list(),
  profile = "progressive"
)

metadataSweepInvalidate(session, revision)

metadataSweepClose(session)
```

## Arguments

| | |
|---|---|
| from | A data frame. The host must invalidate the session after edits, including edits made by reference. |
| revision | One non-missing atomic revision token, supplied by the host. Every read must supply the current token; a mismatch makes the session stale. |
| session | An object returned by metadataSweepCapture(). |
| columns | Integer positions, allowing reordering and repetition, or unambiguous variable names. NULL selects all columns; integer() selects none. |
| fields | Requested field groups. NULL uses the selected profile. |
| xml_options | Named arguments passed to the variable XML generator. |
| profile | One of progressive, dialogr, or codebook. |

## Details

Capture copies stored metadata for all columns without cleaning labels or scanning observations. Normalization is cached only for requested columns. The progressive profile returns identity, annotations and category definitions without scanning values. Request classification, weight_signal, summaries or categories to analyse only the selected columns and cache their results. Category definitions (category_metadata) and frequencies (categories) are separate requests. The latter retain DDI's labelled-category semantics. Category definitions include UI rows plus the original typed labels, levels, na_values and na_range, without cleaning or inference.

The dialogr profile returns row records in projection$dialogr, following DialogR's type, measurement, category, display and selector-flag rules. Its value-dependent interpretation is cached per column, separately from DDI statistical analysis. Bare observation copies are shared by both consumers. The codebook profile returns variable XML and export metadata, reusing cached unweighted analysis. Weighted XML retains the existing weighted export path.

Generated IDs remain stable for the life of the session. Numeric positions are required to distinguish duplicate names. Hosts schedule batches and yield to their UI between calls; this API does not start background browser workers. Sessions are process-local and must be recaptured after serialization or edits. Value-dependent requests support logical, integer, double and character columns; unsupported storage (such as list columns) can be read as metadata but raises an error when values are requested. metadataSweepClose() releases caches and is idempotent; a finalizer also releases them. Closing or invalidating a session never changes the input data.

## Value

Capture returns a session. Read returns an ABI-versioned columnar envelope with revision, columns, completed_fields, projection and cache counters. Invalidate and close return invisible NULL.

## See Also

exportCodebook(), collectMetadata()

## Examples

```
session <- metadataSweepCapture(iris, revision = 1L)
metadataSweepRead(session, 1L, columns = 1:2)
metadataSweepRead(session, 1L, columns = "Species", profile = "dialogr")
metadataSweepRead(session, 1L, profile = "codebook")
book <- makeElement("codeBook", children = list(makeElement("stdyDscr", fill = TRUE)))
path <- tempfile(fileext = ".xml")
exportCodebook(book, path, session = session, revision = 1L)
unlink(path)
metadataSweepClose(session)
```

---

# metadataSweepImport *Import data with a prepared metadata sweep session*

---

## Description

Import a dataset and immediately prepare metadata for a declared consumer profile. The returned session remains reusable for every profile; `prepare` controls only which projection is warm when the import returns.

## Usage

```
metadataSweepImport(
  from,
  revision,
  prepare = "progressive",
  columns = NULL,
  fields = NULL,
  xml_options = list(),
  ...
)
```

## Arguments

| | |
|---|---|
| `from` | Path to a dataset supported by `convert()`. |
| `revision` | One non-missing atomic revision token for the imported data. |
| `prepare` | One of `"progressive"`, `"dialogr"`, or `"codebook"`. |
| `columns` | Integer positions or unambiguous variable names to prepare. `NULL` prepares every column. Progressive consumers should provide their initial visible range. |
| `fields` | Optional field groups passed to `metadataSweepRead()`. |
| `xml_options` | Named arguments passed to the variable XML generator. |
| `...` | Import arguments passed to `convert()`. The import destination is always R; `to` must not be supplied. |

## Value

An object with `data`, `session`, and the prepared `projection`. Close `session` with `metadataSweepClose()` when it is no longer needed.

## Examples

```
## Not run:
imported <- metadataSweepImport(
    "survey.sav",
    revision = 1L,
    prepare = "progressive",
    columns = 1:20
)
metadataSweepClose(imported$session)

## End(Not run)
```

---

# 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"` 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.

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 `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, 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 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

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 *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

---

## 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*)*, 6
addChildren, *14*
addChildren *(*DDI-children*)*, 6
addContent *(*DDI-children*)*, 6
anyAttributes *(*DDI-children*)*, 6
anyChildren *(*DDI-children*)*, 6

buildDictionary, 2

changeAttributes *(*DDI-children*)*, 6
changeContent *(*DDI-children*)*, 6
collectMetadata(), *15*
convert, 3
convert(), *16*

DDI-children, 6
declared, *5*

exportCodebook, 8
exportCodebook(), *15*

getChildren, *14*
getChildren *(*DDI-children*)*, 6
getCodebook, *5*, 10
globalAttributes *(*showDetails*)*, 23

hasAttributes *(*DDI-children*)*, 6
hasChildren *(*DDI-children*)*, 6

indexChildren *(*DDI-children*)*, 6

makeCategories, 11
makeDataNotes, 12
makeElement, *6*, 13, *25*
makePath *(*DDI-children*)*, 6
metadataSweepCapture, 14
metadataSweepCapture(), *9*
metadataSweepClose
*(*metadataSweepCapture*)*, 14
metadataSweepClose(), *17*
metadataSweepImport, 16
metadataSweepInvalidate
*(*metadataSweepCapture*)*, 14
metadataSweepRead
*(*metadataSweepCapture*)*, 14
metadataSweepRead(), *16*
moveChild *(*DDI-children*)*, 6

read.csv, *4*
read_sas, *5*
read_sav, *4*
recodeCharcat, *4*, 17
recodeMissings, 18
removeAttributes *(*DDI-children*)*, 6
removeChildren *(*DDI-children*)*, 6
removeContent *(*DDI-children*)*, 6
replaceChild *(*DDI-children*)*, 6

searchFor, 20
setupfile, *5*, 21
showAttributes *(*showDetails*)*, 23
showDescription *(*showDetails*)*, 23
showDetails, *14*, 23
showExamples *(*showDetails*)*, 23
showLineages *(*showDetails*)*, 23
showRelations *(*showDetails*)*, 23

testValid, 24

updateCodebook, 25
updateSchema, 25

write_dta, *4*