Package ‘xmlconvert’

October 14, 2022

Type Package

Title Comfortably Converting XML Documents to Dataframes and Vice Versa

Description Converts XML documents to R dataframes and dataframes to XML documents. A wide variety of options allows for different XML formats and flexible control of the conversion process. Results can be exported to CSV and Excel, if desired. Also converts XML data to R lists.

Version 0.1.2

Maintainer Joachim Zuckarelli <joachim@zuckarelli.de>

Depends R (>= 3.5.0)

License GPL-3

Imports xml2, stringr, readr, tibble, httr, utils, lubridate

Suggests xlsx

Repository CRAN

BugReports https://github.com/jsugarelli/xmlconvert/issues

URL https://github.com/jsugarelli/xmlconvert/

Encoding UTF-8

ByteCompile true

RoxygenNote 7.1.1

NeedsCompilation no

Author Joachim Zuckarelli [aut, cre] (<https://orcid.org/0000-0002-9280-3016>)

Date/Publication 2021-03-27 08:10:02 UTC

R topics documented:

df_to_xml *Converting XML to data frames and vice versa*

Description

Converts dataframes to XML documents.

Usage

```
df_to_xml(
  df,
  fields = "tags",
  record.tag = "record",
  field.names = NULL,
  only.fields = NULL,
  exclude.fields = NULL,
  root.node = "root",
  xml.file = NULL,
  non.exist = NULL,
  encoding = "UTF-8",
  no.return = FALSE
)
```

Arguments

`df`	Dataframe to be converted to XML
`fields`	A character value, either `"tags"` or `"attributes"`. Specifies whether the fields of each data record will be represented in the resulting XML document by XML tags or by attributes. See the *Details* section of `df_to_xml()` for more on this topic. Default is `"tags"`
`record.tag`	Name of the tags that will represent the data records in the resulting XML (i.e. each record has one element with this tag name).
`field.names`	Names of the fields in the XML file. If `NULL` then the variable names from the dataframe are used. Field names are corrected automatically to comply with XML naming requirements.
`only.fields`	Optional vector variable names from the dataframe that will be included in the XML document. If `NULL` then all fields from the dataframe are included in the XML document.
`exclude.fields`	Optional vector of variable names from the dataframe that will not be included in the XML document.
`root.node`	A character value with the desired name of the root element of the XML document; `"root"` by default.
`xml.file`	Name of a file the XML document it written to. If `NULL` no file will be created.

`non.exist`	Value that will be written into the XML document as field value in case the respective element in the dataframe is `NA`. If `NULL`, which is the default, no XML tag/attribute (depennding on the `fields` argument) will be created for this dataframe element. Using a non-`NULL` value for `non-exist` allows to make sure that each data record tag in the XML document has exactly the same structure even if some values may be empty (because they are `NA` in the original data)
`encoding`	Encoding of the XML document; default is `"UTF-8"`
`no.return`	Logical option to prevent `df_to_xml()` from returning the XML document it creates; use this if you are only interested in saving the XML document to a file using the `xml.file` argument.

Value

The resulting XML document that be edited with the functions from the **xml2** package. There is no return value if the `no.return` argument is set to `TRUE`.

See Also

Other xmlconvert: `xml_to_df()`

Examples

```
# Create a dataframe
soccer.worldcups <- data.frame(list(year=c(2014, 2010, 2006),
    location=c("Brazil", "South Africa", "Germany"),
    goals_scored=c(171, 145, 147),
    average_goals=c(2.7, 2.3, 2.4),
    average_attendance=c(57918, 49669,52491)),
    stringsAsFactors = FALSE)

# Convert to XML with the fields (i.e. dataframe variables/columns) stored in XML tags
xml <- df_to_xml(soccer.worldcups, fields="tags", record.tag = "worldcup")

# Convert to XML with the fields (i.e. dataframe variables/columns) stored in attributes
xml <- df_to_xml(soccer.worldcups, fields="tags", record.tag = "worldcup")
```

`xmlconvert` *Package 'xmlconvert'*

Description

Tools for converting XML documents to dataframes and vice versa

Functions available:

- `xml_to_df()`: Converts an XML document to an R dataframe
- `df_to_xml()`: Converts a dataframe to an XML document

xml_to_df *Converting XML to data frames and vice versa*

Description

`xml_to_df()` converts XML data to a dataframe. It provides a lot of flexibility with its arguments but can usually be used with just a couple of them to achieve the desired results. See the examples below for simple applications.

Usage

```
xml_to_df(
  file = NULL,
  text = NULL,
  first.records = NULL,
  xml.encoding = "",
  records.tags = NULL,
  records.xpath = NULL,
  fields = "tags",
  field.names = NULL,
  only.fields = NULL,
  exclude.fields = NULL,
  check.datatypes = TRUE,
  data.dec = ".",
  data.thds = ",",
  stringsAsFactors = FALSE,
  na = NA,
  non.exist = na,
  no.hierarchy = FALSE,
  hierarchy.field.delim = "|",
  hierarchy.value.sep = "~",
  no.return = FALSE,
  excel.filename = NULL,
  excel.sheetname = NULL,
  excel.pw = NULL,
  csv.filename = NULL,
  csv.sep = ",",
  csv.dec = ".",
  csv.encoding = "",
  strip.ns = FALSE,
  ...
)
```

Arguments

`file`	XML file to be converted. Instead of specifying a file, the XML code can put in directly via the `text` argument,

`text`	XML code to be converted. Instead of providing the XML code, an XML file can be specified with the `file` argument.
`first.records`	Number of records to be converted. If `NULL` (default) all records will be converted.
`xml.encoding`	Encoding of the XML file (optional), e.g. (`"UTF-8"`)
`records.tags`	Name (or vector of names) of the tags that represent the data records in the XML (i.e. each record has one element with this tag name). All elements with this tag name will be considered data records. Instead of specifying the tag name, an XPatch expression can be used to identify the data records (see `records.xpath`)
`records.xpath`	XPath expression that specifies the XML element to be selected as data records; can be used instead of specifying the data record XML tags directly with the `data.records` argument. If both, `records.tags` and `records.path` are provided, only the XPath expressions determines the tags to be selected.
`fields`	A character value, either `"tags"` or `"attributes"`. Specifies whether the fields of each data record are represented as XML tags or as attributes. See *Details* below for more on this topic. Default is `"tags"`
`field.names`	If the data fields are represented by XML elements/tags (i.e. `fields = "tags"`) and it is not the tag name that identifies the name of the data field but an attribute of that field tag then the name of the attribute that provides the field name can be specified here. If `NULL`, the tag names will be used as field names. See *Details* for more information.
`only.fields`	Optional vector of tag or attribute names (depending on the `fields` argument) of an XML record that will be included in the resulting dataframe. `NULL` means all fields found in the data will end up as columns in the dataframe.
`exclude.fields`	Optional vector of fields that will be excluded from the conversion; fields specified here will not end up as columns in the resulting dataframe
`check.datatypes`	Logical option that specifies if `xml_to_df()` tries to identify the data types of the fields in the XML data. If `TRUE` (default), `xml_to_df()` tries to identify numeric fields and changes the class of the respective columns in the resulting dataframe accordingly. Use the `data.dec` and `data.thds` arguments to specify a number format different from the standard US/EN scheme. At this point, there is no data type identification for logical and time/date values available. If `check.datatypes` equals `FALSE` then all variables in the resulting dataframe will be of class `character`
`data.dec`	A decimal separator used in the identification of numeric data when `check.datatypes = TRUE`. Default is dot (`.`)
`data.thds`	A thousands separator used in the identification of numeric data when `check.datatypes = TRUE`. Default is comma (`,`)
`stringsAsFactors`	Logical option specifying if character values will be converted to factors in the resulting data frame. Is only applied if `check.datatypes = TRUE` which is the default
`na`	Value that will be put into the resulting dataframe if the XML data field is *empty*. Default is `NA`. If a data record in the XML does not have a specific field at all it is filled with the value provided via the `non.exist` argument

`non.exist`	Value that will be put into the resulting dataframe if a data record in the XML data does not have a specific field at all. Default is the value of the `na` (the default of which is `NA`). If instead a field is present in the XML data but empty, then it will have the value of the `na` argument in the resulting data frame
`no.hierarchy`	If the fields in the XML data are represented by XML elements/tags (i.e. argument `fields = "tags"`) and there is a hierarchical structure below a data field element then this hierarchical structure gets 'flattened', i.e. it will be represented by a single character value. See *Details* for an example
`hierarchy.field.delim`	One or two-element character vector specifying the tag delimiters used when 'flattening' a hierarchy in the XML data. If only one delimiter is specified then this delimiter will be used for both, the beginning of the tag and the end of the tag. See *Details* for an example
`hierarchy.value.sep`	Character value that is used as the separator between the tag name and the value of the tag when 'flattening' a hierarchy in the XML data. See *Details* for an example
`no.return`	Logical option to prevent `xml_to_df()` from returning the dataframe it creates; use this if you are only interested in saving the dataframe as Excel or CSV.
`excel.filename`	Name of an Excel file the resulting dataframe will be exported to. If `NULL` (default) there will be no Excel export.
`excel.sheetname`	Name of the worksheet the resulting dataframe will be exported to when using the Excel export via the `excel.filename` argument. If `NULL`, `xml_to_df()` will figure out a name,
`excel.pw`	Password that is applied to the Excel workbook when the resulting data.frame is exported via the `excel.filename` argument. Default NULL means the workbook will not be protected with a password
`csv.filename`	Name of a CSV file the resulting dataframe will be exported to. If `NULL` there will be no CSV export.
`csv.sep`	Separator used to separate fields in the CSV file when exporting the resulting dataframe via the `csv.filename` argument. Default is comma (`","`)
`csv.dec`	Decimal separator used when exporting the resulting dataframe via the `csv.filename` argument, Default is dot (`.`)
`csv.encoding`	Text encoding used when exporting the resulting dataframe via the `csv.filename` argument
`strip.ns`	Logical option that can be used to strip the namespaces from the XML data. Default is `FALSE`. Try this if parsing of your XML data fails and namespaces are present in your data. Please note: Removing the namespaces from the XML data may increase the time needed for parsing.
`...`	Additional arguments passed on the `write.table()` when exporting the resulting dataframe via the `csv.filename` argument, Default is dot (`.`)

Details

This section provides some more details on how `xml_to_df()` works with different ways of representing data fields in the XML (tags versus attributes) and on working with nested XML field structures.

Two ways of representing the data: Tags and attributes: For `xml_to_df()` records are always represented by tags (i.e XML elements). Data fields within a record, however, can be represented by either tags or attributes.

In the former case the XML would like like this:
```
<xml>
....<record>
........<field1>Value 1-1</field1>
........<field2>Value 1-2</field2>
....</record>
....<record>
........<field1>Value 2-1</field1>
........<field2>Value 2-2</field2>
....</record>
....</xml>
```

Here, each data field is represented by its own tag (e.g. `field1`). In this case the `records.tag` argument would need to be `"record"` (or we would specify an XPath expression with `records.xpath` that selects these elements) as this is the name of the tags that carry the data records; the `fields` argument would need to be `"tags"` because the actual data fields are represented by tags nested into the record elements.
A variant of this case would be if the fields are represented by tags but the field names are not the tag names but are provided by some attribute of these tags. This is the case in the following example:
```
<xml>
....<record>
........<data name="field1">Value 1-1</data>
........<data name="field2">Value 1-2</data>
....</record>
....<record>
........<data name="field1">Value 2-1</data>
........<data name"field2">Value 2-2</data>
....</record>
....</xml>
```

Here, we would use the optional `field.names` argument to tell `xml_to_df()` with `field.names = "name"` that each data tag has an attribute called `"name"` that specifies the field name for this data field tag.

In contrast to these cases, data fields can also be represented by attributes of the record:
```
<xml>
```

```
....<record field1="Value 1-1" field2="Value 1-2" />
....<record field1="Value 2-1" field2="Value 2-2" />
</xml>
```

Here would need to change the `fields` argument to `"attributes"`.

Working with the nested field values: When data fields are represented by XML elements / tag then there may nested structures beneath them. If the `no.hierarchy` argument is `FALSE` (default) this nested structure within a field is recursively analyzed and represented in a single character value for this field. Each nested element is enclosed in the delimiters provided with the `hierarchy.field.delim` argument. By default, there is only one such delimiter (and that is `"|"`) which is used mark both, the start and the end of an element in the resulting value. However, it is possible to specify to different symbols in order to distinguish start and end of an element. The `hierarchy.value.sep` argument is the symbol that separates the name of the argument from its value. Consider the following example:

```
<xml>
....<ship>
........<name>Excelsior<name>
........<lastcaptain>Hikaru Sulu</lastcaptain>
....</ship>
....<ship>
........One proud ship name, many captains
........<name>Enterprise<name>
........<lastcaptain>
............<NCC-1701-A>James Tiberius Kirk</NCC-1701-A>
............<NCC-1701-B>John Harriman</NCC-1701-B>
............<NCC-1701-C>Rachel Garrett</NCC-1701-C>
............<NCC-1701-D>Jean-Luc Picard</NCC-1701-D>
........</lastcaptain>
....</ship>
</xml>
```

Calling `xml_to_df()` with the default values for `hierarchy.field.delim` and `hierarchy.value.sep` would result in the following value of the `lastcapatin` field for the `Enterprise` record:

```
One proud name, many captains|NCC-1701-A~James Tiberius Kirk||NCC-1701-B~John Harriman||NCC-1701-C~Ra
Garrett||NCC-1701-D~Jean-Luc Picard|
```

If we would use `hierarchy.field.delim = c("[", "]")` then we would better see the start and of end of each element:

```
One proud name, many captains[NCC-1701-A~James Tiberius Kirk][NCC-1701-B~John Harriman][NCC-1701-C~Ra
Garrett][NCC-1701-D~Jean-Luc Picard]
```

Value

The resulting dataframe. There is no return value if the `no.return` argument is set to `TRUE`.

See Also

Other xmlconvert: `df_to_xml()`

Examples

```
# Data used: World population figures from the United Nations Statistics Division

# Read in the raw XML; two versions: one with data fields as XML
# elements/tags, one with data fields as attributes
example.tags <- system.file("worldpopulation_tags.xml", package="xmlconvert")
example.attr <- system.file("worldpopulation_attr.xml", package="xmlconvert")

# Convert XML data to dataframe
worldpop.tags <- xml_to_df(text = example.tags, records.tags = c("record"),
    fields = "tags", field.names = "name")
worldpop.attr <- xml_to_df(text = example.attr, records.tags = c("record"),
    fields = "attributes")
```

`xml_to_list` *Converting XML documents to lists*

Description

Converts XML documents to lists. Uses the `as_list()` function from the `xml2` package but improves its output. As an effect, numbers, dates and times are converted correctly, unnecessary nested sub-lists with only element are avoided, and empty XML nodes can be removed altogether. This makes the resulting list look cleaner and better structured.

Usage

```
xml_to_list(
  xml,
  cleanup = TRUE,
  convert.types = TRUE,
  dec = ".",
  thsd = ",",
  num.replace = "",
  datetime.formats = NULL
)
```

Arguments

`xml`	XML document to be converted. Can be read in from a file using `xml2`'s `read_xml()` function.
`cleanup`	If `TRUE` (default) empty XML nodes (with no sub-nodes or values) will not appear in the list.
`convert.types`	If `TRUE` (default) `xml_to_list()` will try to infer the data type of value elements in the XML. If `FALSE`, all value elements in the resulting list will be of type `character`.
`dec`	Decimal separator used in numbers.

`thsd`	Thousands separator used in numbers.
`num.replace`	An optional string that will be removed before `xml_to_list()` tries to convert values to numbers. Can be used, for example, to remove currency symbols or other measurement units from values that are actually numerical.
`datetime.formats`	A vector of date and/or time formats that will be used to recognize the respective datatypes. Formats will need to be written in the general notation used by `strftime()` and other standard R functions.

Value

A `list` object representing the XML document.

Examples

```
xml <- xml2::read_xml(system.file("customers.xml", package="xmlconvert"))
xml.list <- xml_to_list(xml, num.replace="USD", datetime.formats = "%Y-%m-%d")#'
```

Index