Package 'marcxmlr'

September 28, 2026

Title Faithful and Scalable MARCXML Parsing

Version 0.3.1

Description Parses Machine-Readable Cataloging ('MARC 21') XML <https://www.loc.gov/standards/marcxml/> into a canonical tidy long representation while preserving leaders, control fields, data fields, indicators, repeated fields, repeated subfields, and source order. Provides an in-memory reader for manageable catalogues and a bounded-memory converter that writes larger collections as 'Parquet' datasets, with optional local parallel processing.

License MIT + file LICENSE

URL https://github.com/larry77/marcxmlr

BugReports https://github.com/larry77/marcxmlr/issues

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports future (>= 1.69.0), futurize, purrr (>= 1.0.0), rlang, stats, tibble (>= 3.0.0), xml2 (>= 1.3.0)

Suggests arrow, dplyr, future.mirai, furrr, mori, testthat (>= 3.0.0), XML

Config/testthat/edition 3

SystemRequirements libxml2 (>= 2.9.0)

NeedsCompilation yes

Author Lorenzo Isella [aut, cre]

Maintainer Lorenzo Isella <lorenzo.isella@gmail.com>

Repository CRAN

Date/Publication 2026-09-28 07:40:02 UTC

Contents

`marcxmlr-package` *marcxmlr: Faithful and Scalable MARCXML Parsing*

Description

Parse MARC21 XML into a canonical tidy long representation while preserving repeated structures and source order. Use `read_marcxml()` for in-memory work and `marcxml_to_parquet()` for bounded-memory conversion to a disk-backed Parquet dataset.

See Also

Useful links:

- https://www.loc.gov/standards/marcxml/
- https://www.loc.gov/marc/marcdocz.html

`diagnose_canonical` *Diagnose a canonical MARC representation*

Description

Check whether an in-memory canonical `marcxmlr` representation contains enough unambiguous structure to be serialized as MARCXML. Structural problems are reported as errors. Stale or gapped analytical coordinates that can be regenerated after writing and rereading are reported as warnings.

Usage

```
diagnose_canonical(x)
```

Arguments

`x`	A data frame or tibble containing the canonical 11-column `marcxmlr` representation. Additional columns are ignored.

Details

This function diagnoses the canonical representation used by `marcxmlr`. It is not a complete MARC cataloguing validator and does not repair `x`.

Value

A tibble with columns `severity`, `code`, `message`, `record_id`, `field_order`, and `subfield_order`. A clean canonical representation returns a zero-row tibble.

Examples

```
example_file <- system.file(
  "extdata", "example-marcxml.xml", package = "marcxmlr"
)
x <- read_marcxml(example_file)
diagnose_canonical(x)
```

`marcxml_to_parquet` *Convert a MARCXML collection to a Parquet dataset*

Description

`marcxml_to_parquet()` streams complete MARCXML records from a collection, parses them in bounded batches, and writes the canonical long representation as a directory of Parquet files. It does not construct a DOM for the complete XML document and does not materialize the complete parsed result in R.

Usage

```
marcxml_to_parquet(
  file,
  output_dir,
  batch_records = 5000L,
  workers = 1L,
  chunk_records = NULL,
  compression = "snappy",
  verbose = TRUE
)
```

Arguments

`file`	One or more MARCXML collection paths, or glob patterns such as `"catalogue/*.xml"`. Glob matches are processed in sorted order.
`output_dir`	Path for the new Parquet dataset directory. It must not already exist. The directory is published only after successful conversion.
`batch_records`	Maximum number of records converted into one bounded canonical batch before writing. This bounds normal working memory, though an unusually large individual record can itself require substantial memory.
`workers`	Number of local worker processes. The default, `1`, is sequential. Values greater than one require the optional parallel packages listed in `Suggests` and, with current dependency versions, R 4.3 or later.

chunk_records	Number of records assigned to each parsing and writing task. NULL targets approximately two tasks per worker in each batch.
compression	Parquet compression codec passed to arrow::write_parquet().
verbose	Whether to report cumulative records and files after each completed batch.

Details

The input must have a collection root in the official MARCXML namespace (http://www.loc.gov/MARC21/slim) or no namespace. A standalone record can be read with read_marcxml() but is not accepted by this collection converter.

With workers = 1 and default chunk_records = NULL, supported ordinary collections use a two-pass native libxml2 engine. The first pass validates and counts the collection; the second fills bounded canonical batches directly from xmlTextReaderExpand() nodes and writes them with arrow. No record XML is serialized or reparsed on this path.

For one input file, unsupported input, explicit chunk_records, and parallel calls retain the established serialized-record/native or XML event-stream implementations.

When file resolves to multiple files, complete files are the unit of parallel work. Each file is parsed by the existing sequential engine and writes independent Parquet fragments. Global record_id values are assigned deterministically in resolved file order, regardless of worker completion order. XML/libxml2 external pointers are never sent to workers. Explicit chunk_records is not supported for multi-file input.

Parallel work is dispatched through a temporary future.mirai plan using futurize; the caller's previous future plan is restored on exit.

Each task writes a uniquely named temporary file and renames it only after a successful Parquet write. All files are first written under a staging directory beside output_dir; the completed directory is renamed into place only after the XML input has been fully processed. Existing output is never overwritten.

Open the result with arrow::open_dataset(output_dir). Opening a dataset is lazy; calling collect() on the entire dataset will nevertheless materialize every row in R memory.

Value

Invisibly, a tibble with one row per resolved input file containing the normalized input and output paths, record and row counts, number of batches, and number of Parquet files. Single-file input therefore retains the existing one-row return value. Parsed rows remain in the dataset directory.

Examples

```
if (requireNamespace("XML", quietly = TRUE) &&
    requireNamespace("arrow", quietly = TRUE)) {
  example_file <- system.file(
    "extdata", "example-marcxml.xml", package = "marcxmlr"
  )
  output <- tempfile("marcxml-parquet-")

  conversion <- marcxml_to_parquet(
    example_file,
```

```
    output_dir = output,
    workers = 1L,
    verbose = FALSE
  )

  dataset <- arrow::open_dataset(output)
  conversion
  dataset

  unlink(output, recursive = TRUE)
}
```

`read_marcxml` *Read MARCXML into a canonical long tibble*

Description

`read_marcxml()` reads a MARC21 XML collection or a standalone record and returns one row for each leader, control field, or data-field subfield. It preserves repeated fields, repeated subfields, indicators, and source order.

Usage

```
read_marcxml(file, n_max = Inf, workers = 1L, chunk_records = NULL)
```

Arguments

`file`	Path to a MARCXML file.
`n_max`	Maximum number of records to parse. Use `Inf` for every record or `0` to return an empty result with the canonical schema.
`workers`	Number of local worker processes. The default, `1`, is sequential. Values greater than one require the optional parallel packages listed in `Suggests` and, with current dependency versions, R 4.3 or later.
`chunk_records`	Number of records assigned to each parsing task. `NULL` creates one task in sequential mode and approximately two tasks per worker in parallel mode.

Details

This function materializes the parsed result in memory. On the supported sequential native path it does not build a DOM for the complete XML input; compatibility fallbacks may do so. Use `marcxml_to_parquet()` for catalogues whose canonical result may not fit in memory.

`record_id` is the record's positional identity in the input selected for parsing; it is not derived from control field `001`. `field_order` is zero for the leader and then counts variable fields from one. `field_occurrence` counts occurrences of a field type and tag within a record.

Data-field rows carry `subfield_order`, the position of the subfield within its containing field, and `subfield_occurrence`, the occurrence of that code within the same field. Structural columns that do not apply to leaders or control fields are `NA`.

With `workers = 1` and default `chunk_records = NULL`, supported ordinary input uses a two-pass native libxml2 engine: the first pass validates and counts selected records and the second fills the canonical columns directly from expanded record nodes. No record XML is serialized or reparsed on this path. Unsupported input falls back to the reference `xml2` implementation.

Parallel parsing retains the established serialized-record implementation. `xml2`/libxml2 external pointers are never sent to worker processes. The caller's previous future plan is restored when parsing finishes or fails.

Value

A tibble with columns `record_id`, `field_type`, `tag`, `subfield_code`, `value`, `field_order`, `field_occurrence`, `ind1`, `ind2`, `subfield_order`, and `subfield_occurrence`, in that order.

Examples

```
example_file <- system.file(
  "extdata", "example-marcxml.xml", package = "marcxmlr"
)

records <- read_marcxml(example_file)
records

# Inspect repeated subfields without collapsing them.
records[
  records$record_id == 1L & records$tag == "856",
  c("subfield_code", "value", "subfield_order", "subfield_occurrence")
]
```

`write_marcxml` *Write a canonical MARC representation as MARCXML*

Description

Serialize a canonical `marcxmlr` representation to MARCXML. In-memory data frames use the direct in-memory path; Arrow Datasets and lazy Arrow queries are consumed incrementally in bounded batches. MARC record structure and values are preserved, but incidental XML serialization details such as indentation, namespace-prefix spelling, comments, entity spelling, and the original XML declaration are not.

Usage

```
write_marcxml(
  x,
  file,
```

```
  check = TRUE,
  pretty = TRUE,
  records_per_file = Inf,
  compression_level = 6L
)
```

Arguments

x	A data frame or tibble containing the canonical 11-column `marcxmlr` representation, or an Arrow `Dataset` / lazy `arrow_dplyr_query` with the same columns. Arrow inputs require the optional `arrow` package and must present complete record groups in non-decreasing `record_id` order.
file	Output XML path. When `records_per_file` is finite, this path is used as the stem for deterministic shard names such as `catalogue-00001.xml`. Existing target files are not overwritten. Output files are staged before being committed so a serialization failure does not leave a partial shard family.
check	Whether to emit warning-level canonical diagnostics before writing. Structurally ambiguous input is always rejected, including when `check = FALSE`.
pretty	Whether to format the output XML with indentation.
records_per_file	Maximum number of complete MARC records per output file. The default, `Inf`, writes one collection. A finite value writes numbered collection shards and never splits a record.
compression_level	Gzip compression level from 1 to 9. The default is `6`. It is used only when `file` ends in `.gz`; other filenames are written as uncompressed XML.

Details

Records are written in ascending `record_id` order. Within each record, `field_order` and `subfield_order` determine field and subfield ordering. The occurrence columns are diagnostics only and are not used to construct the XML.

Use `diagnose_canonical()` to inspect structural errors and warning-level coordinate issues without writing an XML file.

Value

Invisibly, a character vector containing the output path or paths.

See Also

`diagnose_canonical()`

Examples

```
example_file <- system.file(
  "extdata", "example-marcxml.xml", package = "marcxmlr"
)
```

```
x <- read_marcxml(example_file)
out <- tempfile(fileext = ".xml")
write_marcxml(x, out)

shard_dir <- tempfile("marcxmlr-shards-")
dir.create(shard_dir)
shard_stem <- file.path(shard_dir, "catalogue.xml")
write_marcxml(x, shard_stem, records_per_file = 1)
unlink(shard_dir, recursive = TRUE)
```

Index