Package: marcxmlr (via r-universe)

October 20, 2026

Title Efficient MARCXML Processing and Record Querying

Version 0.5.0

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 in-memory reading, batched conversion to 'Parquet', and reconstruction of MARCXML from canonical data. Supports MARC-aware selection of complete records before materialisation using native XML prefiltering, and after materialisation on canonical tables and 'Arrow' datasets. Includes structural diagnostics for in-memory and 'Arrow'-backed representations, optional recovery from malformed MARC records, and 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

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, knitr, mori, rmarkdown, testthat (>= 3.0.0), XML

VignetteBuilder knitr

Config/testthat/edition 3

SystemRequirements libxml2 (>= 2.9.0), Expat (optional, for record filtering)

NeedsCompilation yes

Config/roxygen2/version 8.1.0

Author Lorenzo Isella [aut, cre]

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

Config/pak/sysreqs libxml2-dev

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

Date/Publication 2026-10-19 09:00:08 UTC

RemoteUrl https://github.com/cran/marcxmlr

RemoteRef HEAD

RemoteSha 43000f6d12a1c14655d9cb0d49b6387b99ebfd47

Contents

marcxmlr-package 2
diagnose_canonical 3
marc_filter_records 4
marc_has 4
marcxml_to_parquet 5
read_marcxml 8
write_marcxml 10

Index **13**

`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 a 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, or an Arrow `Dataset` / lazy `arrow_dplyr_query` with the same columns. Additional columns are ignored. Arrow inputs require the optional `arrow` package and must be grouped in non-decreasing `record_id` order.

Details

In-memory data frames are checked directly. Arrow `Dataset` and lazy Arrow query inputs are scanned incrementally in bounded batches without collecting the complete canonical table. A record that crosses a scanner batch boundary is retained until the complete record is available for diagnosis. Arrow inputs must therefore present complete record groups in non-decreasing `record_id` order, matching the bounded-input contract of `write_marcxml()`.

The 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)
```

marc_filter_records *Filter complete canonical MARC records*

Description

Apply a `marc_has()` query to an already materialized canonical tibble or to an Arrow Dataset. Selection is by `record_id`, and **all rows** belonging to selected records are returned. Arrow operations remain lazy until `dplyr::collect()` is called. `record_id` must uniquely identify records across the entire input Dataset (including all Parquet fragments).

Usage

```
marc_filter_records(.data, query)
```

Arguments

`.data`	A canonical MARC tibble or Arrow Dataset.
`query`	A `marc_has()` query, possibly combined using `&`, `|`, `!`.

Value

All canonical rows of records matching `query`, as a tibble or lazy Arrow query depending on `.data`.

Examples

```
x <- tibble::tibble(record_id = c(1L, 2L), tag = c("650", "856"),
  subfield_code = c("a", "u"), value = c("Economics", "example.org"),
  ind1 = c(" ", "4"), ind2 = c("0", "0"))
marc_filter_records(x, marc_has("650"))
```

marc_has *Test whether a MARC record contains a matching field or subfield*

Description

`marc_has()` constructs a query specification, not a row-wise predicate. Conditions within one call apply to the same canonical row; `&`, `|`, and `!` combine conditions at the **record** level. Use `marc_filter_records()` on a canonical tibble or an Arrow Dataset, or pass the query as `filter` to `read_marcxml()` or `marcxml_to_parquet()`.

Usage

```
marc_has(
  tag,
  subfield = NULL,
  equals = NULL,
  contains = NULL,
  ind1 = NULL,
  ind2 = NULL
)

## S3 method for class 'marc_query'
Ops(e1, e2)
```

Arguments

tag	MARC field tag (or "LDR" for the leader).
subfield	Optional subfield code.
equals	Optional case-sensitive exact value.
contains	Optional case-sensitive literal substring; cannot be used together with equals.
ind1, ind2	Optional indicators (a single blank is allowed).
e1, e2	MARC query objects; e2 is unused for unary !.

Value

An object of class marc_query that supports &, |, and !.

Examples

```
q <- marc_has("650", subfield = "a", contains = "Economics") &
  !marc_has("999")
```

---

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,
  on_marc_error = c("stop", "skip"),
  error_report = NULL,
  filter_tag = NULL,
  filter_subfield = NULL,
  filter_equals = NULL,
  filter = NULL
)
```

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.
`on_marc_error`	Action when a well-formed XML record is structurally invalid MARCXML. The default, `"stop"`, preserves strict behaviour. `"skip"` omits the complete malformed record, writes it to `error_report` with the failure reason, and continues. XML syntax errors always remain fatal. Recovery mode currently supports one input file with `workers = 1`.
`error_report`	Path for the CSV diagnostic report used when `on_marc_error = "skip"`. The file must not already exist.
`filter_tag`	Optional three-digit MARC field tag. Filters records before native materialization using byte offsets. Requires uncompressed UTF-8 XML without DTD.
`filter_subfield`	Optional one-character subfield code within the tag.
`filter_equals`	Optional exact value of the specified subfield.

`filter` Optional record-level query constructed with `marc_has()` and combined with `&`, `|`, or `!`. Cannot be combined with the legacy `filter_*` arguments. Each file is screened in one Expat pass; selected records are written to Parquet in bounded native batches. Multi-file parallel processing is supported, but single-file parallel filtering is prohibited.

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.

With `on_marc_error = "skip"`, supported sequential input uses the native recovery planner and bounded native batch reader. The complete XML document is validated during the first libxml2 pass; structurally invalid MARC records are diagnosed and omitted as complete records, while valid records are written to Parquet directly from native expanded nodes in the second pass. The package never repairs a rejected record. The diagnostic CSV is staged and published only after successful conversion. Skipped records retain their source positions, so `record_id` may contain gaps. Recovery mode adds `source_records` and `skipped_records` to the returned one-row summary.

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,
  on_marc_error = c("stop", "skip"),
  error_report = NULL,
  filter_tag = NULL,
  filter_subfield = NULL,
  filter_equals = NULL,
  filter = 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.
on_marc_error	Action when a well-formed XML record is structurally invalid MARCXML. The default, "stop", preserves strict behaviour. "skip" omits the complete malformed record, writes it to error_report with the failure reason, and continues. XML syntax errors always remain fatal. Recovery mode is currently sequential.
error_report	Path for the CSV diagnostic report used when on_marc_error = "skip". The file must not already exist.
filter_tag	Optional three-digit MARC field tag. When supplied, select records containing the tag using a native Expat offset index. Filtered records receive consecutive record_id values. Requires uncompressed UTF-8 XML without a DTD.
filter_subfield	Optional one-character subfield code to require within filter_tag.
filter_equals	Optional exact text value for filter_subfield.
filter	Optional record-level query constructed with marc_has() and combined with &, |, or !. Cannot be combined with the legacy filter_* arguments. Uses one native Expat scan per XML file and returns complete selected records. The same limitations as native C4 apply.

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.

With on_marc_error = "skip", supported sequential input uses a separate two-pass native recovery engine. The first libxml2 pass validates the complete XML document, counts valid records,

and records diagnostics for structurally invalid MARC records. The second native pass materializes only valid records; rejected records are never parsed one by one in R. Recovery is deliberately native-only: native-unsupported input fails rather than falling back to slow record-by-record R recovery. XML syntax errors remain fatal. Skipped records retain their original source positions, so `record_id` may contain gaps.

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

arrow::write_parquet(), *6*

diagnose_canonical, 3
diagnose_canonical(), *11*

marc_filter_records, 4
marc_has, 4
marcxml_to_parquet, 5
marcxml_to_parquet(), *9*
marcxmlr *(*marcxmlr-package*)*, 2
marcxmlr-package, 2

Ops.marc_query *(*marc_has*)*, 4

read_marcxml, 8
read_marcxml(), *7*

write_marcxml, 10
write_marcxml(), *3*