Package 'xmlrectr'

September 28, 2026

Type Package

Title Rectangle Arbitrary 'XML' into Analysis-Friendly Tables

Version 0.1.0

Description Converts arbitrary 'XML' into canonical node tables and analysis-friendly rectangular outputs without requiring a vocabulary-specific parser. Supports conservative structure proposals, explicit reusable profiles, advisory 'XSD' inspection, bounded streaming, 'CSV' and 'Parquet' output, an analyst-oriented single-table projection, and record-level parallel execution with automatic scheduling. The native implementation uses 'libxml2' for structural acceleration while the 'R' implementation remains the semantic reference.

License GPL-3

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

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

Encoding UTF-8

Depends R (>= 4.4.0)

Imports stats, tibble, tools, utils, xml2, XML

Suggests arrow, future, future.mirai, futurize, furrr, jsonlite, knitr, mori, pkgdown, progressr, purrr, rmarkdown, testthat (>= 3.0.0), yaml

SystemRequirements libxml2 development files, pkg-config; on Windows use the matching Rtools toolchain

NeedsCompilation yes

VignetteBuilder knitr

Config/testthat/edition 3

Author Lorenzo Isella [aut, cre]

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

Repository CRAN

Date/Publication 2026-09-28 10:30:02 UTC

Contents

`xmlrectr-package` *xmlrectr: generic XML rectangling*

Description

Convert arbitrary XML into canonical node tables and analysis-friendly rectangles with explicit profiles, bounded streaming, optional native acceleration, and automatic record-level parallel execution.

Details

The normal workflow is proposal, review, profile, compilation, and rectangling. For exploratory work, the analyst layer provides a self-contained single-table projection. See the package README and vignettes for complete workflows.

See Also

`propose_xml_profile()`, `xml_profile()`, `rectangle_xml()`, `rectangle_xml_analyst()`

---

as_xml_profile *Coerce a portable object to an XML profile*

Description

Discover structural evidence and define a reviewable, reusable extraction contract. Proposals and XSD evidence are advisory rather than silently executable.

Usage

```
as_xml_profile(x)
```

Arguments

`x`	An existing `xml_profile` object or a compatible list-like profile representation.

Value

An object of class `xml_profile`. The returned named list is a validated portable profile representation suitable for compilation, serialization, and reuse.

See Also

`xml_profile()`, `rectangle_xml()`

---

compile_xml_profile *Compile a profile into an executable rectangle specification*

Description

Discover structural evidence and define a reviewable, reusable extraction contract. Proposals and XSD evidence are advisory rather than silently executable.

Usage

```
compile_xml_profile(
profile,
sample,
whitespace = c("drop_blank", "preserve"))
```

Arguments

profile	A human-facing profile created by xml_profile() or coercible with as_xml_profile().
sample	A representative XML sample accepted by rectangle_spec(), or a previously inspected xml_structure.
whitespace	How blank text nodes are handled while inspecting the sample: "drop_blank" or "preserve".

Value

An object of class xml_rect_spec. It is an executable, namespace-aware rectangle specification resolved against sample and includes the originating validated xml_profile in its profile component.

See Also

xml_profile(), rectangle_xml()

---

inspect_xml *Inspect XML structure before defining a rectangle*

Description

Work with the canonical XML-node representation that preserves node identity, parentage, source order, namespaces, node types and scalar values.

Usage

```
inspect_xml(
x,
whitespace = c("drop_blank", "preserve"))
```

Arguments

x	An XML file path or an existing canonical node table.
whitespace	How blank text nodes are handled when an XML file is read.

Value

An object of class xml_structure, implemented as a list containing the source, whitespace policy, canonical node table, namespace-aware structural index, and likely record paths. It represents reviewable structural evidence about the XML sample.

See Also

xml_profile(), rectangle_xml()

---

inspect_xsd *Inspect advisory XSD evidence*

Description

Discover structural evidence and define a reviewable, reusable extraction contract. Proposals and XSD evidence are advisory rather than silently executable.

Usage

```
inspect_xsd(xsd)
```

Arguments

xsd	Path to an XML Schema (XSD) file to inspect as advisory structural evidence.

Details

This is deliberately an advisory XSD inspection layer, not a complete XSD validator or resolver.

Value

An object of class xml_xsd_inspection, implemented as a list containing the schema source, target namespace, a declaration tibble, and explicit inspection limitations. It represents advisory evidence from directly declared XSD elements and attributes rather than a complete schema validation result.

See Also

xml_profile(), rectangle_xml()

---

make_rectangle *Create a lower-level rectangle specification*

Description

Discover structural evidence and define a reviewable, reusable extraction contract. Proposals and XSD evidence are advisory rather than silently executable.

Usage

```
make_rectangle(
structure,
one_row_per = NULL,
namespace = NULL,
identify_by = NULL,
fields = NULL,
types = NULL,
representation = c("auto", "wide", "long"))
```

Arguments

structure	An inspected XML structure returned by inspect_xml().
one_row_per	Optional visible element name or path defining one output record. If omitted, an unambiguous proposed record path is used.
namespace	Optional namespace URI used to disambiguate the row element.
identify_by	Optional visible value name or path used as the record identifier; omit to generate sequential identifiers.
fields	Optional character vector of values to retain. Named entries rename output columns.
types	Optional named character vector declaring output types for selected fields.
representation	Requested table representation: automatic safe choice, wide, or long.

Value

An object of class xml_rect_spec, implemented as a list describing the resolved record path, identifier rule, representation, repeated paths, field mapping, and field catalogue. It is an executable rectangle specification compiled from the inspected structure.

See Also

xml_profile(), rectangle_xml()

---

propose_xml_profile *Propose candidate rows, identifiers and fields*

Description

Discover structural evidence and define a reviewable, reusable extraction contract. Proposals and XSD evidence are advisory rather than silently executable.

Usage

```
propose_xml_profile(
sample,
rows = NULL,
namespace = NULL,
xsd = NULL,
whitespace = c("drop_blank", "preserve"))
```

Arguments

sample	An XML sample accepted by `inspect_xml()` or an existing `xml_structure`.
rows	Optional visible row element name or path to evaluate instead of relying only on automatic row candidates.
namespace	Optional namespace URI used to disambiguate row candidates.
xsd	Optional XSD file whose directly declared structure is added as advisory evidence.
whitespace	How blank text nodes are handled while inspecting the XML sample.

Details

The returned proposal is review-only and is not an executable profile.

Value

An object of class `xml_profile_proposal`, implemented as a review-only list containing ranked row candidates, identifier candidates, field candidates, optional XSD evidence, and a portable profile template when a row structure can be proposed. It is not an executable `xml_profile`.

See Also

`xml_profile()`, `rectangle_xml()`

---

read_xml_profile *Read an XML profile from JSON or YAML*

Description

Discover structural evidence and define a reviewable, reusable extraction contract. Proposals and XSD evidence are advisory rather than silently executable.

Usage

```
read_xml_profile(
file,
format = c("auto", "json", "yaml"))
```

Arguments

file	Path to a serialized XML profile.
format	Profile serialization format. "auto" infers JSON or YAML from the filename extension.

Value

An object of class xml_profile, reconstructed and validated from the serialized JSON or YAML representation.

See Also

xml_profile(), rectangle_xml()

---

rectangle_spec *Create an explicit rectangle specification*

Description

Discover structural evidence and define a reviewable, reusable extraction contract. Proposals and XSD evidence are advisory rather than silently executable.

Usage

```
rectangle_spec(
sample,
one_row_per = NULL,
namespace = NULL,
identify_by = NULL,
fields = NULL,
types = NULL,
representation = c("auto", "wide", "long"),
whitespace = c("drop_blank", "preserve"))
```

Arguments

sample	An XML sample accepted by inspect_xml() or an existing xml_structure.
one_row_per	Optional visible element name or path defining one output record.
namespace	Optional namespace URI used to disambiguate the row element.
identify_by	Optional visible value name or path used as the record identifier.
fields	Optional character vector of values to retain. Named entries rename output columns.
types	Optional named character vector declaring output types for selected fields.
representation	Requested table representation: automatic safe choice, wide, or long.
whitespace	How blank text nodes are handled when the sample is read.

Value

An object of class `xml_rect_spec`. It is the executable rectangle specification obtained by inspecting `sample` and resolving the requested row, identifier, field, type, and representation choices.

See Also

`xml_profile()`, `rectangle_xml()`

`rectangle_xml` *Read and rectangle an XML file*

Description

Apply an explicit profile/specification while preserving repeated values and output order. Parallelism is an execution option of the same public workflow.

Usage

```
rectangle_xml(
file,
spec,
whitespace = NULL,
parallel = FALSE,
workers = NULL,
strategy = "auto",
chunk_records = NULL,
task_records = NULL,
progress = FALSE)
```

Arguments

`file`	Path to the XML file to rectangle in memory.
`spec`	A compiled rectangle specification or an `xml_profile`.
`whitespace`	Optional blank-text policy. When omitted, the policy stored in the specification/profile is used.
`parallel`	Execution mode: `FALSE` for sequential, `TRUE` for parallel, or `"auto"` to choose automatically.
`workers`	Optional positive number of parallel workers. When omitted, a balanced default is chosen.
`strategy`	Parallel scheduling strategy. `"auto"` chooses the context-appropriate default.
`chunk_records`	Optional number of complete records in an outer processing chunk.
`task_records`	Optional number of records assigned to each inner parallel task.
`progress`	Logical; whether to emit progress events through **progressr** for supported in-memory parallel execution.

Details

Use `parallel = FALSE` for the exact sequential path, `parallel = TRUE` to request tuned parallel defaults, or `parallel = "auto"` to let the engine avoid parallel overhead on small in-memory workloads.

Value

A tibble containing the rectangular analytical result for `file`, with the same single-table atomic-column contract as `xml_rectangle()`. Its columns and wide/long structure are determined by the supplied profile or compiled specification.

See Also

`xml_profile()`, `rectangle_xml()`

---

rectangle_xml_analyst *Create an analyst-oriented table directly from XML*

Description

Create or work with the exploratory single-table analyst projection, which keeps universal xml_* provenance/entity columns alongside atomic analytical values.

Usage

```
rectangle_xml_analyst(
file,
whitespace = c("drop_blank", "preserve"),
infer_types = TRUE,
include_misc = TRUE)
```

Arguments

`file`	Path to the XML file.
`whitespace`	How blank text nodes are handled.
`infer_types`	Logical; whether conservative scalar type inference is applied to analyst data columns.
`include_misc`	Logical; whether miscellaneous XML content/provenance columns are retained.

Value

A tibble in the same self-contained analyst-table format as `xml_analyst_table()`: one row per emitted XML entity occurrence, `xml_*` provenance/entity columns, atomic analytical columns, and the `xml_analyst_metadata` attribute.

See Also

`xml_profile()`, `rectangle_xml()`

rectangle_xml_analyst_csv *Write the analyst-oriented XML table to CSV*

Description

Create or work with the exploratory single-table analyst projection, which keeps universal xml_* provenance/entity columns alongside atomic analytical values.

Usage

```
rectangle_xml_analyst_csv(
file,
output,
whitespace = c("drop_blank", "preserve"),
infer_types = TRUE,
include_misc = TRUE,
na = "\\N",
overwrite = FALSE)
```

Arguments

file	Path to the XML file.
output	Destination CSV file.
whitespace	How blank text nodes are handled.
infer_types	Logical; whether conservative scalar type inference is applied.
include_misc	Logical; whether miscellaneous XML content/provenance columns are retained.
na	Character string written for missing values.
overwrite	Logical; whether an existing destination may be replaced.

Value

Invisibly, the analyst-table tibble that was written to output. The side effect is transactional publication of that same table as one CSV file.

See Also

xml_profile(), rectangle_xml()

`rectangle_xml_analyst_parquet` *Write the analyst-oriented XML table to Parquet*

Description

Create or work with the exploratory single-table analyst projection, which keeps universal xml_* provenance/entity columns alongside atomic analytical values.

Usage

```
rectangle_xml_analyst_parquet(
file,
output,
whitespace = c("drop_blank", "preserve"),
infer_types = TRUE,
include_misc = TRUE,
overwrite = FALSE,
compression = "snappy")
```

Arguments

`file`	Path to the XML file.
`output`	Destination Parquet file.
`whitespace`	How blank text nodes are handled.
`infer_types`	Logical; whether conservative scalar type inference is applied.
`include_misc`	Logical; whether miscellaneous XML content/provenance columns are retained.
`overwrite`	Logical; whether an existing destination may be replaced.
`compression`	Parquet compression codec passed to **arrow**.

Value

Invisibly, the analyst-table tibble that was written to `output`. The side effect is transactional publication of that same table as one Parquet file.

See Also

`xml_profile()`, `rectangle_xml()`

`rectangle_xml_csv` *Rectangle XML to a staged CSV file*

Description

Apply an explicit profile/specification while preserving repeated values and output order. Parallelism is an execution option of the same public workflow.

Usage

```
rectangle_xml_csv(
file,
spec,
output,
overwrite = FALSE,
document_id = NULL,
whitespace = NULL,
chunk_rows = 100000L,
batch_rows = 100000L,
id_check = c("memory", "none"),
parallel = FALSE,
workers = NULL,
strategy = "auto",
chunk_records = NULL,
task_records = NULL)
```

Arguments

`file`	Path to the XML file.
`spec`	A compiled rectangle specification.
`output`	Destination CSV file.
`overwrite`	Logical; whether an existing destination may be replaced.
`document_id`	Optional identifier attached to canonical rows; by default it is derived from the input file.
`whitespace`	Optional blank-text policy; when omitted, the specification policy is used.
`chunk_rows`	Positive number controlling the canonical SAX node-buffer capacity.
`batch_rows`	Maximum number of rectangled rows accumulated before a writer callback flush.
`id_check`	For source identifiers, `"memory"` checks global uniqueness; `"none"` avoids retaining the uniqueness set.
`parallel`	Execution mode: sequential, explicitly parallel, or automatic.
`workers`	Optional positive number of parallel workers.
`strategy`	Parallel scheduling strategy or `"auto"`.
`chunk_records`	Optional number of complete XML records in an outer parallel chunk.
`task_records`	Optional number of records per inner parallel task.

Details

The streaming parser and record-boundary detection remain coordinator-side. Parallel workers receive only complete independent record subtrees.

Value

Invisibly, a one-row tibble of streaming execution statistics augmented with `output_file` and `output_format`. The main side effect is transactional publication of the completed CSV at `output`.

See Also

`xml_profile()`, `rectangle_xml()`

---

rectangle_xml_parallel

Read and rectangle XML with explicit parallel execution

Description

Lower-level explicit parallel entry point retained for diagnostics, regression tests and advanced tuning. Routine use should normally use the corresponding ordinary function with the parallel argument.

Usage

```
rectangle_xml_parallel(
file,
spec,
whitespace = NULL,
workers = NULL,
strategy = c("shared_chunk", "parallel_chunks"),
chunk_records = NULL,
task_records = NULL,
progress = FALSE)
```

Arguments

`file`	Path to the XML file.
`spec`	A compiled rectangle specification or an `xml_profile`.
`whitespace`	Optional blank-text policy.
`workers`	Optional positive number of parallel workers.
`strategy`	Parallel scheduling strategy.
`chunk_records`	Optional number of complete records in an outer processing chunk.
`task_records`	Optional number of records per inner task.
`progress`	Logical; whether to emit progress events through **progressr**.

Value

A tibble containing the same rectangular analytical result and atomic-column structure as `rectangle_xml()`, after reading the XML file and executing the compiled/profile rectangle in parallel.

See Also

`xml_profile()`, `rectangle_xml()`

`rectangle_xml_parquet` *Rectangle XML to a staged Parquet dataset*

Description

Apply an explicit profile/specification while preserving repeated values and output order. Parallelism is an execution option of the same public workflow.

Usage

```
rectangle_xml_parquet(
file,
spec,
output_dir,
overwrite = FALSE,
compression = "snappy",
document_id = NULL,
whitespace = NULL,
chunk_rows = 100000L,
batch_rows = 100000L,
id_check = c("memory", "none"),
parallel = FALSE,
workers = NULL,
strategy = "auto",
chunk_records = NULL,
task_records = NULL)
```

Arguments

`file`	Path to the XML file.
`spec`	A compiled rectangle specification.
`output_dir`	Destination directory for the Parquet dataset.
`overwrite`	Logical; whether an existing destination may be replaced.
`compression`	Parquet compression codec passed to **arrow**.
`document_id`	Optional identifier attached to canonical rows; by default it is derived from the input file.
`whitespace`	Optional blank-text policy; when omitted, the specification policy is used.

chunk_rows	Positive number controlling the canonical SAX node-buffer capacity.
batch_rows	Maximum number of rectangled rows accumulated before a writer flush.
id_check	For source identifiers, "memory" checks global uniqueness; "none" avoids retaining the uniqueness set.
parallel	Execution mode: sequential, explicitly parallel, or automatic.
workers	Optional positive number of parallel workers.
strategy	Parallel scheduling strategy or "auto".
chunk_records	Optional number of complete XML records in an outer parallel chunk.
task_records	Optional number of records per inner parallel task.

Details

The streaming parser and record-boundary detection remain coordinator-side. Parallel workers receive only complete independent record subtrees.

Value

Invisibly, a one-row tibble of streaming execution statistics augmented with the published output directory, format, number of Parquet parts, and compression setting. The main side effect is transactional publication of the completed Parquet dataset.

See Also

xml_profile(), rectangle_xml()

---

review_rectangle *Review a compiled rectangle specification*

Description

Discover structural evidence and define a reviewable, reusable extraction contract. Proposals and XSD evidence are advisory rather than silently executable.

Usage

```
review_rectangle(spec)
```

Arguments

spec	A compiled rectangle specification returned by rectangle_spec(), make_rectangle(), or compile_xml_profile().

Value

A tibble giving the atomic output-field mapping of an xml_rect_spec. It reports each output name, source, declared type, value kind, owning entity, whether it lies under repetition, and its observed maximum multiplicity per record.

See Also

xml_profile(), rectangle_xml()

---

review_xml_proposal *Review one part of an XML profile proposal*

Description

Discover structural evidence and define a reviewable, reusable extraction contract. Proposals and XSD evidence are advisory rather than silently executable.

Usage

```
review_xml_proposal(
proposal,
what = c("rows", "ids", "fields", "xsd"))
```

Arguments

proposal	A proposal returned by propose_xml_profile().
what	Which proposal component to review: row candidates, ID candidates, fields, or XSD evidence.

Value

A tibble containing the requested review section of an xml_profile_proposal. Depending on what, it contains row candidates, identifier candidates, field candidates, or XSD declarations, with internal matching keys removed from the user-facing candidate tables.

See Also

xml_profile(), rectangle_xml()

---

validate_xml_nodes *Validate the canonical XML node-table contract*

Description

Work with the canonical XML-node representation that preserves node identity, parentage, source order, namespaces, node types and scalar values.

Usage

```
validate_xml_nodes(nodes)
```

Arguments

nodes	A data frame intended to satisfy the canonical XML node-table contract.

Value

Invisibly returns TRUE when nodes satisfies the canonical node-table invariants. The function otherwise stops with an error describing the violated invariant.

See Also

xml_profile(), rectangle_xml()

---

write_xml_profile *Write an XML profile to JSON or YAML*

Description

Discover structural evidence and define a reviewable, reusable extraction contract. Proposals and XSD evidence are advisory rather than silently executable.

Usage

```
write_xml_profile(
profile,
file,
format = c("auto", "json", "yaml"),
overwrite = FALSE)
```

Arguments

profile	An xml_profile or compatible profile object.
file	Destination JSON or YAML file.
format	Serialization format. "auto" infers it from the destination extension.
overwrite	Logical; whether an existing destination may be replaced.

Value

Invisibly, a character scalar giving the normalized path of the JSON or YAML file that was written. Writing the serialized profile is the function's side effect.

See Also

xml_profile(), rectangle_xml()

xml_analyst_expand_context *Expand analyst-table entity context*

Description

Create or work with the exploratory single-table analyst projection, which keeps universal xml_* provenance/entity columns alongside atomic analytical values.

Usage

```
xml_analyst_expand_context(x, entity = NULL, levels = Inf)
```

Arguments

x	A table returned by `xml_analyst_table()` or `rectangle_xml_analyst()`.
entity	Optional entity name or vector of entity names to retain.
levels	Maximum number of ancestor-entity levels from which contextual values may be propagated; `Inf` means all available levels.

Value

A tibble with the same columns as `x`, optionally filtered by entity, in which missing analytical cells are filled from owning fields of the requested ancestor levels. Provenance/entity columns are retained so the expanded values remain traceable to the analyst-table hierarchy.

See Also

`xml_profile()`, `rectangle_xml()`

xml_analyst_table *Project canonical XML into one analyst-oriented table*

Description

Create or work with the exploratory single-table analyst projection, which keeps universal xml_* provenance/entity columns alongside atomic analytical values.

Usage

```
xml_analyst_table(
nodes,
infer_types = TRUE,
include_misc = TRUE,
.validate_nodes = TRUE)
```

Arguments

nodes	A canonical XML node table.
infer_types	Logical; whether conservative scalar type inference is applied to analyst data columns.
include_misc	Logical; whether miscellaneous XML content/provenance columns are retained.
.validate_nodes	Internal logical fast-path control. Ordinary callers should leave this at TRUE.

Value

A tibble representing one self-contained analyst-oriented table. Each row is an XML entity occurrence with xml_* provenance/entity columns plus atomic analytical value columns; the table also carries an xml_analyst_metadata attribute containing entity/path mappings, column sources, amplification diagnostics, and audit information.

See Also

xml_profile(), rectangle_xml()

---

xml_nodes_schema *Return the canonical XML node-table schema*

Description

Work with the canonical XML-node representation that preserves node identity, parentage, source order, namespaces, node types and scalar values.

Usage

```
xml_nodes_schema()
```

Value

A zero-row tibble with the 12 canonical XML node-table columns and their stable atomic types. It is the executable schema used to initialize and validate canonical node output.

See Also

xml_profile(), rectangle_xml()

xml_node_children *Return direct canonical children of XML nodes*

Description

Work with the canonical XML-node representation that preserves node identity, parentage, source order, namespaces, node types and scalar values.

Usage

```
xml_node_children(
nodes,
node_id,
document_id = NULL,
include_attributes = FALSE)
```

Arguments

nodes	A canonical XML node table.
node_id	Positive canonical node identifier whose children are requested.
document_id	Optional document identifier used to disambiguate node IDs in combined tables.
include_attributes	Logical; whether attribute rows owned by the node are included with child content.

Value

A tibble containing the direct children of node_id, using the canonical node-table columns. Rows are returned in XML content order, with attributes optionally included ahead of content children.

See Also

xml_profile(), rectangle_xml()

xml_profile *Create a human-facing reusable XML profile*

Description

Discover structural evidence and define a reviewable, reusable extraction contract. Proposals and XSD evidence are advisory rather than silently executable.

Usage

```
xml_profile(
rows,
id = NULL,
fields = NULL,
types = NULL,
namespace = NULL,
layout = c("safe", "wide", "long"))
```

Arguments

rows	Visible XML element name or path defining one output record.
id	Optional visible value name/path used as record identifier; omit or use FALSE for generated IDs.
fields	Optional character vector of values to retain. Named entries rename output columns.
types	Optional named character vector declaring output types.
namespace	Optional namespace URI used to disambiguate the row element.
layout	One-table layout policy: safe automatic layout, explicitly wide, or explicitly long.

Value

An object of class xml_profile, implemented as a named list containing the portable user choices rows, id, fields, types, namespace, and layout together with a profile version. It does not contain sample-specific internal path keys.

See Also

xml_profile(), rectangle_xml()

---

xml_rectangle *Rectangle a canonical XML node table*

Description

Apply an explicit profile/specification while preserving repeated values and output order. Parallelism is an execution option of the same public workflow.

Usage

```
xml_rectangle(
nodes,
spec,
parallel = FALSE,
workers = NULL,
strategy = "auto",
chunk_records = NULL,
task_records = NULL,
progress = FALSE)
```

Arguments

nodes	A canonical XML node table.
spec	A compiled rectangle specification or an xml_profile.
parallel	Execution mode: FALSE, TRUE, or "auto".
workers	Optional positive number of parallel workers.
strategy	Parallel scheduling strategy or "auto".
chunk_records	Optional number of complete records in an outer processing chunk.
task_records	Optional number of records per inner parallel task.
progress	Logical; whether to emit progress events through **progressr**.

Details

Use parallel = FALSE for the exact sequential path, parallel = TRUE to request tuned parallel defaults, or parallel = "auto" to let the engine avoid parallel overhead on small in-memory workloads.

Value

A tibble containing the rectangular analytical result defined by spec. The table is always a single data frame with atomic columns; its exact columns and wide/long structure are determined by the compiled specification while preserving record and repetition semantics.

See Also

xml_profile(), rectangle_xml()

`xml_rectangle_parallel` *Rectangle canonical nodes with explicit parallel execution*

Description

Lower-level explicit parallel entry point retained for diagnostics, regression tests and advanced tuning. Routine use should normally use the corresponding ordinary function with the parallel argument.

Usage

```
xml_rectangle_parallel(
nodes,
spec,
workers = NULL,
strategy = c("shared_chunk", "parallel_chunks"),
chunk_records = NULL,
task_records = NULL,
progress = FALSE)
```

Arguments

`nodes`	A canonical XML node table.
`spec`	A compiled rectangle specification or an `xml_profile`.
`workers`	Optional positive number of parallel workers.
`strategy`	Parallel scheduling strategy.
`chunk_records`	Optional number of complete records in an outer processing chunk.
`task_records`	Optional number of records per inner task.
`progress`	Logical; whether to emit progress events through **progressr**.

Value

A tibble containing the same rectangular analytical result and atomic-column structure as `xml_rectangle()`, computed with explicit record-level parallel execution.

See Also

`xml_profile()`, `rectangle_xml()`

`xml_stream_nodes` *Stream canonical XML node batches*

Description

Work with the canonical XML-node representation that preserves node identity, parentage, source order, namespaces, node types and scalar values.

Usage

```
xml_stream_nodes(
file,
callback,
document_id = NULL,
whitespace = c("preserve", "drop_blank"),
chunk_rows = 100000L)
```

Arguments

`file`	Path to the XML file.
`callback`	Function called with each emitted canonical node-table chunk.
`document_id`	Optional identifier attached to canonical rows; by default it is derived from the input file.
`whitespace`	How blank text nodes are handled.
`chunk_rows`	Positive maximum number of canonical rows buffered before the callback is invoked.

Value

Invisibly, a one-row tibble containing `document_id`, total `node_count`, and emitted `chunk_count`. The main side effect is repeated invocation of `callback` with bounded canonical node-table chunks.

See Also

`xml_profile()`, `rectangle_xml()`

`xml_stream_rectangle` *Stream complete XML records through a rectangle callback*

Description

Apply an explicit profile/specification while preserving repeated values and output order. Parallelism is an execution option of the same public workflow.

Usage

```
xml_stream_rectangle(
file,
spec,
callback,
document_id = NULL,
whitespace = NULL,
chunk_rows = 100000L,
batch_rows = 100000L,
id_check = c("memory", "none"),
parallel = FALSE,
workers = NULL,
strategy = "auto",
chunk_records = NULL,
task_records = NULL)
```

Arguments

`file`	Path to the XML file.
`spec`	A compiled rectangle specification.
`callback`	Function called with each emitted rectangled result batch.
`document_id`	Optional identifier attached to canonical rows.
`whitespace`	Optional blank-text policy; when omitted, the specification policy is used.
`chunk_rows`	Positive number controlling the canonical SAX node-buffer capacity.
`batch_rows`	Maximum number of rectangled rows passed to a callback batch.
`id_check`	For source identifiers, `"memory"` checks global uniqueness; `"none"` avoids retaining the uniqueness set.
`parallel`	Execution mode: sequential, explicitly parallel, or automatic.
`workers`	Optional positive number of parallel workers.
`strategy`	Parallel scheduling strategy or `"auto"`.
`chunk_records`	Optional number of complete XML records in an outer parallel chunk.
`task_records`	Optional number of records per inner parallel task.

Details

The streaming parser and record-boundary detection remain coordinator-side. Parallel workers receive only complete independent record subtrees.

Value

Invisibly, a one-row tibble of streaming execution statistics, including document/node counts, record count, emitted result rows and chunks, maximum record size, and identifier-check status; parallel execution also reports scheduling statistics. Rectangular result batches are delivered through `callback` as the main side effect.

See Also

`xml_profile()`, `rectangle_xml()`

---

xml_stream_rectangle_parallel

Stream XML records with explicit parallel execution

Description

Lower-level explicit parallel entry point retained for diagnostics, regression tests and advanced tuning. Routine use should normally use the corresponding ordinary function with the parallel argument.

Usage

```
xml_stream_rectangle_parallel(
file,
spec,
callback,
document_id = NULL,
whitespace = NULL,
chunk_rows = 100000L,
batch_rows = 100000L,
id_check = c("memory", "none"),
workers = NULL,
strategy = c("shared_chunk", "parallel_chunks"),
chunk_records = NULL,
task_records = NULL)
```

Arguments

`file`	Path to the XML file.
`spec`	A compiled rectangle specification.
`callback`	Function called with each emitted rectangled result batch.

document_id	Optional identifier attached to canonical rows.
whitespace	Optional blank-text policy; when omitted, the specification policy is used.
chunk_rows	Positive number controlling the canonical SAX node-buffer capacity.
batch_rows	Maximum number of rectangled rows passed to a callback batch.
id_check	For source identifiers, "memory" checks global uniqueness; "none" avoids retaining the uniqueness set.
workers	Optional positive number of parallel workers.
strategy	Parallel scheduling strategy.
chunk_records	Optional number of complete XML records in an outer parallel chunk.
task_records	Optional number of records per inner parallel task.

Value

Invisibly, a one-row tibble of streaming and parallel execution statistics: document/node counts, parser chunks, record and result counts, maximum record size, identifier-check status, strategy, worker count, parallel chunk/task counts, buffering, and chunk/task sizes. Rectangular result batches are delivered through `callback` as the main side effect.

See Also

`xml_profile()`, `rectangle_xml()`

---

xml_text_to_nodes_memory

Read XML text into the canonical node table

Description

Work with the canonical XML-node representation that preserves node identity, parentage, source order, namespaces, node types and scalar values.

Usage

```
xml_text_to_nodes_memory(
text,
document_id = "inline-xml",
whitespace = c("preserve", "drop_blank"),
initial_capacity = 1024L)
```

Arguments

text	One character string containing XML markup.
document_id	Identifier assigned to all canonical rows from the supplied XML text.
whitespace	How blank text nodes are handled.
initial_capacity	Positive initial size of the internal row buffer; it grows automatically as needed.

Value

A tibble with one row per retained node in the supplied XML text and the 12 canonical node-table columns describing document identity, tree position, node type, names, namespace information, and scalar value.

See Also

`xml_profile()`, `rectangle_xml()`

`xml_to_nodes_memory` *Read an XML file into the canonical node table*

Description

Work with the canonical XML-node representation that preserves node identity, parentage, source order, namespaces, node types and scalar values.

Usage

```
xml_to_nodes_memory(
file,
document_id = NULL,
whitespace = c("preserve", "drop_blank"),
initial_capacity = 1024L)
```

Arguments

`file`	Path to the XML file.
`document_id`	Optional identifier attached to all canonical rows; by default it is derived from the input file.
`whitespace`	How blank text nodes are handled.
`initial_capacity`	Positive initial size of the internal row buffer; it grows automatically as needed.

Value

A tibble with one row per retained XML node and the 12 canonical node-table columns describing document identity, tree position, node type, names, namespace information, and scalar value.

See Also

`xml_profile()`, `rectangle_xml()`

`xml_to_nodes_stream` *Read XML with the sequential streaming canonical engine*

Description

Work with the canonical XML-node representation that preserves node identity, parentage, source order, namespaces, node types and scalar values.

Usage

```
xml_to_nodes_stream(
file,
document_id = NULL,
whitespace = c("preserve", "drop_blank"),
chunk_rows = 100000L)
```

Arguments

`file`	Path to the XML file.
`document_id`	Optional identifier attached to canonical rows; by default it is derived from the input file.
`whitespace`	How blank text nodes are handled.
`chunk_rows`	Positive maximum number of canonical rows buffered per emitted chunk.

Value

A complete canonical XML node-table tibble assembled from the streaming reader, with one row per retained XML node and the same 12 columns and types as `xml_nodes_schema()`.

See Also

`xml_profile()`, `rectangle_xml()`

Index