# Package 'dataset'

June 3, 2026

**Title** Create Data Frames for Exchange and Reuse

**Version** 0.4.5

**Date** 2026-06-03

**Language** en-GB

**Maintainer** Daniel Antal <daniel.antal@dataobservatory.eu>

**Description** The 'dataset' package extends tidy data frames with machine-readable metadata, semantic definitions, and provenance information. It supports incremental semantic stabilization, interoperable dataset exchange, and FAIR-oriented publication workflows by preserving contextual metadata directly within R objects. The package facilitates the creation, exchange, reuse, and RDF serialization of datasets in line with ISO and W3C standards.

**License** GPL (>= 3)

**Encoding** UTF-8

**URL** https://docs.ropensci.org/dataset/, https://github.com/ropensci/dataset, https://dataset.dataobservatory.eu

**BugReports** https://github.com/ropensci/dataset/issues

**LazyData** true

**Imports** assertthat, haven, ISOcodes, labelled, pillar, stats, tibble, utils, vctrs

**Suggests** dplyr, jsonld, knitr, rdflib, rmarkdown, spelling, tidyr, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**Depends** R (>= 3.5)

**VignetteBuilder** knitr

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** no

**Author** Daniel Antal [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7513-6760>),
Marcelo Perlin [rev] (ORCID: <https://orcid.org/0000-0002-9839-4268>),
Anna Márta Mester [rev] (ORCID:
<https://orcid.org/0009-0008-2274-8163>),
Mauro Lepore [rev] (ORCID: <https://orcid.org/0000-0002-1986-7988>)

**Repository** CRAN

**Date/Publication** 2026-06-03 09:10:14 UTC

# Contents

---

`as.data.frame.dataset_df` *Convert a* `dataset_df` *to a base* `data.frame`

---

## Description

Converts a `dataset_df` into a plain `data.frame`. By default this strips semantic metadata (label, unit, concept/definition, namespace) from each column, but this can be controlled via the `strip_attributes` argument.

Dataset-level metadata remains attached as inert attributes.

## Usage

```
## S3 method for class 'dataset_df'
as.data.frame(
  x,
  ...,
  strip_attributes = TRUE,
  optional = FALSE,
  stringsAsFactors = FALSE
)
```

## Arguments

| | |
|---|---|
| `x` | A `dataset_df`. |
| `...` | Passed to `base::as.data.frame()`. |
| `strip_attributes` | Logical: should column-level semantic metadata be stripped? Default: `TRUE`. |

optional — logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional. Note that all of R's **base** package as.data.frame() methods use optional only for column names treatment, basically with the meaning of data.frame(*, check.names = !optional). See also the make.names argument of the matrix method.

stringsAsFactors — logical: should the character vector be converted to a factor?

## Value

A base R data.frame without the dataset_df class.

## Examples

```
data(orange_df)
as.data.frame(orange_df)
```

---

# as.Date.haven_labelled_defined

*Coerce a defined Date vector to a base R Date*

---

## Description

Coerces a haven_labelled_defined vector whose underlying type is Date into a base R Date vector.

This method preserves the underlying date values and, by default, also retains any semantic metadata attached to the variable.

## Usage

```
## S3 method for class 'haven_labelled_defined'
as.Date(x, strip_attributes = FALSE, ...)
```

## Arguments

x — A vector created with defined() with underlying type Date.

strip_attributes — Logical; should the semantic metadata attributes (label, unit, definition, namespace) be removed from the returned vector? Defaults to FALSE.

... — Additional arguments passed to base::as.Date().

## Details

Use strip_attributes = TRUE when flattening or preparing data for external pipelines, but keep the default when working with defined vectors directly.

Base R's as.Date() also works, as it dispatches to this method via S3. However, using as.Date() on defined vectors is considered safe because this method ensures metadata is handled predictably.

## Value

A `Date` vector, optionally carrying semantic metadata.

## See Also

`as.POSIXct()`, `as_numeric()`, `as_character()`, `as_logical()`, `defined()`

## Examples

```
d <- defined(Sys.Date() + 0:2, label = "Observation date")

# Recommended usage
as.Date(d)

# Stripping metadata
as.Date(d, strip_attributes = TRUE)
```

---

# as.POSIXct.haven_labelled_defined

*Coerce a defined POSIXct vector to a base R POSIXct*

---

## Description

Coerces a `haven_labelled_defined` vector whose underlying type is `POSIXct` into a base R `POSIXct` time vector.

This method preserves both the timestamp values and the original time zone. By default, semantic metadata is also retained.

## Usage

```
## S3 method for class 'haven_labelled_defined'
as.POSIXct(x, tz = "", strip_attributes = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `x` | A vector created with `defined()` with underlying type `POSIXct`. |
| `tz` | a character string. The time zone specification to be used for the conversion, if one is required. System-specific timezones (see `base::timezones()`, but "" is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning. |
| `strip_attributes` | Logical; should semantic metadata attributes (label, unit, definition, namespace) be removed? Defaults to `FALSE`. |
| `...` | Additional arguments passed to `base::as.POSIXct()`. |

## Details

Use `strip_attributes = TRUE` when flattening or preparing data for external pipelines, but keep the default when working with defined vectors directly.
Base R's `as.POSIXct()` also works, as it dispatches to this method via S3. Using this method directly is preferred when metadata preservation matters.

## Value

A `POSIXct` vector with timestamp values preserved.

## See Also

`as.Date()`, `as_numeric()`, `as_character()`, `as_logical()`, `defined()`

## Examples

```
p <- defined(
  as.POSIXct("2024-01-01 12:00:00", tz = "UTC"),
  label = "Timestamp"
)

# Recommended usage
as.POSIXct(p)

# Explicit attribute stripping
as.POSIXct(p, strip_attributes = TRUE)
```

---

# as_character *Coerce a defined vector to character*

---

## Description

`as_character()` is the recommended method to convert a defined() vector into a character vector. It is metadata-aware and provides explicit control over whether semantic attributes are preserved.

Base R `as.character()` always strips the class and metadata.

Convert objects into semantic character representations.

## Usage

```
as_character(x, ...)

## S3 method for class 'haven_labelled_defined'
as_character(x, strip_attributes = TRUE, ...)

## S3 method for class 'haven_labelled_defined'
as.character(x, ...)

as_character(x, ...)
```

### Arguments

| | |
|---|---|
| x | An object. |
| ... | Additional arguments. |
| strip_attributes | Logical; should semantic metadata attributes (such as label, unit, definition, and namespace) be removed from the returned vector? Defaults to FALSE. |

### Details

If preserve_attributes = TRUE, the returned character vector retains metadata attributes (unit, concept, namespace, label). The "defined" class is always removed.

If preserve_attributes = FALSE (default), a plain character vector is returned with *all* metadata stripped.

Use strip_attributes = TRUE when flattening or preparing data for external pipelines, but keep the default when working with defined vectors directly.

Base R's as.character() always drops all attributes and returns plain character values. It is equivalent to: as_character(x, strip_attributes = TRUE).

### Value

A character vector (plain or with attributes).

### Examples

```
x <- defined(c("apple", "banana"), label = "Fruit", unit = "kg")

# Recommended:
as_character(x)

# Preserve metadata:
as_character(x, strip_attributes = FALSE)

# Base R:
as.character(x)
```

---

## as_character.prelabelled *Coerce prelabelled vectors to semantic character workspace*

---

### Description

Convert a prelabelled vector into a semantic character workspace suitable for iterative refinement and inference workflows.

## Usage

```
## S3 method for class 'prelabelled'
as_character(x, ...)
```

## Arguments

| | |
|---|---|
| x | A prelabelled vector. |
| ... | Unused. |

## Details

Unlike base as.character(), this method preserves:

- original observational values;
- semantic label mappings;
- refinement metadata;
- additional custom attributes.

The method operationalises provisional semantic labels into working character values while retaining the original observed vector for reversibility and provenance-aware workflows.

This allows workflows such as:

- prelabel() %>% as.character() %>% refine()
- prelabel() %>% as.character() %>% infer()
- iterative semantic harmonisation;
- contextual semantic derivation.

The original observational values are preserved in the "original_values" attribute.

The method intentionally separates:

- observational evidence;
- semantic operationalisation.

This distinction is important for provenance-aware semantic refinement workflows where semantic interpretations may evolve while original observations remain stable.

The coercion is therefore semantically reversible.

## Value

A character vector where:

- values are derived from the semantic labels;
- original observations are preserved as attributes;
- refinement metadata is retained;
- the "prelabelled" class is removed.

## Examples

```
x <- prelabel(
  c(
    "r",
    "pdf",
    "qmd"
  ),
  labels = c(
    r = "software development",
    pdf = "publication",
    qmd = "documentation"
  )
)

x

y <- as_character(x)

y

attributes(y)

attr(
  y,
  "original_values"
)
```

---

`as_datacite` *Create a Bibentry Object with DataCite Metadata Fields*

---

## Description

Constructs a bibliographic metadata record conforming to the DataCite Metadata Schema. The resulting object is stored as a modified `utils::bibentry()` enriched with structured Dublin Core and DataCite-compliant metadata.

## Usage

```
as_datacite(x, type = "bibentry", ...)

datacite(
  Title,
  Creator,
  Identifier = NULL,
  Publisher = NULL,
  PublicationYear = NULL,
  Subject = subject_create(term = "data sets", subjectScheme =
    "Library of Congress Subject Headings (LCSH)", schemeURI =
```

```
    "https://id.loc.gov/authorities/subjects.html", valueURI =
    "http://id.loc.gov/authorities/subjects/sh2018002256"),
  Type = "Dataset",
  Contributor = NULL,
  Date = ":tba",
  DateList = NULL,
  Language = NULL,
  AlternateIdentifier = ":unas",
  RelatedIdentifier = ":unas",
  Format = ":tba",
  Version = "0.1.0",
  Rights = ":tba",
  Description = ":tba",
  Geolocation = ":unas",
  FundingReference = ":unas"
)

is.datacite(x)

## S3 method for class 'datacite'
is.datacite(x)

## S3 method for class 'datacite'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x | An object that is tested if it has a class "datacite". |
| type | A DataCite 4.4 metadata can be returned as: "list", "dataset_df", "bibentry" (default), or "ntriples". |
| ... | Optional parameters to add to a datacite object. For example, author = person("Jane", "Doe") adds an author if type = "dataset_df". |
| Title | The name(s) by which the resource is known. Similar to dct:title. |
| Creator | One or more utils::person() objects describing the main authors or contributors responsible for creating the resource. |
| Identifier | A persistent identifier (e.g., DOI or URI). May refer to a specific version or all versions of the resource. |
| Publisher | The name of the organization that holds, publishes, or distributes the resource. Required by DataCite. See publisher(). |
| PublicationYear | The year of public availability (in YYYY format). See publication_year(). |
| Subject | A topic, keyword, or classification term. See subject() and subject_create() for structured vocabularies. |
| Type | The resource type. Defaults to "Dataset" for general use. See dcm:type. |
| Contributor | An individual or institution that contributed to the development, distribution, or curation of the resource. |

| | |
|---|---|
| Date | A date in "YYYY", "YYYY-MM-DD" or ISO datetime format. Can also be a Date or POSIXct object. |
| DateList | A list of multiple dates. Currently not supported. |
| Language | Language code as per IETF BCP 47 / ISO 639-1. See language(). |
| AlternateIdentifier | Optional local or secondary identifier. Defaults to ":unas". |
| RelatedIdentifier | Related resources (e.g., prior versions, papers). Defaults to ":unas". |
| Format | A technical format (e.g., "application/pdf", "text/csv"). |
| Version | A free-text version string (e.g., "1.0.0"). Defaults to "0.1.0". See version(). |
| Rights | Licensing or usage restrictions for the resource. Defaults to ":tba". See rights(). |
| Description | Free-text summary or additional information. Defaults to ":tba". |
| Geolocation | Geographic location covered or referenced by the resource. See geolocation(). |
| FundingReference | Information about funding or financial support. Defaults to ":unas". Structured funding metadata not yet implemented. |

## Details

DataCite is a leading non-profit organization that provides persistent identifiers (DOIs) for research data and other research outputs. Members of the research community use DataCite to register datasets with globally resolvable metadata for citation and discovery.

This function sets "Dataset" as the default resource type. The Size attribute (e.g., bytes, pages, etc.) is automatically added if available.

## Value

as_datacite(x, type) returns the DataCite bibliographical metadata of x either as a list, a bibentry object, an N-Triples text serialisation or a dataset_df object.

A utils::bibentry() object with DataCite-compliant fields. Use as_datacite() to extract the metadata as a list or bibentry object.

is.datacite() returns a logical values (if the object x is of class datacite).

## Source

- DataCite 4.3 Mandatory Properties
- DataCite 4.3 Optional Properties

## See Also

Learn more in the vignette: bibrecord

Other bibrecord functions: as_dublincore(), bibrecord()

## Examples

```
datacite(
  Title = "Growth of Orange Trees",
  Creator = c(
    person(
      given = "N.R.",
      family = "Draper",
      role = "cre",
      comment = c(VIAF = "http://viaf.org/viaf/84585260")
    ),
    person(
      given = "H",
      family = "Smith",
      role = "cre"
    )
  ),
  Publisher = "Wiley",
  Date = 1998,
  Language = "en"
)

# Extract bibliographic metadata
as_datacite(orange_df)

# As a list
as_datacite(orange_df, "list")
```

---

# as_dublincore *Add or Retrieve Dublin Core Metadata*

---

## Description

Adds or retrieves metadata conforming to the Dublin Core Metadata Terms standard, enabling consistent and structured citation and retrieval of R dataset objects.

`is.dublincore()` checks whether an object inherits from the `"dublincore"` class.

## Usage

```
as_dublincore(x, type = "bibentry", ...)

dublincore(
  title,
  creator,
  contributor = NULL,
  year = NULL,
  publisher = NULL,
  identifier = NULL,
```

```
  subject = NULL,
  type = "DCMITYPE:Dataset",
  dataset_date = NULL,
  language = NULL,
  relation = NULL,
  dataset_format = "application/r-rds",
  rights = NULL,
  datasource = NULL,
  description = NULL,
  coverage = NULL
)

is.dublincore(x)

## S3 method for class 'dublincore'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x | An object to test. |
| type | The resource type. For datasets, use "Dataset". See DCMI Type Vocabulary. |
| ... | Additional metadata fields. |
| title | A name given to the resource. See dataset_title(). |
| creator | One or more utils::person() objects representing the creator(s). See creator(). |
| contributor | Additional contributors (utils::person()) with optional roles. See contributor(). |
| year | An explicit publication year. If omitted, inferred from dataset_date. |
| publisher | A character or utils::person() indicating the publishing entity. See publisher(). |
| identifier | A unique persistent identifier (e.g., DOI). See identifier(). |
| subject | A keyword or controlled vocabulary term. See subject() and subject_create(). |
| dataset_date | A publication or release date (Date, POSIXct, or character in YYYY, YYYY-MM-DD, or ISO format). |
| language | ISO 639-1 language code. See language(). |
| relation | A related resource (e.g., version, paper, or parent dataset). Currently only supports an URI, for example, "https://doi.org/10.32614/CRAN.package.dataset". |
| dataset_format | The technical format of the dataset (e.g., MIME type). See dataset_format(). |
| rights | A string describing intellectual property or usage rights. Use a URI like "https://creativecommons.org |
| datasource | A URL or label for the original source of the dataset. |
| description | A free-text summary of the dataset. See description(). |
| coverage | Geographic or temporal extent (spatial/temporal coverage). |

## Details

The Dublin Core Metadata Element Set (DCMES) is a standardized vocabulary for describing digital and physical resources. It includes 15 core fields and is formally standardized as ISO 15836, IETF RFC 5013, and ANSI/NISO Z39.85.

This function constructs a `utils::bibentry()` object extended with DCMI terms and is compatible with `dataset_df()` objects. The resulting metadata can be used for semantic documentation and machine-readable citation.

For compatibility with `utils::bibentry()`, the `dataset_date` parameter is automatically used to derive both `publication_date` and `year` fields.

## Value

A `bibentry` object extended with class `"bibrecord"`, storing structured Dublin Core metadata. Use `as_dublincore()` to extract the metadata in list, tabular, or RDF form.

A logical value: `TRUE` if `x` is a Dublin Core metadata record (i.e., inherits from `"dublincore"`), otherwise `FALSE`.

## Source

- DCMI Metadata Terms

## See Also

Learn more in the vignette: bibrecord

Other bibrecord functions: `as_datacite()`, `bibrecord()`

## Examples

```
orange_bibentry <- dublincore(
  title = "Growth of Orange Trees",
  creator = c(
    person(
      given = "N.R.",
      family = "Draper",
      role = "cre",
      comment = c(VIAF = "http://viaf.org/viaf/84585260")
    ),
    person(given = "H", family = "Smith", role = "cre")
  ),
  contributor = person(given = "Antal", family = "Daniel", role = "dtm"),
  publisher = "Wiley",
  datasource = "https://isbnsearch.org/isbn/9780471170822",
  dataset_date = 1998,
  identifier = "https://doi.org/10.5281/zenodo.14917851",
  language = "en",
  description = "The Orange data frame has 35 rows and 3 columns of records
                 of the growth of orange trees."
)
```

```
# To inspect structured metadata from a dataset_df object:
as_dublincore(orange_df, type = "list")
```

---

`as_factor` *Coerce a defined vector to a factor*

---

## Description

`as_factor()` converts a `defined()` vector into a standard R factor. If value labels are present, they are turned into factor levels via `haven::as_factor()`. Otherwise, the underlying values are converted with `base::factor()`.

## Usage

```
as_factor(x, ...)

## S3 method for class 'haven_labelled_defined'
as_factor(x, strip_attributes = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `x` | A vector created with `defined()`. |
| `...` | Reserved for future extensions. |
| `strip_attributes` | Logical; should semantic metadata attributes (such as `label`, `unit`, `definition`, and `namespace`) be removed from the returned vector? Defaults to 'TRUE'. |

## Details

Use `strip_attributes = TRUE` when flattening or preparing data for external pipelines, but keep the default when working with defined vectors directly.

## Value

A factor vector.

## Examples

```
sex <- defined(
  c(0, 1, 1, 0),
  label  = "Sex",
  labels = c("Female" = 0, "Male" = 1)
)

as_factor(sex)
as_factor(sex, strip_attributes = FALSE)
```

as_logical *Coerce a defined vector to logical*

## Description

Coerces a haven_labelled_defined vector created with defined() into a base R logical vector.

This function is the recommended, semantic-aware interface for converting defined logical vectors. It preserves the underlying logical values and, unless otherwise requested, retains the semantic metadata attached to the variable.

## Usage

```
as_logical(x, ...)

## S3 method for class 'haven_labelled_defined'
as_logical(x, strip_attributes = TRUE, ...)
```

## Arguments

| | |
|---|---|
| x | A vector created with defined(). |
| ... | Additional arguments (currently unused). |
| strip_attributes | Logical; should semantic metadata attributes (such as label, unit, definition, and namespace) be removed from the returned vector? Defaults to FALSE. |

## Details

Use strip_attributes = TRUE when flattening or preparing data for external pipelines, but keep the default when working with defined vectors directly.

Users may also call base R's as.logical(), which will dispatch to this method automatically via S3. However, as_logical() is preferred because it makes the semantics explicit and supports the strip_attributes argument.

## Value

A logical vector. If strip_attributes = FALSE, any semantic metadata attached to x will be carried over to the returned vector.

## See Also

as_numeric(), as_character(), as.Date(), as.POSIXct(), defined()

## Examples

```
# Basic usage
flg <- defined(c(TRUE, FALSE, TRUE), label = "Flag")
as_logical(flg)

# Metadata preserved by default
attr(as_logical(flg), "label")

# Stripping metadata
as_logical(flg, strip_attributes = TRUE)

# Base R coercion also works (via S3 method dispatch)
as.logical(flg)
```

---

# as_numeric *Coerce a defined vector to numeric*

---

## Description

`as_numeric()` converts a defined() vector to a numeric vector. It validates that the underlying data are numeric, and optionally preserves or strips semantic metadata.

## Usage

```
as_numeric(x, ...)

## S3 method for class 'haven_labelled_defined'
as_numeric(x, strip_attributes = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `x` | A vector created with `defined()`. |
| `...` | Reserved for future use. |
| `strip_attributes` | Logical; whether to remove semantic metadata (`label`, `unit`, `concept`, `namespace`). Defaults to `TRUE`. |

## Details

Use `strip_attributes = TRUE` when flattening or preparing data for external pipelines, but keep the default when working with defined vectors directly.

`as.numeric()` drops all metadata and returns only the numeric values in a vector.

## Value

A numeric vector with or without preserved attributes.

## Examples

```
x <- defined(
  1:3,
  label = "Count",
  unit = "n",
  concept = "http://example.org/count",
  namespace = "http://example.org/ns"
)

as_numeric(x)
```

---

# as_tibble.dataset_df *Coerce a* dataset_df *to a tibble*

---

## Description

as_tibble() method for dataset_df objects.

Converts a dataset_df into a tibble. By default, additional metadata attached to the object is removed unless strip_attributes = FALSE is specified.

## Usage

```
as_tibble(x, ...)

## S3 method for class 'dataset_df'
as_tibble(x, ..., strip_attributes = TRUE, .name_repair = "check_unique")

as.tibble.dataset_df(
  x,
  ...,
  strip_attributes = TRUE,
  .name_repair = "check_unique"
)
```

## Arguments

| | |
|---|---|
| x | A dataset_df object. |
| ... | Additional arguments passed to as.data.frame.dataset_df(). |
| strip_attributes | Logical: should column-level semantic metadata be stripped? Default: TRUE. |
| .name_repair | Treatment of problematic column names:<br>• "minimal": No name checks.<br>• "unique": Enforce uniqueness.<br>• "check_unique": (default) require unique names.<br>• "universal": Make unique and syntactic.<br>• A function or purrr-style anonymous function, see rlang::as_function(). |

## Value

A [tibble][tibble::tibble()] containing the data (and optionally some attributes) of the dataset_df‘.

## Metadata handling

The `strip_attributes` argument controls which attributes of the `dataset_df` object are preserved. When `strip_attributes = TRUE` (the default), only column-level information is kept.

## About column names

The `dataset_df` class may internally use reserved names for indexing, identifiers, or metadata. These are never exposed in the resulting tibble. Column names in the output tibble may be repaired according to `.name_repair`.

## See Also

- `tibble::as_tibble()`
- `as.data.frame.dataset_df()`

## Examples

```
# Convert a dataset_df to a tibble
x <- dataset_df(orange_df)
as_tibble(x)

# Keep attributes
as_tibble(x, strip_attributes = FALSE)
```

---

# as_value_key *Coerce semantic mappings to canonical key-value form*

---

## Description

Convert semantic mapping carriers into a canonical named character vector representation.

Convert semantic mappings into a two-column relational form suitable for joins, inspection, and roundtrip conversion with `as_value_key()`.

## Usage

```
as_value_key(x)

invert_value_key(x)
```

## Arguments

| | |
|---|---|
| x | A semantic mapping carrier:<br>• named character vector;<br>• named list;<br>• two-column data frame or tibble. |

## Details

`as_value_key()` standardises lightweight semantic mappings supplied as:

- named vectors;
- named lists;
- two-column data frames or tibbles.

The resulting object is suitable for:

- `prelabel()`;
- semantic grouping;
- contextual reconstruction;
- relational projection workflows.

Named lists may contain one-to-many mappings:

```
list(
  conceptualisation = c(
    "D:/_package/alpha",
    "D:/_markdown/alpha-methodology"
  ),
  betaR = c(
    "D:/_packages/beta",
    "D:/_packages/prebeta"
  )
)
```

where multiple values share the same semantic key.

The function is intentionally lightweight. It does not:

- validate ontologies;
- enforce uniqueness;
- resolve semantic conflicts;
- construct graph objects;
- perform recursive inheritance logic.

## Value

A named character vector representing canonical semantic key-value mappings.

## Examples

```
# named vector
as_value_key(
  c(
    R = "functional_programming",
    png = "visualisation"
  )
)

# named list
as_value_key(
  list(
    R = "functional_programming",
    png = "visualisation"
  )
)

# tibble
mapping_tbl <- tibble::tibble(
  extension = c("R", "png"),
  activity = c(
    "functional_programming",
    "visualisation"
  )
)

as_value_key(mapping_tbl)
```

---

# bibrecord *Create a Modern Metadata Object Compatible with bibentry*

---

## Description

Constructs a `utils::bibentry()` object extended with Dublin Core and DataCite-compatible fields. This unified structure supports use with functions such as `dublincore()` and `datacite()`, and is the internal format for storing rich metadata with datasets.

## Usage

```
bibrecord(
  title,
  author,
  contributor = NULL,
  publisher = NULL,
  year = NULL,
  date = Sys.Date(),
  identifier = NULL,
  subject = NULL,
```

```
  ...
)
```

## Arguments

| | |
|---|---|
| `title` | A character string specifying the dataset title. |
| `author` | A `utils::person()` or list/vector of person objects. Mapped to `creator` in DataCite and DCMI. |
| `contributor` | Optional list or vector of `utils::person()` objects. Contributor roles are merged if duplicated. |
| `publisher` | A character string or `utils::person()` representing the publishing entity. |
| `year` | Publication year. Automatically derived from `date` if not provided explicitly. |
| `date` | A Date object or character string in ISO format. |
| `identifier` | A persistent identifier (e.g., DOI or URL). |
| `subject` | Optional keyword, tag, or controlled vocabulary term. |
| `...` | Additional fields such as `language`, `format`, `rights`, or `description`. |

## Value

An object of class `"bibrecord"` and `"bibentry"`, suitable for citation and embedding in metadata-aware structures such as `dataset_df()`.

## See Also

Learn more in the vignette: bibrecord

Other bibrecord functions: `as_datacite()`, `as_dublincore()`

## Examples

```
bibrecord(
  title = "Gross domestic product, volumes",
  author = person("Eurosat"),
  publisher = person("Eurostat"),
  identifier = "https://doi.org/10.2908/TEINA011",
  date = as.Date("2025-05-20")
)
```

`bind_defined_rows` *Bind strictly defined rows*

## Description

Add rows of dataset `y` to dataset `x`, validating all semantic metadata. Metadata (labels, units, concept definitions, namespaces) must match exactly. Additional dataset-level metadata such as title and creator can be overridden using `...`.

## Usage

```
bind_defined_rows(x, y, ..., strict = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | A `dataset_df` object. |
| `y` | A `dataset_df` object to bind to `x`. |
| `...` | Optional dataset-level attributes such as `title` or `creator()` to override. |
| `strict` | Logical. If `TRUE` (default), require full semantic compatibility, including rowid. |

## Details

This function combines two semantically enriched datasets created with `dataset_df()`. All variable-level attributes <U+2014> including labels, units, concept definitions, and namespaces <U+2014> must match. If `strict = TRUE` (the default), the row identifier namespace (used in the `rowid` column) must also match exactly.

If `strict = FALSE`, row identifiers from `y` may differ and will be ignored; the output will inherit `x`'s row identifier scheme.

## Value

A new `dataset_df` object with rows from `x` and `y`, combined semantically.

## Examples

```
A <- dataset_df(
  length = defined(c(10, 15),
    label = "Length",
    unit = "cm", namespace = "http://example.org"
  ),
  identifier = c(id = "http://example.org/dataset#"),
  dataset_bibentry = dublincore(
    title = "Dataset A",
    creator = person("Alice", "Smith")
  )
)
```

```
B <- dataset_df(
  length = defined(c(20, 25),
    label = "Length",
    unit = "cm", namespace = "http://example.org"
  ),
  identifier = c(id = "http://example.org/dataset#")
)

bind_defined_rows(A, B) # succeeds

C <- dataset_df(
  length = defined(c(30, 35),
    label = "Length",
    unit = "cm", namespace = "http://example.org"
  ),
  identifier = c(id = "http://another.org/dataset#")
)

## Not run:
bind_defined_rows(A, C, strict = TRUE) # fails: mismatched rowid

## End(Not run)

bind_defined_rows(A, C, strict = FALSE) # succeeds: rowid inherited
```

---

`c.haven_labelled_defined` *Combine defined vectors with metadata checks*

---

## Description

The `c()` method for `defined` vectors ensures that all semantic metadata (label, unit, concept, namespace, and value labels) match exactly. This prevents accidental loss or mixing of incompatible definitions during concatenation.

## Usage

```
## S3 method for class 'haven_labelled_defined'
c(...)
```

## Arguments

| | |
|---|---|
| `...` | One or more vectors created with `defined()`. |

## Details

All input vectors must:

- Have identical `label` attributes
- Have identical `unit`, `concept`, and `namespace`
- Have identical value labels (or none)

## Value

A single `defined` vector with concatenated values and retained metadata.

## See Also

`defined()`

## Examples

```
a <- defined(1:3, label = "Length", unit = "meter")
b <- defined(4:6, label = "Length", unit = "meter")
c(a, b)
```

---

`contributor` *Get or set contributors*

---

## Description

`contributor()` is a lightweight wrapper around `creator()` that works only with contributors. It retrieves or updates only the contributor entries in the dataset's bibliographic metadata.

## Usage

```
contributor(x)

contributor(x, overwrite = FALSE) <- value
```

## Arguments

| | |
|---|---|
| `x` | A dataset object created with `dataset_df()` or `as_dataset_df()`. |
| `overwrite` | Logical. If `TRUE`, replace all existing contributors with `value`. If `FALSE`, append `value` to the existing contributors. Defaults to `FALSE`. |
| `value` | A `utils::person()` object representing a single contributor. If the `role` field is missing, it will be set to `"ctb"`. If `NULL`, the dataset is returned unchanged. |

## Details

All people are stored in the `author` slot of the underlying `utils::bibentry()`. This helper preserves primary creators and filters or updates only those entries that represent contributors.

A *contributor* is defined as:

- a person with `role == "ctb"`, **or**
- a person with a `comment[["contributorType"]]`.
  Primary creators (authors) typically have `role %in% c("aut", "cre")`.
  Contributors can be further annotated with metadata in `comment`, for example:

```
comment = c(contributorType = "hostingInstitution",        ORCID = "0000-0000-0000-0000")
```

## Value

- `contributor()` returns a `utils::person()` or a list of such objects corresponding to contributors.
- `contributor<-()` returns the updated dataset (invisibly).

## See Also

Other bibliographic helper functions: `creator()`, `dataset_format()`, `dataset_title()`, `description()`, `geolocation()`, `get_bibentry()`, `language()`, `publication_year()`, `publisher()`, `relation()`, `rights()`, `subject()`

## Examples

```
df <- dataset_df(data.frame(x = 1))
creator(df) <- person("Jane", "Doe", role = "aut")

# Add a contributor
contributor(df, overwrite = FALSE) <-
  person("GitHub",
    role = "ctb",
    comment = c(contributorType = "hostingInstitution")
  )

# Replace all contributors
contributor(df) <- person("Support", "Team", role = "ctb")

# Inspect only contributors
contributor(df)
```

---

# creator *Get/set the Creator of the object.*

---

## Description

Add the optional `Creator` property as an attribute to a dataset object.

## Usage

```
creator(x)

creator(x, overwrite = TRUE) <- value
```

## Arguments

| | |
|---|---|
| `x` | A semantically rich data frame object created by `dataset_df()` or `dataset::\link{as_dataset_df}`. |

| | |
|---|---|
| overwrite | If the attributes should be overwritten. In case it is set to FALSE,it gives a message with the current Creator property instead of overwriting it. Defaults to TRUE when the attribute is set to value regardless of previous setting. |
| value | The Creator as a utils::person() object. |

## Details

The Creator corresponds to dct:creator in Dublin Core and Creator in DataCite. The name of the entity that holds, archives, publishes prints, distributes, releases, issues, or produces the dataset. This property will be used to formulate the citation, so consider the prominence of the role.

## Value

The Creator attribute as a character of length one is added to x.

## See Also

Other bibliographic helper functions: contributor(), dataset_format(), dataset_title(), description(), geolocation(), get_bibentry(), language(), publication_year(), publisher(), relation(), rights(), subject()

## Examples

```
creator(orange_df)
# To change author:
creator(orange_df) <- person("Jane", "Doe")
# To add author:
creator(orange_df, overwrite = FALSE) <- person("John", "Doe")
```

---

# dataset_df *Create a new* dataset_df *object*

---

## Description

The dataset_df() constructor creates semantically rich modern data frames. These inherit from tbl_df and carry structured metadata using attributes.

## Usage

```
dataset_df(
  ...,
  identifier = c(obs = "http://example.com/dataset#obs"),
  var_labels = NULL,
  units = NULL,
  concepts = NULL,
  dataset_bibentry = NULL,
  dataset_subject = NULL
)
```

```
as_dataset_df(
  df,
  identifier = c(obs = "http://example.com/dataset#obs"),
  var_labels = NULL,
  units = NULL,
  concepts = NULL,
  dataset_bibentry = NULL,
  dataset_subject = NULL,
  ...
)

is.dataset_df(x)

## S3 method for class 'dataset_df'
print(x, ...)

is_dataset_df(x)

## S3 method for class 'dataset_df'
names(x)
```

## Arguments

| | |
|---|---|
| `...` | Vectors (columns) that should be included in the dataset. |
| `identifier` | A named vector of one or more URI prefixes for row IDs. Defaults to `c(eg = "http://example.com/dataset#")`. For example, if your dataset will be published under DOI `https://doi.org/1234`, you may use `c(obs = "https://doi.org/1234#")`, which will generate row URIs such as `https://doi.org/1234#1`, ..., `#n`. |
| `var_labels` | A named list of human-readable labels for each variable. |
| `units` | A named list of measurement units for measured variables. |
| `concepts` | A named list of linked concepts (URIs) for variables or dimensions. |
| `dataset_bibentry` | A bibliographic metadata record for the dataset, created using `datacite()` or `dublincore()`. |
| `dataset_subject` | A subject descriptor created with `subject()` or `subject_create()`. |
| `df` | A `data.frame` to convert to a `dataset_df`. |
| `x` | A `dataset_df` object (used in method dispatch). |

## Details

Use `is.dataset_df()` to check class membership.

S3 methods for `dataset_df` include:

- `print()` to display the dataset with metadata
- `summary()` to summarize both data and metadata

- `names()` to retrieve variable names using standard data frame semantics

For full details, see `vignette("dataset_df", package = "dataset")`.

## Value

A `dataset_df` object: a tibble with attached metadata stored in attributes.

`is.dataset_df` returns a logical value (if the object is of class `dataset_df`.)

## Note

A simple, serverless scaffolding for publishing `dataset_df` objects on the web (with HTML + RDF exports) is available at https://github.com/dataobservatory-eu/dataset-template.

## See Also

`defined()`, `dublincore()`, `datacite()`, `subject()`

## Examples

```
my_dataset <- dataset_df(
  country_name = defined(
    c("AD", "LI"),
    concept = "http://data.europa.eu/bna/c_6c2bb82d",
    namespace = "https://www.geonames.org/countries/$1/"
  ),
  gdp = defined(
    c(3897, 7365),
    label = "Gross Domestic Product",
    unit = "million dollars",
    concept = "http://data.europa.eu/83i/aa/GDP"
  ),
  identifier = c(
    obs = "https://dataobservatory-eu.github.io/dataset-template#"
  ),
  dataset_bibentry = dublincore(
    title = "GDP of Andorra and Liechtenstein",
    description = "A small but semantically rich dataset example.",
    creator = person("Jane", "Doe", role = "cre"),
    publisher = "Open Data Institute",
    language = "en"
  )
)

# Basic usage
print(my_dataset)
head(my_dataset)
summary(my_dataset)

# Metadata access
as_dublincore(my_dataset)
as_datacite(my_dataset)
```

```
# Export description as RDF triples
my_description <- describe(my_dataset, con = tempfile())
my_description
```

---

`dataset_format` *Get or set the technical format of a dataset*

---

## Description

Adds or retrieves the optional `"format"` field of a dataset's bibentry. This field is the dataset's technical/media type (e.g., a MIME type).

## Usage

```
dataset_format(x)

dataset_format(x, overwrite = FALSE) <- value
```

## Arguments

| | |
|---|---|
| `x` | A semantically rich data frame created with `dataset_df()` or `as_dataset_df()`. |
| `overwrite` | Logical. Replace an existing non<U+2011>default value? If `FALSE` and a non<U+2011>default value already exists, a message is emitted and the value is kept. Defaults to `FALSE`. |
| `value` | A length<U+2011>one character string specifying the format (e.g., `"text/csv"`). Use `NULL` to reset to the default. |

## Details

The format field corresponds to dct:format in Dublin Core and to `format` in DataCite. It is useful for indicating serialization such as `"text/csv"`, `"application/parquet"`, or `"application/r-rds"`.

If no format is set, this helper uses the package default `"application/r-rds"`.

## Value

The `"format"` (technical format) as a character string (length 1). When assigning, the updated object `x` is returned invisibly.

## See Also

Other bibliographic helper functions: `contributor()`, `creator()`, `dataset_title()`, `description()`, `geolocation()`, `get_bibentry()`, `language()`, `publication_year()`, `publisher()`, `relation()`, `rights()`, `subject()`

### Examples

```
dataset_format(orange_df) <- "text/csv"
dataset_format(orange_df)

# Reset to the package default
dataset_format(orange_df) <- NULL
```

---

## dataset_title *Get or Set the Title of a Dataset*

---

### Description

Retrieve or assign the main title of a dataset, typically used as the primary label in metadata exports (e.g., DataCite or Dublin Core).

### Usage

```
dataset_title(x)

dataset_title(x, overwrite = FALSE) <- value
```

### Arguments

| | |
|---|---|
| x | A dataset object created by dataset_df() or as_dataset_df(). |
| overwrite | Logical. If TRUE, the existing title is replaced. If FALSE (default) and a title is already present, a warning is issued and the title is not changed. |
| value | A character string representing the new title. If NULL, a placeholder value ": tba" is assigned. If value is a character vector of length > 1, an error is raised. |

### Details

According to the [Dublin Core specification for title, the title represents the name by which the resource is formally known.

The DataCite metadata schema supports multiple titles (e.g., translated, alternative), but this function currently supports only a single main title.

### Value

dataset_title() returns the current dataset title as a character string. dataset_title<-() returns the updated dataset object (invisible).

### See Also

Other bibliographic helper functions: contributor(), creator(), dataset_format(), description(), geolocation(), get_bibentry(), language(), publication_year(), publisher(), relation(), rights(), subject()

## Examples

```
dataset_title(orange_df)

# Set a new title with overwrite = TRUE
dataset_title(orange_df, overwrite = TRUE) <- "The Growth of Orange Trees"
dataset_title(orange_df)
```

---

`dataset_to_triples` *Dataset to triples (three columns or N-Triples)*

---

## Description

Converts a dataset to RDF-style triples with subject, predicate, and object columns. Supports semantic expansion via variable metadata.

## Usage

```
dataset_to_triples(x, idcol = NULL, expand_uri = TRUE, format = "data.frame")
```

## Arguments

| | |
|---|---|
| `x` | A `dataset_df` or `data.frame`. |
| `idcol` | Name or index of the subject column. If NULL, defaults to `"rowid"` or rownames. |
| `expand_uri` | Logical; if TRUE, expands URIs using namespaces and definitions. |
| `format` | Output format: `"data.frame"` (default) or `"nt"` for N-Triples. |

## Details

For publishing examples, a minimal serverless scaffold is provided at https://github.com/dataobservatory-eu/dataset-template, which shows how to host CSV + RDF serialisations on GitHub Pages without any server setup.

## Value

Either a `data.frame` with columns `s`, `p`, and `o`, or a character vector of N-Triple lines.

## Note

A simple, serverless scaffolding for publishing `dataset_df` objects on the web (with HTML + RDF exports) is available at https://github.com/dataobservatory-eu/dataset-template.

## Examples

```
# A minimal example with just rowid and geo
data("gdp", package = "dataset")
small_geo <- dataset_df(
  geo = defined(
    gdp$geo[1:3],
    label = "Geopolitical entity",
    concept = "http://example.com/prop/geo",
    namespace = "https://dd.eionet.europa.eu/vocabulary/eurostat/geo/$1"
  )
)

# View as triple table
dataset_to_triples(small_geo)

# View as N-Triples
dataset_to_triples(small_geo, format = "nt")
```

---

`defined` *Create a semantically enriched vector with variable-level metadata*

---

## Description

`defined()` constructs a vector that behaves like a base R vector but carries semantic metadata used for documentation, validation, and interoperability. The resulting object inherits from `haven::labelled()` (for numeric, character, and factor data) or from base date/time classes, and adds:

## Usage

```
defined(
  x,
  labels = NULL,
  label = NULL,
  unit = NULL,
  concept = NULL,
  namespace = NULL,
  ...
)

is.defined(x)

## S3 method for class 'haven_labelled_defined'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| `x` | A vector to annotate. |

| | |
|---|---|
| labels | Optional named vector of value labels. Only supported for numeric or character vectors (not for logical). |
| label | A short human-readable variable label (character of length 1). |
| unit | Unit of measurement (character length 1) or NULL. |
| concept | A URI or identifier describing the meaning or definition of the variable. This replaces the deprecated definition argument. |
| namespace | Optional string or named character vector used to generate value-level URIs via substitution ($1 macro). |
| ... | For backward compatibility; the deprecated definition argument is still accepted and mapped to concept. |
| object | An R object to be summarised. |

## Details

- a human-readable variable label,
- an optional unit of measurement,
- a **concept URI** identifying the meaning of the variable (formerly called *definition*),
- an optional namespace used for value-level URI expansion,
- optional labelled values (where supported).

The concept attribute is a general semantic anchor and may refer to: a measure or dimension concept (SDMX-style), a property IRI (e.g. Wikibase), or any URI that defines or describes the variable.

defined() vectors preserve their metadata during subsetting, printing, summarizing, comparisons, and many tidyverse operations. They integrate smoothly with dataset_df() objects and can be safely flattened via as.data.frame(), as_tibble(), or coercion helpers such as as_numeric() and as_character().

## Value

A vector of class haven_labelled_defined or datetime_defined, depending on the input type.

## Supported input types

- numeric (integer or double)
- character
- factor (converted via labelled::to_labelled())
- Date
- POSIXct
- haven::labelled()
- logical (with restrictions: logical vectors **cannot** have value labels)

## See Also

is.defined(), as_numeric(), as_character(), as_logical(), strip_defined(), dataset_df().

## Examples

```
gdp_vector <- defined(
  c(3897, 7365, 6753),
  label = "Gross Domestic Product",
  unit = "million dollars",
  concept = "http://data.europa.eu/83i/aa/GDP"
)

is.defined(gdp_vector)
print(gdp_vector)
summary(gdp_vector)
gdp_vector[1:2]
```

---

`describe` *Describe a dataset in N-Triples format*

---

## Description

Writes provenance and Dublin Core metadata of a dataset to a file or connection in N-Triples format.

## Usage

```
describe(x, con)
```

## Arguments

| | |
|---|---|
| `x` | A `dataset_df` object. |
| `con` | A connection or a character string path (e.g. from `tempfile()`). |

## Value

Writes N-Triples to `con` and invisibly returns `x`.

## Examples

```
test_ds <- dataset_df(
  rowid = defined(c("eg:1", "eg:2"),
    namespace = "http://example.com/dataset#"
  ),
  geo = defined(
    gdp$geo[1:2],
    label = "Country",
    concept = "http://example.com/prop/geo",
    namespace = "https://eionet.europa.eu/geo/$1"
  ),
  dataset_bibentry = dublincore(
    title = "Example Dataset",
    creator = person("John", "Doe")
```

```
  )
)

# returns invisibly the contents of the text file serialisation:
testdescription <- describe(test_ds, con = tempfile())
testdescription
```

---

`description` *Get or set the dataset Description*

---

## Description

Get or set the optional `Description` property as an attribute on a dataset object.

## Usage

```
description(x)

description(x, overwrite = FALSE) <- value
```

## Arguments

| | |
|---|---|
| `x` | A dataset object created with `dataset_df()` or `as_dataset_df()`. |
| `overwrite` | Logical. If `TRUE`, will overwrite any existing description. If `FALSE` (default), will warn and keep the existing description. |
| `value` | The new description, as a character string. |

## Details

The `Description` is recommended for discovery in DataCite. It captures additional information that does not fit other metadata categories <U+2014> such as technical notes or dataset usage. It is a free-text field. See dct:description.

## Value

The `Description` attribute as a character vector of length 1.

## See Also

Other bibliographic helper functions: `contributor()`, `creator()`, `dataset_format()`, `dataset_title()`, `geolocation()`, `get_bibentry()`, `language()`, `publication_year()`, `publisher()`, `relation()`, `rights()`, `subject()`

## Examples

```
description(orange_df)
description(orange_df, overwrite = TRUE) <- "This dataset records orange tree growth."
description(orange_df)
```

gdp *A Small GDP Dataset*

## Description

A compact sample of GDP and main aggregates from Eurostat's annual international cooperation dataset. This data subset contains illustrative records for select countries and time periods.

## Usage

```
gdp
```

## Format

A data frame with 10 rows and 5 variables:

- `geo`: Country name (character)
- `year`: Reference year (integer)
- `gdp`: Gross Domestic Product value (numeric)
- `unit`: Unit of measurement, e.g., "Million EUR" (character)
- `freq`: Observation frequency, e.g., "Annual" (character)

## Details

This dataset is intended for examples, tests, and demonstration purposes. It reflects simplified GDP data as published by Eurostat. The actual Eurostat dataset includes more countries, breakdowns, and metadata.

## Source

Eurostat (2021). GDP and main aggregates - international data cooperation (annual data). doi:10.2908/NAIDA_10_GDP

## Examples

```
head(gdp)
```

`geolocation` *Get or Set the Geolocation of a Dataset Object*

## Description

Access or assign the optional `geolocation` attribute to a semantically rich dataset object.

## Usage

```
geolocation(x)

geolocation(x, overwrite = TRUE) <- value
```

## Arguments

| | |
|---|---|
| `x` | A dataset object created by `dataset_df()` or `dataset::as_dataset_df()`. |
| `overwrite` | Logical. If `TRUE` (default), the existing `geolocation` attribute is replaced with `value`. If `FALSE`, the function returns a message and does not overwrite the existing value. |
| `value` | A character string specifying the `geolocation`. |

## Details

The `geolocation` field describes the spatial region or named place where the data was collected or that the dataset is about. This field is recommended for data discovery in DataCite Metadata Schema 4.4.

See: DataCite: Geolocation Guidance

## Value

A character string of length 1, representing the `geolocation` attribute attached to `x`.

## See Also

Other bibliographic helper functions: `contributor()`, `creator()`, `dataset_format()`, `dataset_title()`, `description()`, `get_bibentry()`, `language()`, `publication_year()`, `publisher()`, `relation()`, `rights()`, `subject()`

## Examples

```
orange_dataset <- orange_df
geolocation(orange_df) <- "US"
geolocation(orange_df)

geolocation(orange_df, overwrite = FALSE) <- "GB"
```

get_bibentry *Get or set the bibentry*

## Description

Retrieve or replace the bibliographic entry stored in a dataset's attributes. The entry is a `utils::bibentry()` used to hold citation metadata for `dataset_df()` objects.

## Usage

```
get_bibentry(dataset)

set_bibentry(dataset) <- value
```

## Arguments

| | |
|---|---|
| dataset | A dataset created with `dataset_df()`. |
| value | A `utils::bibentry()` to store on the dataset. If `NULL`, a minimal default entry is created. |

## Details

New datasets are initialized with reasonable defaults. To build a new bibentry with sensible defaults and field names, use `datacite()` (DataCite) or `dublincore()` (Dublin Core), then assign it with `set_bibentry(dataset) <- value`.

See the vignette for more background: `vignette("bibentry", package = "dataset")`.

## Value

- `get_bibentry(dataset)` returns the `utils::bibentry()` stored in `dataset`'s attributes.
- `set_bibentry(dataset) <- value` sets the attribute and returns the modified dataset invisibly.

## See Also

Other bibliographic helper functions: `contributor()`, `creator()`, `dataset_format()`, `dataset_title()`, `description()`, `geolocation()`, `language()`, `publication_year()`, `publisher()`, `relation()`, `rights()`, `subject()`

## Examples

```
# Get the bibentry of a dataset_df object:
be <- get_bibentry(orange_df)

# Create a well-formed bibentry (DataCite-style):
be2 <- datacite(
  Creator = person("Jane", "Doe"),
```

```
  Title = "The Orange Trees Dataset",
  Publisher = "MyOrg"
)

# Assign the new bibentry:
set_bibentry(orange_df) <- be2

# Inspect in different notations:
as_datacite(orange_df, type = "list")
as_dublincore(orange_df, type = "list")
```

---

## get_variable_concepts *Get concepts for all variables in a dataset_df*

---

### Description

Returns a named list of concept URIs (or NULLs) for all variables.

### Usage

```
get_variable_concepts(x)
```

### Arguments

| | |
|---|---|
| x | A dataset_df object. |

### Value

A named list of concept URIs for each variable.

### Examples

```
get_variable_concepts(orange_df)
```

---

## identifier *Get or Set the Identifier of a Dataset or Metadata Record*

---

### Description

Retrieve or assign the identifier attribute of a dataset or bibliographic metadata object.

### Usage

```
identifier(x)

identifier(x, overwrite = TRUE) <- value
```

## Arguments

| | |
|---|---|
| x | A dataset_df() object or a utils::bibentry() object (including dublincore() or datacite() records). |
| overwrite | Logical. If TRUE (default), any existing identifier is replaced. If FALSE, an existing identifier is preserved unless it is ":unas" or ":tba". |
| value | A character string giving the identifier. Can be named (e.g., c(doi = "...")) or unnamed. Numeric values are coerced to character. |

## Details

An *identifier* provides an unambiguous reference to a resource. Recommended practice is to supply a persistent identifier string, such as a DOI, ISBN, or URN, that conforms to a recognized identification system.

Both Dublin Core and DataCite 4.4 define identifier as a core property. If the identifier is a DOI, it will also be stored in the doi field of the metadata record.

Although identifier is not part of the minimal Dublin Core term set, it is always included in dataset metadata for compatibility with publishing and indexing systems. You may omit it if working under a strict DC profile.

For best practice in choosing identifier schemes, see the IANA-registered URI schemes.

## Value

For identifier(), the current identifier as a character string. For identifier<-(), the updated object (invisible).

## Examples

```
orange_copy <- orange_df

# Get the current identifier
identifier(orange_copy)

# Set a new identifier (e.g., a DOI)
identifier(orange_copy) <- "https://doi.org/10.9999/example.doi"

# Prevent accidental overwrite
identifier(orange_copy, overwrite = FALSE) <- "https://example.org/id"

# Use numeric and NULL values
identifier(orange_copy) <- 12345
identifier(orange_copy) <- NULL # Sets ":unas"
```

## id_to_column *Add Identifier to First Column of a Dataset*

### Description

Adds a prefixed identifier (e.g., `eg:`) to the first column of a dataset, useful for generating semantic row IDs (e.g., for RDF serialization).

### Usage

```
id_to_column(x, prefix = "eg:", ids = NULL)
```

### Arguments

| | |
|---|---|
| `x` | A dataset created with `dataset_df()`, or a regular data frame. |
| `prefix` | A character string used as the prefix for row identifiers. Defaults to `"eg:"` (referring to example.com). |
| `ids` | Optional. A character vector of custom IDs to use instead of row names. |

### Value

A dataset of the same class as `x`, with the first column updated to include unique prefixed identifiers.

### Examples

```
# Example with a dataset_df object:
id_to_column(orange_df)

# Example with a regular data.frame:
id_to_column(Orange, prefix = "orange:")
```

## language *Set the Primary Language of a Dataset*

### Description

Assign the primary language of a semantically rich dataset object using an ISO 639 language code or full language name. This sets the `language` attribute in the dataset's metadata.

### Usage

```
language(x)

language(x, iso_639_code = "639-3") <- value

language(x, iso_639_code = "639-3") <- value
```

## Arguments

| | |
|---|---|
| x | A dataset object created by dataset_df() or as_dataset_df(). |
| iso_639_code | A character string indicating the desired return format: either "639-3" (default; terminologic) or "639-1" (2-letter code). |
| value | A 2-letter or 3-letter language code (ISO 639-1 or ISO 639-2), or a full language name (case-insensitive). |

## Details

This function supports recognition of:

- 2-letter codes (ISO 639-1, e.g., "en", "fr")
- 3-letter codes from both:
  - Alpha_3_B (bibliographic, e.g., "fre")
  - Alpha_3_T (terminologic, e.g., "fra")
- Full language names (e.g., "English", "French")

For compatibility with open science repositories and modern metadata standards, this function **returns the terminologic code** (Alpha_3_T) when available. If Alpha_3_T is missing for a language, the legacy bibliographic code (Alpha_3_B) is used as a fallback.

Full language names (e.g., "English", "Spanish") are matched case-insensitively against the ISO 639-2 Name field. Exact matches are attempted first; if none are found, a prefix match is used. For example:

- "English" returns "eng"
- "English, Old" returns "ang"

This means that:

- Both "fra" (terminologic) and "fre" (bibliographic) will be accepted as valid input for French
- The resulting value stored and returned will be "fra"

This behaviour aligns with:

- DataCite Metadata Schema 4.4
- schema.org
- Common repository practices (Zenodo, OSF, Figshare)

If value is NULL, the language is marked as ":unas" (unspecified).

In some cases<U+2014>especially for historical or moribund languages<U+2014>multiple similar names may exist. In such cases, it is safer to use a specific language code (e.g., "ang" instead of "English, Old" and "enm" for "English, Middle (1100-1500)"). You can also refer directly to the definitions in ISOcodes::ISO_639_2 for clarity.

## Value

The dataset with an updated language attribute, typically an ISO 639-2/T code (Alpha_3_T) such as "fra", "eng", "spa", etc.

## See Also

Other bibliographic helper functions: contributor(), creator(), dataset_format(), dataset_title(), description(), geolocation(), get_bibentry(), publication_year(), publisher(), relation(), rights(), subject()

## Examples

```
df <- dataset_df(data.frame(x = 1:3))

language(df) <- "English" # Returns "eng"
language(df) <- "fre" # Legacy code; returns "fra"
language(df) <- "fra" # Returns "fra"
language(df, iso_639_code = "639-1") <- "fra" # Returns "fr"

language(df) <- NULL # Sets ":unas"
```

---

# n_triple *Create an N-Triple*

---

## Description

Create a single N-Triple triple.

## Usage

```
n_triple(s, p, o)
```

## Arguments

| | |
|---|---|
| s | The subject of a triplet. |
| p | The predicate of a triplet. |
| o | The object of a triplet. |

## Details

N-Triples is an easy to parse line-based subset of Turtle to serialize RDF. An N-Triple triple is a sequence of RDF terms representing the subject, predicate and object of an RDF Triple. Use n_triples() to serialize multiple statements.

## Value

A character vector containing one N-Triple string.

## Source

RDF 1.1 N-Triples

### Examples

```
s <- "http://example.org/show/218"
p <- "http://www.w3.org/2000/01/rdf-schema#label"
o <- "That Seventies Show"
n_triple(s, p, o)
```

---

`n_triples` *Create N-Triples*

---

### Description

Create RDF triple statements to annotate your dataset with standard, interoperable metadata.

### Usage

```
n_triples(triples)
```

### Arguments

| | |
|---|---|
| `triples` | A character vector of concatenated N-Triples, created with `n_triple()`. |

### Details

N-Triples is a line-based serialization format for RDF. It is easy to parse and widely supported. For details, see the W3C RDF 1.2 N-Triples specification.

### Value

A character vector of unique N-Triple strings.

### Examples

```
triple_1 <- n_triple(
  "http://example.org/show/218",
  "http://www.w3.org/2000/01/rdf-schema#label",
  "That Seventies Show"
)

triple_2 <- n_triple(
  "http://example.org/show/218",
  "http://example.org/show/localName",
  '"Cette S<U+00E9>rie des Ann<U+00E9>es Septante"@fr-be'
)

n_triples(c(triple_1, triple_2, triple_1))
```

orange_df *Growth of Orange Trees*

## Description

A dataset recording the growth of orange trees, replicated from the classic datasets::Orange dataset and implemented as a `dataset_df` S3 class with enhanced semantic metadata.

## Usage

```
orange_df
```

## Format

A data frame with 35 rows and 4 variables:

- `rowid`: A unique identifier for each row (character)
- `tree`: Tree identifier (ordered factor)
- `age`: Age of the tree in days (numeric)
- `circumference`: Trunk circumference in mm (numeric)

## Details

This is a semantically enriched version of the classic Orange dataset, constructed using the `dataset_df()` and `dublincore()` constructors. Each column includes semantic metadata such as units, labels, concepts, or namespace identifiers. The dataset also embeds a machine-readable citation for reproducibility and provenance tracking.

**Constructor Example:**

```
orange_bibentry <- dublincore(
  title = "Growth of Orange Trees",
  creator = c(
    person(
      given = "N.R.",
      family = "Draper",
      role = "cre",
      comment = c(VIAF = "http://viaf.org/viaf/84585260")
    ),
    person(
      given = "H",
      family = "Smith",
      role = "cre"
    )
  ),
  contributor = person(
    given = "Antal",
```

```
    family = "Daniel",
    role = "dtm"
  ),
  publisher = "Wiley",
  datasource = "https://isbnsearch.org/isbn/9780471170822",
  dataset_date = 1998,
  identifier = "https://doi.org/10.5281/zenodo.14917851",
  language = "en",
  description = "The Orange data frame has 35 rows and 3 columns of records of the growth of orange trees
)

orange_df <- dataset_df(
  rowid = defined(paste0("orange:", row.names(Orange)),
    label = "ID in the Orange dataset",
    namespace = c("orange" = "datasets::Orange")
  ),
  tree = defined(Orange$Tree,
    label = "The number of the tree"
  ),
  age = defined(Orange$age,
    label = "The age of the tree",
    unit = "days since 1968/12/31"
  ),
  circumference = defined(Orange$circumference,
    label = "circumference at breast height",
    unit = "milimeter",
    concept = "https://www.wikidata.org/wiki/Property:P2043"
  ),
  dataset_bibentry = orange_bibentry
)

orange_df$rowid <- defined(orange_df$rowid,
  namespace = "https://doi.org/10.5281/zenodo.14917851"
)
```

## References


- Draper, N. R. & Smith, H. (1998). *Applied Regression Analysis* (3rd ed.). Wiley.
- Pinheiro, J. C. & Bates, D. M. (2000). *Mixed-effects Models in S and S-PLUS*. Springer.
- Becker, R. A., Chambers, J. M. & Wilks, A. R. (1988). *The New S Language*. Wadsworth & Brooks/Cole.


## Examples

```
# Print with semantic citation and data preview
print(orange_df)

# Access semantic metadata associated with variables
print(orange_df$age)
```

```
# Retrieve the embedded bibliographic record
as_dublincore(orange_df)
```

---

`prelabel` *Add lightweight semantic mappings to a vector*

---

## Description

`prelabel()` applies lightweight semantic mappings to a vector before formal definition with `defined()`.

## Usage

```
prelabel(x, labels, unmatched = "keep", missing_label = "<NA>")

is.prelabelled(x)
```

## Arguments

| | |
|---|---|
| `x` | A vector. |
| `labels` | Candidate semantic mappings describing provisional semantic assertions.<br>`labels` is internally normalised with `as_value_key()` and may therefore be supplied as:<br>• a named vector;<br>• a named list;<br>• a two-column data frame or tibble. |
| `unmatched` | Behaviour for unmatched observational values.<br>One of:<br>• `"keep"` (default): preserve unmatched values as self-describing semantic assertions;<br>• `"na"`: operationalise unmatched values as `NA` during semantic coercion. |
| `missing_label` | Semantic assertion used internally for missing observational values. |

## Details

The `prelabelled` class is intended for:

- provisional harmonisation;
- contextual grouping;
- lightweight classification;
- semantic preprocessing workflows.

Unlike `defined()`, `prelabel()` does not enforce formal semantic definitions, namespaces, or units.

Semantic mappings may be supplied as:

- named vectors;
- named lists;
- two-column data frames.

These mappings are normalised internally with `as_value_key()`.

`is.prelabelled()` tests if a vector inherits the `prelabelled` class.

`prelabelled` vectors intentionally separate:

- observational evidence;
- semantic operationalisation;
- contextual semantic refinement.

The original observational values remain unchanged while semantic assertions may evolve through iterative refinement workflows.

Semantic operationalisation is provided with:

- `as.character()` for lightweight semantic coercion;
- `as_character()` for provenance-preserving semantic workspaces.

## Value

A vector with:

- class `"prelabelled"`;
- attached provisional semantic vocabulary stored in the `"prelabel"` attribute.

## Examples

```
x <- c("R","png", "csv", "unknown")

extension_map <- c(
  R = "functional_programming",
  png = "visualisation",
  csv = "tabular_data"
)

x <- prelabel(x, labels = extension_map)

x

is.prelabelled(x)

as.character(x)

semantic_workspace <- as_character(x)

attributes(semantic_workspace)
```

`print.haven_labelled_defined` *Print a defined (haven_labelled_defined) vector*

## Description

Custom print method for haven_labelled_defined vectors created with `defined()`. It prints the variable name, label, and a short semantic summary before the underlying values.

## Usage

```
## S3 method for class 'haven_labelled_defined'
print(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `haven_labelled_defined` vector. |
| `...` | Passed on to `base::print()`. |

## Value

`x`, invisibly.

## See Also

`defined()`, `summary.haven_labelled_defined()`

## Examples

```
sex <- defined(
  c(0, 1, 1, 0),
  label  = "Sex",
  labels = c("Female" = 0, "Male" = 1)
)

print(sex)
```

provenance *Get or update provenance information*

## Description

Retrieve or append provenance statements (in N<U+2011>Triples form) stored on a `dataset_df()` object.

## Usage

```
provenance(x)

provenance(x) <- value
```

## Arguments

| | |
|---|---|
| `x` | A dataset created with `dataset_df()`. |
| `value` | Character vector of N<U+2011>Triples created by `n_triple()` or `n_triples()` to append to existing provenance. |

## Details

Provenance is stored in the `"prov"` attribute as N<U+2011>Triples text. Use `n_triple()` or `n_triples()` to construct valid statements that follow PROV<U+2011>O (e.g., `prov:wasGeneratedBy`, `prov:wasInformedBy`).

## Value

- `provenance(x)` returns the contents of the `"prov"` attribute (character vector of N<U+2011>Triples), or `NULL` if none is set.
- `provenance(x) <- value` appends `value` to the `"prov"` attribute and returns the modified dataset invisibly.

## Examples

```
provenance(orange_df)

# Add a provenance statement:
provenance(orange_df) <- n_triple(
  "https://doi.org/10.5281/zenodo.10396807",
  "http://www.w3.org/ns/prov#wasInformedBy",
  "http://example.com/source#1"
)
```

publication_year *Get or Set the Publication Year of a Dataset Object*

## Description

Access or assign the optional publication_year attribute to a semantically rich dataset object.

## Usage

```
publication_year(x)

publication_year(x, overwrite = TRUE) <- value
```

## Arguments

| | |
|---|---|
| x | A dataset object created by dataset_df() or dataset::as_dataset_df(). |
| overwrite | Logical. If TRUE (default), the existing publication_year attribute is replaced with value. If FALSE, the function returns a message and does not overwrite the existing value. |
| value | A character string specifying the publication year. |

## Details

The publication_year represents the year when the dataset was or will be made publicly available, in YYYY format. For additional context, see DataCite: Publication Year-Additional Guidance.

## Value

The publication_year attribute as a character string.

## See Also

Other bibliographic helper functions: contributor(), creator(), dataset_format(), dataset_title(), description(), geolocation(), get_bibentry(), language(), publisher(), relation(), rights(), subject()

## Examples

```
publication_year(orange_df)
publication_year(orange_df) <- "1998"
```

publisher *Get or Set the Publisher of a Dataset Object*

## Description

The publisher is the entity responsible for holding, archiving, releasing, or distributing the resource. It is typically included in dataset citation metadata.

For software, this might refer to a code repository (e.g., GitHub). If both a hosting platform and a producing institution are involved, use the publisher for the institution and `creator()` with `contributorType = "hostingInstitution"` for the platform.

## Usage

```
publisher(x)

publisher(x, overwrite = TRUE) <- value
```

## Arguments

| | |
|---|---|
| `x` | A dataset object created with `dataset_df()` or `as_dataset_df()`. |
| `overwrite` | Logical. Should existing publisher metadata be overwritten? Defaults to `FALSE`. If `FALSE` and the field exists, a warning is issued. |
| `value` | A character string specifying the publisher. |

## Details

Adds or retrieves the optional `"publisher"` attribute for a dataset object. This property aligns with `dct:publisher` (Dublin Core) and `publisher` (DataCite).

## Value

A character string of length one containing the `"publisher"` attribute. When assigning, the updated object `x` is returned invisibly.

## See Also

Other bibliographic helper functions: `contributor()`, `creator()`, `dataset_format()`, `dataset_title()`, `description()`, `geolocation()`, `get_bibentry()`, `language()`, `publication_year()`, `relation()`, `rights()`, `subject()`

## Examples

```
publisher(orange_df) <- "Wiley"
publisher(orange_df)
```

`relation` *Add or retrieve related items (DataCite/Dublin Core)*

## Description

Manage related resources for a dataset using a unified accessor.

- For **DataCite 4.x**, this maps to `relatedIdentifier` (+ type & relation).
- For **Dublin Core**, this maps to `dct:relation` (string).

## Usage

```
relation(x)

relation(x) <- value

related_create(
  relatedIdentifier,
  relationType,
  relatedIdentifierType,
  resourceTypeGeneral = NULL
)

is.related(x)

related_item(x)

related_item(x) <- value
```

## Arguments

| | |
|---|---|
| `x` | A dataset object created with `dataset_df()` or `as_dataset_df()`. |
| `value` | A `related` object from `related_create()` or a character. Vectors of characters are also supported and will be converted to a list of `"related"` objects. |
| `relatedIdentifier` | A string with the identifier of the related resource. |
| `relationType` | A string naming the relation type (per DataCite vocabulary). |
| `relatedIdentifierType` | A string naming the identifier type (`"DOI"`, `"URL"`, etc.). |
| `resourceTypeGeneral` | Optional: a string naming the general type of the related resource. |

## Details

To remain compatible with `utils::bibentry()`, the bibentry stores only the **string identifier** (e.g., DOI/URL). The full structured object created by `related_create()` is preserved in the `"relation"` attribute.

A `"related"` object is a small S3 list with the following elements:

- `relatedIdentifier`: the related resource identifier (DOI, URL, etc.)
- `relationType`: the DataCite relation type (e.g., `"IsPartOf"`, `"References"`)
- `relatedIdentifierType`: the type of identifier (`"DOI"`, `"URL"`, etc.)
- `resourceTypeGeneral`: optional, the general type of the related resource (e.g., `"Text"`, `"Dataset"`)

## Value

- `relation(x)` returns:
  - a single structured `"related"` object (from `related_create()`) if only one relation is present,
  - a list of `"related"` objects if multiple relations are present,
  - otherwise it falls back to the bibentry field (`relatedidentifier` for DataCite or `relation` for Dublin Core).
- `relation(x) <- value` sets the `"relation"` attribute (structured object or list of objects) and the bibentry string fields (`relatedidentifier` and `relation`), and returns the dataset invisibly.
- `related_create()` constructs a structured `"related"` object.
- `is.related(x)` returns `TRUE` if `x` inherits from class `"related"`.

## See Also

Other bibliographic helper functions: `contributor()`, `creator()`, `dataset_format()`, `dataset_title()`, `description()`, `geolocation()`, `get_bibentry()`, `language()`, `publication_year()`, `publisher()`, `rights()`, `subject()`

## Examples

```
df <- dataset_df(data.frame(x = 1))
relation(df) <- related_create(
  relatedIdentifier = "10.1234/example",
  relationType = "IsPartOf",
  relatedIdentifierType = "DOI"
)
relation(df) # structured object
get_bibentry(df)$relation # "10.1234/example"
get_bibentry(df)$relatedidentifier # "10.1234/example"

# Character input is normalized to a DOI/URL with default types
relation(df) <- "https://doi.org/10.5678/xyz"
relation(df) # structured object (relationType/Type filled with defaults)
```

```
# Create related object directly
rel <- related_create("https://doi.org/10.5678/xyz", "References", "DOI")
is.related(rel) # TRUE
```

---

`rights` *Get or Set the Rights of a Dataset Object*

---

## Description

Adds or retrieves the optional `"rights"` attribute of a dataset object. This field contains information about intellectual property or usage rights.

## Usage

```
rights(x)

rights(x, overwrite = FALSE) <- value
```

## Arguments

| | |
|---|---|
| `x` | A semantically rich data frame created with `dataset_df()` or `as_dataset_df()`. |
| `overwrite` | Logical. Should the existing value be replaced? If `FALSE` and a value already exists, the function emits a message instead of overwriting. Defaults to `FALSE`. |
| `value` | A character string specifying the rights (e.g., `"CC-BY-4.0"`). |

## Details

The `"rights"` field corresponds to dct:rights from Dublin Core, and to `rights` in DataCite.

Rights information typically includes statements about legal ownership, licensing, or usage conditions. It helps ensure that users understand how a dataset may be reused, cited, or shared.

## Value

The `"rights"` attribute of the dataset as a character string (length 1). When assigning, the updated object `x` is returned invisibly.

## See Also

Other bibliographic helper functions: `contributor()`, `creator()`, `dataset_format()`, `dataset_title()`, `description()`, `geolocation()`, `get_bibentry()`, `language()`, `publication_year()`, `publisher()`, `relation()`, `subject()`

## Examples

```
rights(orange_df) <- "CC-BY-SA"
rights(orange_df)
```

## strip_defined *Strip the class from a defined vector*

### Description

Converts a `defined` vector to a base R numeric or character, retaining metadata as passive attributes.

### Usage

```
strip_defined(x)
```

### Arguments

| | |
|---|---|
| `x` | A `defined` vector. |

### Value

A base R vector with attributes (`label`, `unit`, etc.) intact.

### See Also

`as_numeric()`, `as_character()`

### Examples

```
gdp <- defined(c(3897L, 7365L), label = "GDP", unit = "million dollars")
strip_defined(gdp)

fruits <- defined(c("apple", "avocado", "kiwi"),
  label = "Fruit", unit = "kg"
)

strip_defined(fruits)
```

## subject *Create, add, or retrieve a subject*

### Description

Manage the subject metadata of a dataset. The subject can be stored as a simple character term or as a structured object with subproperties created by `subject_create()`.

## Usage

```
subject(x)

subject_create(
  term,
  schemeURI = NULL,
  valueURI = NULL,
  prefix = NULL,
  subjectScheme = NULL,
  classificationCode = NULL
)

subject(x) <- value

is.subject(x)
```

## Arguments

| | |
|---|---|
| x | A dataset object created with dataset_df() or as_dataset_df(). |
| term | A subject term, for example "Data sets". |
| schemeURI | URI of the subject identifier scheme, for example "http://id.loc.gov/authorities/subjects". |
| valueURI | URI of the subject term, for example "https://id.loc.gov/authorities/subjects/sh2018002256". |
| prefix | Abbreviated prefix for a scheme URI, for example "lcch:". Widely used namespaces (schemes) have conventional abbreviations. |
| subjectScheme | Name of the subject scheme, classification code, or authority if one is used. This acts as a namespace. |
| classificationCode | Classification code for schemes that do not have valueURI entries for each subject term (e.g., ANZSRC). |
| value | A subject object created by subject_create() or a character string. Used by subject<- to replace the subject. |

## Details

The subject property records what the dataset is about. The DataCite subject property allows multiple subproperties, but these cannot be stored directly in a standard utils::bibentry() object. Therefore:

- If you set a character string as the subject, it is stored in both the bibentry and the "subject" attribute.
- If you set a structured subject (via subject_create()), the $term value is stored in the bibentry, and the full object is stored in the "subject" attribute of the dataset_df object.

## Value

- subject(x) returns:

  - a single "subject" object if only one is present,
  - a list of "subject" objects if multiple are present,
  - otherwise falls back to the plain string from the bibentry.
- subject(x) <- value accepts a character vector, a "subject" object, or a list of "subject" objects, and updates both the bibentry slot and the "subject" attribute. Returns the dataset invisibly.
- subject_create() returns a structured "subject" object <U+2014> or a list of them if multiple terms are provided.
- is.subject(x) returns TRUE if x inherits from class "subject".

## See Also

Other bibliographic helper functions: contributor(), creator(), dataset_format(), dataset_title(), description(), geolocation(), get_bibentry(), language(), publication_year(), publisher(), relation(), rights()

## Examples

```
# Set a structured subject
subject(orange_df) <- subject_create(
  term = "Oranges",
  schemeURI = "http://id.loc.gov/authorities/subjects",
  valueURI = "http://id.loc.gov/authorities/subjects/sh85095257",
  subjectScheme = "LCCH",
  prefix = "lcch:"
)

# Retrieve subject with subproperties
subject(orange_df)
```

---

# var_concept *Get / set a concept definition for a vector or a dataset*

---

## Description

Assigns a concept URI to a vector created with defined(). This method updates the concept attribute and validates that the input is a single character string or NULL.

## Usage

```
var_concept(x, ...)

var_concept(x) <- value

## Default S3 replacement method:
var_concept(x) <- value
```

### Arguments

| | |
|---|---|
| x | A vector to which the concept URI will be assigned. |
| ... | Further parameters for inheritance, not in use. |
| value | A character string with a concept URI or NULL to remove the concept. |

### Details

`get_variable_concepts()` is identical to `var_concept()`.

### Value

The (linked) concept of the meaning of the data contained by a vector constructed with`defined()`.

The modified vector with updated `concept` metadata.

### Examples

```
small_country_dataset <- dataset_df(
  country_name = defined(c("Andorra", "Lichtenstein"), label = "Country"),
  gdp = defined(c(3897, 7365),
    label = "Gross Domestic Product",
    unit = "million dollars"
  )
)
var_concept(small_country_dataset$country_name) <- "http://data.europa.eu/bna/c_6c2bb82d"
var_concept(small_country_dataset$country_name)
# To remove a concept definition of variable
var_concept(small_country_dataset$country_name) <- NULL
x <- defined(c(1, 2, 3), label = "Example Variable")
var_concept(x) <- "http://example.org/concept/XYZ"
var_concept(x)
```

---

## `var_label` *Get or Set a Variable Label*

---

### Description

Adds or retrieves a human-readable label as a metadata attribute for a variable or vector. This label is useful for making variables easier to understand than their programmatic names (e.g., column names).

`label_attribute()` is a low-level helper that retrieves the `"label"` attribute of an object without any fallback or printing logic. It is primarily used internally.

The `var_label<-` assignment method sets or removes the `"label"` attribute of a vector or data frame column. This allows attaching human-readable descriptions to variables for interpretability and downstream metadata use.

## Usage

```
## S3 method for class 'defined'
var_label(x, ...)

label_attribute(x)

var_label(x) <- value

## S3 replacement method for class 'haven_labelled_defined'
var_label(x) <- value

## S3 method for class 'dataset_df'
var_label(
  x,
  unlist = FALSE,
  null_action = c("keep", "fill", "skip", "na", "empty"),
  recurse = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| x | A vector or data frame. |
| ... | Further arguments passed to or used by methods. |
| value | A character string to assign as the label, or NULL to remove it. |
| unlist | For data frames, return a named vector instead of a list. |
| null_action | For data frames, controls how to handle columns without a variable label. Options are:<br>• "keep" (default): keep NULL for unlabeled columns<br>• "fill": use the column name as a fallback<br>• "skip": exclude columns with no label from the result<br>• "na": use NA_character_<br>• "empty": use an empty string "" |
| recurse | If TRUE, applies var_label() recursively on packed columns (as created by tidyr::pack()) to retrieve sub-column labels. If FALSE, only the outer (grouped) column label is returned. |

## Details

This interface builds on labelled::var_label() and is compatible with the defined() infrastructure for semantic metadata (labels, namespaces, units, and variable identifiers).

See labelled::var_label() for low-level usage. For a comprehensive guide to working with variable labels and semantic metadata, see: vignette("defined", package = "dataset").

## Value

- `var_label(x)` returns the `"label"` attribute of `x` as a character string.
- `var_label(x) <- value` sets, removes, or replaces the label attribute of `x`, returning the updated object invisibly.

A character string if the `"label"` attribute exists, or `NULL` if not present.

The modified object `x`, returned invisibly with the updated `"label"` attribute.

## See Also

`labelled::var_label()`, `var_labels()`, `defined()`

Other defined metadata methods and functions: `var_labels()`, `var_namespace()`, `var_unit()`

## Examples

```
# Retrieve the label attribute
var_label(orange_df$circumference)

# Set or update the label attribute
var_label(orange_df$circumference) <- "circumference (breast height)"

# Example: Retrieve variable labels from a dataset_df
df <- dataset_df(
  id = defined(1:3, label = "Observation ID"),
  temp = defined(c(22.5, 23.0, 21.8), label = "Temperature (<U+00B0>C)"),
  site = defined(c("A", "B", "A"))
)

# List form (default)
var_label(df)

# Character vector form
var_label(df, unlist = TRUE, null_action = "empty")

# Exclude variables without labels
var_label(df, null_action = "skip")

# Replace missing labels with column names
var_label(df, null_action = "fill")
```

---

# var_labels *Get or set all variable labels on a dataset*

---

## Description

Retrieve or assign labels for all variables (columns) in a dataset.

## Usage

```
var_labels(
  x,
  unlist = FALSE,
  null_action = c("keep", "fill", "skip", "na", "empty")
)

var_labels(x) <- value
```

## Arguments

| | |
|---|---|
| x | A data.frame or dataset_df object. |
| unlist | Logical; if TRUE, return a named character vector instead of a list. Defaults to FALSE. |
| null_action | How to handle columns without labels. One of:<br>• "keep" (default): keep NULL values for unlabeled columns.<br>• "fill": use the column name as a fallback label.<br>• "skip": exclude unlabeled columns from the result.<br>• "na": use NA_character_ for unlabeled columns.<br>• "empty": use an empty string "" for unlabeled columns. |
| value | • For setting: a named list or named character vector of labels. Names must match column names in x. Unnamed elements are ignored.<br>• For getting: ignored. |

## Details

This is the dataset-level equivalent of var_label(). It works with any data.frame-like object, including dataset_df(), and returns/sets the "label" attribute of each column.

Labels are useful for storing human-readable descriptions of variables that may have short or cryptic column names.

For internal purposes, this function uses the "var_labels" dataset attribute and delegates to var_label() and var_label<-() on individual columns.

## Value

- Getter: a named list (or vector if unlist = TRUE) of variable labels.
- Setter: the modified x with updated labels, returned invisibly.

## See Also

var_label()

Other defined metadata methods and functions: var_label, var_namespace(), var_unit()

## Examples

```
df <- dataset_df(
  id = defined(1:3, label = "Observation ID"),
  temp = defined(c(22.5, 23.0, 21.8), label = "Temperature (<U+00B0>C)"),
  site = defined(c("A", "B", "A"))
)

# Get all variable labels
var_labels(df)

# Set multiple labels at once
var_labels(df) <- list(site = "Site code")

# Return as a named vector with empty string for unlabeled vars
var_labels(df, unlist = TRUE, null_action = "empty")
```

---

# var_namespace *Get or Set the Namespace of a Variable*

---

## Description

Retrieve or assign the namespace part of a permanent, global variable identifier, independent of the current R session or instance.

## Usage

```
var_namespace(x, ...)

var_namespace(x) <- value

get_variable_namespaces(x, ...)

namespace_attribute(x)

get_namespace_attribute(x)

set_namespace_attribute(x, value)

namespace_attribute(x) <- value
```

## Arguments

| | |
|---|---|
| x | A vector. |
| ... | Additional arguments for method compatibility with other classes. |
| value | A character string specifying the namespace, or NULL to remove it. |

## Details

The namespace attribute is useful when working with remote, linked, or open data sources. Variable identifiers in such datasets are often qualified with a common namespace prefix. When combined, the prefix and namespace form a persistent URI or IRI for the variable.

Retaining the namespace ensures the identifiers remain valid and resolvable during validation, merging, or future updates of the vector (such as when it is used as a column in a dataset).

`get_variable_namespaces()` is an alias for `var_namespace()`. `namespace_attribute()` and `set_namespace_attribute()` are internal helpers.

For full usage, see: `vignette("defined", package = "dataset")` <U+2014> demonstrating integration of variable labels, namespaces, units of measure, and machine-independent identifiers.

## Value

A character string representing the namespace attribute of a vector constructed with `defined()`. Returns the updated object (in setter forms).

## See Also

Other defined metadata methods and functions: `var_label`, `var_labels()`, `var_unit()`

## Examples

```
# Define a vector with a namespace
x <- defined("Q42", namespace = c(wd = "https://www.wikidata.org/wiki/"))

# Get the namespace
var_namespace(x)
get_variable_namespaces(x)

# Set the namespace
var_namespace(x) <- "https://example.org/ns/"

# Remove the namespace
var_namespace(x) <- NULL

# Use lower-level helpers (not typically used directly)
namespace_attribute(x)
namespace_attribute(x) <- "https://example.org/custom/"
```

---

`var_unit` *Get or Set a Unit of Measure*

---

## Description

Adds or retrieves a unit of measure (UoM) attribute to a vector. Units provide semantic meaning for numeric or character data <U+2014> such as currency, weight, or time <U+2014> helping prevent incorrect operations like merging values measured in incompatible units.

The `var_unit<-` assignment method sets, updates, or removes the `"unit"` attribute of a vector. This can be used with `defined()` vectors or base vectors to ensure consistent semantic annotation.

`unit_attribute()` is a low-level helper to directly access the `"unit"` attribute of a vector, without applying fallback logic. It is mainly used internally.

`get_unit_attribute()` is an alias for `unit_attribute()`, included for naming consistency in codebases that distinguish getter/setter patterns.

`set_unit_attribute()` is the low-level assignment function that sets or removes the `"unit"` attribute of an object. Used internally by `unit_attribute<-`.

## Usage

```
var_unit(x, ...)

var_unit(x) <- value

## Default S3 replacement method:
var_unit(x) <- value

get_variable_units(x, ...)

unit_attribute(x)

get_unit_attribute(x)

set_unit_attribute(x, value)

unit_attribute(x) <- value
```

## Arguments

| | |
|---|---|
| `x` | A vector. |
| `...` | Further arguments for method extensions. |
| `value` | A single character string or `NULL`. If not of length one, an error is thrown. |

## Details

The `"unit"` attribute stores a machine-readable representation of a unit of measure (e.g., `"kg"`, `"USD"`, `"days"`). This is useful when working with linked open data or when combining data from multiple sources where silent mismatches in units could cause errors.

For full integration with semantic metadata (e.g., labels, concepts, namespaces), use `defined()` vectors or `dataset_df()` objects.

`get_variable_units()` is an alias for `var_unit()`.

See `vignette("defined", package = "dataset")` for end-to-end examples involving semantic enrichment.

## Value

- `var_unit(x)` returns the `"unit"` attribute as a character string.
- `var_unit(x) <- value` sets, updates, or removes the unit and returns the modified vector invisibly.

The modified object `x`, returned invisibly with the updated `"unit"` attribute.

The `"unit"` attribute of the object `x`, or `NULL` if not set.

The object `x` with updated `"unit"` attribute.

## See Also

Other defined metadata methods and functions: `var_label`, `var_labels()`, `var_namespace()`

## Examples

```
# Retrieve the unit of measure (if defined)
var_unit(orange_df$circumference)

# Regular data.frame columns have no unit by default
var_unit(mtcars$wt)

# Add a unit to a column
var_unit(mtcars$wt) <- "1000 lbs"

# Remove the unit
var_unit(mtcars$wt) <- NULL
```

---

# xsd_convert *Convert to XML Schema Definition (XSD) Types*

---

## Description

Converts R vectors, data frames, and `dataset_df` objects to XML Schema Definition (XSD) compatible string representations such as `xsd:decimal`, `xsd:boolean`, `xsd:date`, and `xsd:dateTime`.

## Usage

```
xsd_convert(x, idcol = NULL, shortform = TRUE, ...)

## S3 method for class 'haven_labelled_defined'
xsd_convert(x, idcol = NULL, shortform = TRUE, ...)

## S3 method for class 'data.frame'
```

```
xsd_convert(x, idcol = NULL, shortform = TRUE, ...)

## S3 method for class 'dataset_df'
xsd_convert(x, idcol = "rowid", shortform = TRUE, ...)

## S3 method for class 'tbl_df'
xsd_convert(x, idcol = NULL, shortform = TRUE, ...)

## S3 method for class 'character'
xsd_convert(x, idcol = NULL, shortform = TRUE, ...)

## S3 method for class 'numeric'
xsd_convert(x, idcol = NULL, shortform = TRUE, ...)

## S3 method for class 'integer'
xsd_convert(x, idcol = NULL, shortform = TRUE, ...)

## S3 method for class 'logical'
xsd_convert(x, idcol = NULL, shortform = TRUE, ...)

## S3 method for class 'factor'
xsd_convert(x, idcol = NULL, shortform = TRUE, ...)

## S3 method for class 'POSIXct'
xsd_convert(x, idcol = NULL, shortform = TRUE, ...)

## S3 method for class 'Date'
xsd_convert(x, idcol = NULL, shortform = TRUE, ...)

## S3 method for class 'difftime'
xsd_convert(x, idcol = NULL, shortform = TRUE, ...)
```

## Arguments

| | |
|---|---|
| x | An object (vector, data frame, tibble, or dataset_df). |
| idcol | Column name or position to use as row (observation) identifier. If NULL, row names are used. |
| shortform | Logical. If TRUE (default), datatypes are abbreviated with the xsd: prefix (e.g. "42"^^<xsd:integer>). If FALSE, datatypes are expanded to full URIs (e.g. "42"^^<http://www.w3.org/2001/XMLSchema#integer>). |
| ... | Additional arguments passed to methods. |

## Details

This is primarily used for generating RDF-compatible typed literals.

- For **vectors**, returns a character vector of typed literals.
- For **data frames** or tibbles, returns a data frame with the same structure but with all values converted to XSD strings.

- For `dataset_df` objects, behaves like the data frame method but preserves dataset-level attributes.

## Value

A character vector or data frame with values serialized as XSD-compatible RDF literals.

## Class-specific examples

```
xsd_convert(42L)                    # integer -> xsd:integer
xsd_convert(c(TRUE, FALSE, NA))     # logical -> xsd:boolean
xsd_convert(Sys.Date())             # Date -> xsd:date
xsd_convert(Sys.time())             # POSIXct -> xsd:dateTime
xsd_convert(factor("apple"))        # factor -> xsd:string
xsd_convert(c("apple", "banana"))  # character -> xsd:string
```

## Examples

```
# Simple data frame with mixed types
df <- data.frame(
  id     = 1:2,
  value  = c(3.14, 2.71),
  active = c(TRUE, FALSE),
  date   = as.Date(c("2020-01-01", "2020-12-31"))
)

# Short vs long-form URI:
xsd_convert(120L, shortform = TRUE)
xsd_convert(121L, shortform = FALSE)
```

# Index