# Package: RDelta (via r-universe)

October 29, 2026

**Type** Package

**Title** Parse and Query DELTA-Format Taxonomic Descriptions

**Version** 0.1.0

**Date** 2026-10-18 # Update to current date

**Maintainer** Mauro Cavalcanti <maurobio@gmail.com>

**Description** Provides an R interface to the 'tDelta' C++ library for parsing DELTA (DEscription Language for TAxonomy) format files <https://freedelta.sourceforge.net>. Supports loading character lists, item descriptions, and specifications, with efficient searching and matching of items based on character values. It also supports the generation of identification keys, diagnoses, natural-language descriptions, and clustering and ordination analyses by means of the DELTA programs 'CONFOR', 'KEY', and 'DIST'.

**License** GPL (>= 3)

**URL** https://github.com/maurobio/rdelta

**BugReports** https://github.com/maurobio/rdelta/issues

**Depends** R (>= 3.5.0)

**Imports** Rcpp (>= 1.0.0), graphics, methods, stats, utils

**LinkingTo** Rcpp

**Suggests** knitr, rmarkdown

**VignetteBuilder** knitr

**SystemRequirements** C++17

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**Config/roxygen2/version** 8.1.1

**Repository** https://maurobio.r-universe.dev

**Date/Publication** 2026-10-29 12:00:25 UTC

**RemoteUrl** https://github.com/maurobio/rdelta

**RemoteRef** HEAD

**RemoteSha** 935a651f97b9e3e412ac0331e078db05cbfcf10c

# Contents

attribute_count . . . 4
attribute_string . . . 4
char_count . . . 5
char_feature . . . 5
char_type . . . 6
char_type_name . . . 6
char_unit . . . 7
characters_df . . . 7
chars_filename . . . 8
check . . . 8
checkc . . . 9
checki . . . 10
cluster . . . 11
debug_all . . . 12
delta_char_types . . . 12
delta_version . . . 13
delta2sliks . . . 13
DeltaParser . . . 14
dependent_characters . . . 15
dependent_count . . . 15
dist . . . 16
find_matching . . . 17
first_matching . . . 18
get_all_depchar . . . 18
get_all_dependencies . . . 19
get_all_implicit_values . . . 20
get_attribute . . . 20
get_attributes_nb . . . 21
get_char_feature . . . 21
get_char_type . . . 22
get_char_unit . . . 22
get_chars_debug . . . 23
get_chars_nb . . . 23
get_depchar . . . 24
get_depchar_nb . . . 25
get_example_files . . . 26
get_filename . . . 26
get_implicit_value . . . 27
get_item . . . 28
get_item_name . . . 28
get_items_debug . . . 29

get_items_filename . . . 29
get_items_nb . . . 30
get_specs_debug . . . 30
get_specs_filename . . . 31
get_state . . . 31
get_states_nb . . . 32
get_version . . . 32
has_specifications . . . 33
implicit_value . . . 33
implicit_values . . . 34
is_chars_parsed . . . 34
is_dependent . . . 35
is_items_parsed . . . 36
is_parsed . . . 36
is_specs_parsed . . . 37
item_attributes . . . 37
item_count . . . 38
item_data . . . 38
item_name . . . 39
item_names . . . 40
items_df . . . 40
items_filename . . . 41
key . . . 41
load_delta . . . 43
matches . . . 44
natlan . . . 44
next_matching . . . 47
parse_attribute . . . 47
parse_characters . . . 48
parse_items . . . 48
parse_specs . . . 49
pcoa . . . 49
retrieve_all . . . 50
retrieve_all_chars . . . 50
retrieve_all_items . . . 51
retrieve_all_specs . . . 51
save_delta . . . 52
set_char_type . . . 53
set_char_type_by_name . . . 53
set_chars_filename . . . 54
set_filename . . . 54
set_items_file . . . 55
set_items_filename . . . 55
set_specs_file . . . 56
set_specs_filename . . . 56
specs_filename . . . 57
state_count . . . 57
state_name . . . 58

states . . . . . 58
strip_comments . . . . . 59
summary . . . . . 59
uncoded . . . . . 60

**Index** **62**

---

`attribute_count` *Get number of attributes for an item*

---

## Description

Get number of attributes for an item

## Usage

```
attribute_count(delta, itemnum)
```

## Arguments

| | |
|---|---|
| `delta` | A DeltaParser object |
| `itemnum` | Item number |

## Value

Integer number of attributes

---

`attribute_string` *Get attribute string*

---

## Description

Get attribute string

## Usage

```
attribute_string(delta, itemnum, attrnum)
```

## Arguments

| | |
|---|---|
| `delta` | A DeltaParser object |
| `itemnum` | Item number |
| `attrnum` | Attribute number |

## Value

Character string with the attribute

## char_count *Get number of characters*

### Description

Get number of characters

### Usage

```
char_count(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Integer number of characters

### Examples

```
delta <- tryCatch(
  load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  ),
  error = function(e) NULL
)
if (!is.null(delta)) char_count(delta)
```

## char_feature *Get character feature*

### Description

Get character feature

### Usage

```
char_feature(delta, charnum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |

## Value

Character string with the feature description

## Examples

```
delta <- tryCatch(
  load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  ),
  error = function(e) NULL
)
if (!is.null(delta)) char_feature(delta, 1)
```

---

# char_type *Get character type*

---

## Description

Get character type

## Usage

```
char_type(delta, charnum)
```

## Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |

## Value

Integer character type code

---

# char_type_name *Get character type name*

---

## Description

Get character type name

## Usage

```
char_type_name(delta, charnum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |

### Value

Character string with type name

---

## char_unit *Get character unit*

---

### Description

Get character unit

### Usage

```
char_unit(delta, charnum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |

### Value

Character string with the unit

---

## characters_df *Get character information as a data frame*

---

### Description

Extracts all character definitions from a DELTA dataset and returns them as a data frame.

### Usage

```
characters_df(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

A data.frame with character information

## chars_filename *Get characters filename*

### Description

Get characters filename

### Usage

```
chars_filename(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Character string with the filename

## check *Check the characters and items*

### Description

This function creates a directives file for the CONFOR program to check the characters and items in a DELTA dataset.

### Usage

```
check(delta, remove_temp = TRUE)
```

### Arguments

| | |
|---|---|
| delta | A Delta dataset object (created by load_delta) |
| remove_temp | Logical; if TRUE, remove temporary files after conversion (default = TRUE) |

### Details

The function assumes that the CONFOR program executable is available in the system PATH or in the current working directory.

### Value

Integer: 0 if check is successful, 1 if check fails

## Examples

```
# Requires the CONFOR executable on the PATH.

try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  check(delta)
})
```

---

checkc *Check the characters*

---

## Description

This function creates a directives file for the CONFOR program to check the characters in a DELTA dataset.

## Usage

```
checkc(delta, remove_temp = TRUE)
```

## Arguments

| | |
|---|---|
| delta | A Delta dataset object (created by load_delta) |
| remove_temp | Logical; if TRUE, remove temporary files after conversion (default = TRUE) |

## Details

The function assumes that the CONFOR program executable is available in the system PATH or in the current working directory.

## Value

Integer: 0 if check is successful, 1 if check fails

## Examples

```
# Requires the CONFOR executable on the PATH.

try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
```

```
  checkc(delta)
})
```

---

checki *Check the items*

---

## Description

This function creates a directives file for the CONFOR program to check the items in a DELTA dataset.

## Usage

```
checki(delta, remove_temp = TRUE)
```

## Arguments

| | |
|---|---|
| delta | A Delta dataset object (created by load_delta) |
| remove_temp | Logical; if TRUE, remove temporary files after conversion (default = TRUE) |

## Details

The function assumes that the CONFOR program executable is available in the system PATH or in the current working directory.

## Value

Integer: 0 if check is successful, 1 if check fails

## Examples

```
# Requires the CONFOR executable on the PATH.

try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  checki(delta)
})
```

`cluster` *Hierarchical Cluster Analysis*

## Description

Performs hierarchical cluster analysis on a distance matrix generated by the dist() function. This function reads the distance matrix and item names from the DIST output files and creates a dendrogram.

## Usage

```
cluster(notu, method = "average")
```

## Arguments

| | |
|---|---|
| `notu` | Number of operational taxonomic units (items) in the dataset. |
| `method` | The agglomeration method to be used. This should be one of "ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC) or "centroid" (= UPGMC). Default is "average". |

## Value

Invisibly returns a list with the following components:

**hclust** The `hclust` object produced by `hclust`, which can be further processed or plotted by the user.

**cophenetic** The cophenetic correlation coefficient (a single numeric value between -1 and 1) measuring how faithfully the dendrogram represents the original distance matrix. Values closer to 1 indicate a better fit.

**dist** The `dist` object representing the input distance matrix.

## Examples

```
# Requires the CONFOR and DIST executables on the PATH.

try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  dist(delta)
  res <- cluster(delta$get_items_nb())
  res$cophenetic
  res <- cluster(delta$get_items_nb(), method = "complete")
})
```

## debug_all *Get all debug output*

### Description

Get all debug output

### Usage

```
debug_all(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Character string with all debug output

## delta_char_types *Get character type constants*

### Description

Get character type constants

### Usage

```
delta_char_types()
```

### Value

Named integer vector of DELTA character types

## delta_version *Get package version*

### Description

Get package version

### Usage

```
delta_version()
```

### Value

Package version string

## delta2sliks *Convert DELTA dataset to SLIKS format*

### Description

Converts a DELTA dataset to SLIKS (Simple List of Interactive Key States) format, which is used for interactive identification keys.

### Usage

```
delta2sliks(
  delta,
  output_file = "data.js",
  exclude_numeric = TRUE,
  remove_temp = TRUE
)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object (created by load_delta) |
| output_file | Output file name (default: "data.js") |
| exclude_numeric | Logical; if TRUE, exclude numeric and text characters |
| remove_temp | Logical; if TRUE, remove temporary files after conversion |

### Details

The function assumes that the CONFOR program executable is available in the system PATH or in the current working directory.

## Value

Invisibly returns the path to the output file

## Examples

```
# Requires the CONFOR executable on the PATH.

try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  delta2sliks(delta, output_file = file.path(tempdir(), "data.js"))
})
```

---

`DeltaParser` *Create a DELTA Parser Object*

---

## Description

Creates a DeltaParser object for parsing DELTA files.

## Usage

```
DeltaParser(file)
```

## Arguments

| | |
|---|---|
| `file` | Path to the DELTA file to parse. |

## Value

An object of class `Rcpp_DeltaParser`.

## Examples

```
parser <- DeltaParser("path/to/delta/file.txt")
```

## dependent_characters *Get all dependent characters*

### Description

Get all dependent characters

### Usage

```
dependent_characters(delta, ccnum, ccstate)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| ccnum | Control character number |
| ccstate | Control character state |

### Value

Integer vector of dependent character numbers

## dependent_count *Get number of dependent characters*

### Description

Get number of dependent characters

### Usage

```
dependent_count(delta, ccnum, ccstate)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| ccnum | Control character number |
| ccstate | Control character state |

### Value

Integer count of dependent characters

dist *Generate Distance Matrix using Gower's Coefficient*

## Description

This function replicates the DIST command in the DELTA system for generating a distance matrix using Gower's coefficient. It creates the necessary directives files and runs the CONFOR and DIST programs.

## Usage

```
dist(
  delta,
  heading = NULL,
  match_overlap = FALSE,
  minimum_comparisons,
  phylip_format = FALSE,
  exclude_items = NULL,
  exclude_characters = NULL,
  remove_temp = TRUE
)
```

## Arguments

| | |
|---|---|
| `delta` | A Delta dataset object (created by `load_delta`) |
| `heading` | Heading for the output (default = NULL, uses heading from specs file) |
| `match_overlap` | For unordered multistate characters, the contribution of a character to the distance between two items is 0 if the items have any of the values of the character in common (default = FALSE) |
| `minimum_comparisons` | Minimum number of character comparisons required to calculate the distance between two items (default = ceiling(sqrt(number of included characters))) |
| `phylip_format` | Causes the taxon names to be interleaved with the rows of the distance matrix, as required by the program Phylip (default = FALSE) |
| `exclude_items` | A list of items to be excluded (default = NULL) |
| `exclude_characters` | A list of characters to be excluded (default = NULL) |
| `remove_temp` | Logical; if TRUE, remove temporary files after conversion |

## Details

The function assumes that the CONFOR and DIST program executables are available in the system PATH or in the current working directory.

## Value

Invisibly returns 0 on success

## Examples

```
# Requires the CONFOR and DIST executables on the PATH.

try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  dist(delta)
  dist(delta, match_overlap = TRUE, phylip_format = TRUE,
       minimum_comparisons = 10)
})
```

---

`find_matching` *Find all matching items*

---

## Description

Find all matching items

## Usage

```
find_matching(delta, charnum, values, strict = TRUE, with_extrval = TRUE)
```

## Arguments

| | |
|---|---|
| `delta` | A DeltaParser object |
| `charnum` | Character number |
| `values` | Numeric vector of values |
| `strict` | Logical; strict comparison |
| `with_extrval` | Logical; include extreme values |

## Value

Integer vector of matching item numbers

## first_matching *Find first matching item*

### Description

Find first matching item

### Usage

```
first_matching(delta, charnum, values, strict = TRUE, with_extrval = TRUE)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |
| values | Numeric vector of values |
| strict | Logical; strict comparison |
| with_extrval | Logical; include extreme values |

### Value

Integer item number or 0 if none

## get_all_depchar *Get all dependent characters*

### Description

Get all dependent characters

### Usage

```
get_all_depchar(delta, ccnum, ccstate)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| ccnum | Control character number |
| ccstate | Control character state |

### Value

Integer vector of dependent character numbers

## Examples

```
try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  get_all_depchar(delta, 13, 2)
})
```

---

# get_all_dependencies *Get all dependency relationships*

---

## Description

Get all dependency relationships

## Usage

```
get_all_dependencies(delta)
```

## Arguments

| | |
|---|---|
| delta | A DeltaParser object |

## Value

List of all dependency relationships

## Examples

```
try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  get_all_dependencies(delta)
})
```

## get_all_implicit_values *Get all implicit values*

### Description

Get all implicit values

### Usage

```
get_all_implicit_values(delta, iv_type = 1)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| iv_type | Implicit value type (1 or 2, default: 1) |

### Value

Integer vector of implicit values

### Examples

```
try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  get_all_implicit_values(delta)
})
```

## get_attribute *Get attribute string (C++ style)*

### Description

Get attribute string (C++ style)

### Usage

```
get_attribute(delta, itemnum, attrnum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| itemnum | Item number |
| attrnum | Attribute number |

### Value

Character string with the attribute

---

## get_attributes_nb *Get number of attributes (C++ style)*

---

### Description

Get number of attributes (C++ style)

### Usage

```
get_attributes_nb(delta, itemnum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| itemnum | Item number |

### Value

Integer number of attributes

---

## get_char_feature *Get character feature (C++ style)*

---

### Description

Get character feature (C++ style)

### Usage

```
get_char_feature(delta, charnum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |

### Value

Character string with the feature description

## get_char_type *Get character type (C++ style)*

### Description

Get character type (C++ style)

### Usage

```
get_char_type(delta, charnum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |

### Value

Integer character type code

## get_char_unit *Get character unit (C++ style)*

### Description

Get character unit (C++ style)

### Usage

```
get_char_unit(delta, charnum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |

### Value

Character string with the unit

## get_chars_debug *Get character debug output*

### Description

Get character debug output

### Usage

```
get_chars_debug(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Character string with debug output

## get_chars_nb *Get number of characters (C++ style)*

### Description

Get number of characters (C++ style)

### Usage

```
get_chars_nb(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Integer number of characters

get_depchar *Get dependent character by rank (C++ style)*

## Description

Get dependent character by rank (C++ style)

Get dependent character at specific rank

## Usage

```
get_depchar(delta, ccnum, ccstate, rank = 1)

get_depchar(delta, ccnum, ccstate, rank = 1)
```

## Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| ccnum | Control character number |
| ccstate | Control character state |
| rank | Rank position (default: 1) |

## Value

Integer dependent character number

Dependent character number or 0 if none

## Examples

```
try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  get_depchar(delta, 13, 2, 1)
})
```

`get_depchar_nb` *Get number of dependent characters (C++ style)*

## Description

Get number of dependent characters (C++ style)

Get number of dependent characters

## Usage

```
get_depchar_nb(delta, ccnum, ccstate)

get_depchar_nb(delta, ccnum, ccstate)
```

## Arguments

| | |
|---|---|
| `delta` | A DeltaParser object |
| `ccnum` | Control character number |
| `ccstate` | Control character state |

## Value

Integer count of dependent characters

Number of dependent characters

## Examples

```
try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  get_depchar_nb(delta, 13, 2)
})
```

## get_example_files *Get example file paths for testing*

### Description

Get example file paths for testing

### Usage

```
get_example_files(dataset = "viola")
```

### Arguments

| | |
|---|---|
| dataset | Character string: "viola" or "grass" |

### Value

List with file paths

## get_filename *Get filename (C++ style)*

### Description

Get filename (C++ style)

### Usage

```
get_filename(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Character string with the filename

`get_implicit_value` *Get implicit value (C++ style)*

## Description

Get implicit value (C++ style)

Get implicit value for a character

## Usage

```
get_implicit_value(delta, charnum, iv_type = 1)

get_implicit_value(delta, charnum, iv_type = 1)
```

## Arguments

| | |
|---|---|
| `delta` | A DeltaParser object |
| `charnum` | Character number |
| `iv_type` | Implicit value type (1 or 2, default: 1) |

## Value

Integer implicit value

Integer value or NA if not defined

## Examples

```
try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  get_implicit_value(delta, 1)
})
```

## get_item *Get complete item data as a list*

### Description

Retrieves all data for a specific item.

### Usage

```
get_item(delta, itemnum, include_comments = FALSE)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| itemnum | Integer item number |
| include_comments | Logical; if TRUE, include comments in item names |

### Value

A list with item name and attributes

## get_item_name *Get item name (C++ style)*

### Description

Get item name (C++ style)

### Usage

```
get_item_name(delta, itemnum, include_comments = TRUE)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| itemnum | Item number |
| include_comments | Logical; if TRUE, include comments in item names |

### Value

Character string with item name

## get_items_debug *Get items debug output*

### Description

Get items debug output

### Usage

```
get_items_debug(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Character string with debug output

## get_items_filename *Get items filename (C++ style)*

### Description

Get items filename (C++ style)

### Usage

```
get_items_filename(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Character string with the filename

## get_items_nb *Get number of items (C++ style)*

### Description

Get number of items (C++ style)

### Usage

```
get_items_nb(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Integer number of items

## get_specs_debug *Get specifications debug output*

### Description

Get specifications debug output

### Usage

```
get_specs_debug(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Character string with debug output

## get_specs_filename *Get specifications filename (C++ style)*

### Description

Get specifications filename (C++ style)

### Usage

```
get_specs_filename(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Character string with the filename

## get_state *Get state name (C++ style)*

### Description

Get state name (C++ style)

### Usage

```
get_state(delta, charnum, statenum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |
| statenum | State number |

### Value

Character string with state name

## get_states_nb *Get number of states (C++ style)*

### Description

Get number of states (C++ style)

### Usage

```
get_states_nb(delta, charnum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |

### Value

Integer number of states

## get_version *Get version of the underlying C++ library*

### Description

Get version of the underlying C++ library

### Usage

```
get_version(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Version string

## has_specifications *Check if specifications are available*

### Description

Check if specifications are available

### Usage

```
has_specifications(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Logical

## implicit_value *Get implicit value*

### Description

Get implicit value

### Usage

```
implicit_value(delta, charnum, iv_type = 1)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |
| iv_type | Implicit value type (1 or 2) |

### Value

Integer implicit value

## implicit_values *Get all implicit values*

### Description

Get all implicit values

### Usage

```
implicit_values(delta, iv_type = 1)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| iv_type | Implicit value type (1 or 2) |

### Value

Integer vector of implicit values

## is_chars_parsed *Check if characters are parsed*

### Description

Check if characters are parsed

### Usage

```
is_chars_parsed(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Logical

`is_dependent` *Check if character is dependent*

## Description

Check if character is dependent

Check if a character is dependent

## Usage

```
is_dependent(delta, dcnum, ccnum, ccstate)

is_dependent(delta, dcnum, ccnum, ccstate)
```

## Arguments

| | |
|---|---|
| `delta` | A DeltaParser object |
| `dcnum` | Dependent character number |
| `ccnum` | Control character number |
| `ccstate` | Control character state |

## Value

Logical indicating if dependent

Logical indicating if dcnum is dependent

## Examples

```
try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  is_dependent(delta, 14, 13, 2)
})
```

## is_items_parsed *Check if items are parsed*

### Description

Check if items are parsed

### Usage

```
is_items_parsed(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Logical

## is_parsed *Check if parsed (C++ style)*

### Description

Check if parsed (C++ style)

### Usage

```
is_parsed(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Logical

## is_specs_parsed *Check if specifications are parsed*

### Description

Check if specifications are parsed

### Usage

```
is_specs_parsed(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Logical

## item_attributes *Get item attributes as data frame*

### Description

Get item attributes as data frame

### Usage

```
item_attributes(delta, itemnum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| itemnum | Item number |

### Value

Data frame with attributes

## item_count *Get number of items*

### Description

Get number of items

### Usage

```
item_count(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Integer number of items

### Examples

```
delta <- tryCatch(
  load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  ),
  error = function(e) NULL
)
if (!is.null(delta)) item_count(delta)
```

## item_data *Get item data as list*

### Description

Get item data as list

### Usage

```
item_data(delta, itemnum, include_comments = FALSE)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| itemnum | Item number |
| include_comments | Logical; if TRUE, include comments in item names |

## Value

List with item data

---

`item_name` *Get item name*

---

## Description

Get item name

## Usage

```
item_name(delta, itemnum, include_comments = TRUE)
```

## Arguments

| | |
|---|---|
| `delta` | A DeltaParser object |
| `itemnum` | Item number |
| `include_comments` | Logical; if TRUE, include comments in item names |

## Value

Character string with item name

## Examples

```
delta <- tryCatch(
  load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  ),
  error = function(e) NULL
)
if (!is.null(delta)) item_name(delta, 1)
```

## item_names *Get all item names*

### Description

Get all item names

### Usage

```
item_names(delta, include_comments = FALSE)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| include_comments | Logical; if TRUE, include comments in item names |

### Value

Character vector of item names

## items_df *Get item information as a data frame*

### Description

Extracts all item information from a DELTA dataset and returns it as a data frame.

### Usage

```
items_df(delta, include_comments = FALSE)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| include_comments | Logical; if TRUE, include comments in item names |

### Value

A data.frame with item information

## items_filename *Get items filename*

### Description

Get items filename

### Usage

```
items_filename(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Character string with the filename

## key *Generate Dichotomous Key using CONFOR/KEY*

### Description

This function replicates the KEY command in the DELTA system for generating dichotomous keys. It creates the necessary directives files and runs the CONFOR and KEY programs.

### Usage

```
key(
  delta,
  heading = NULL,
  add_character_numbers = FALSE,
  no_bracketted_key = FALSE,
  no_tabular_key = TRUE,
  number_of_confirmatory_characters = 0,
  treat_characters_as_variable = NULL,
  use_normal_values = NULL,
  character_reliabilities = NULL,
  key_states = NULL,
  abase = 2,
  rbase = 1.4,
  reuse = 1.01,
  varywt = 0.8,
  print_width = 80,
  exclude_items = NULL,
```

```
  exclude_characters = NULL,
  remove_temp = TRUE
)
```

## Arguments

| | |
|---|---|
| delta | A Delta dataset object (created by load_delta) |
| heading | Heading for the output (default = NULL, uses heading from specs file) |
| add_character_numbers | Character numbers are to be inserted in bracketed keys (default = FALSE) |
| no_bracketted_key | Suppresses the production of a key in the conventional, 'bracketed' format (default = FALSE) |
| no_tabular_key | Suppresses the production of a key in tabular format (default = TRUE) |
| number_of_confirmatory_characters | Number of confirmatory characters that will be sought for each main character in the key (default = 0, maximum = 4) |
| treat_characters_as_variable | A list of characters for which particular attributes are to be treated as 'variable' (default = NULL) |
| use_normal_values | A list of numeric characters for which extreme values are to be ignored and normal values used when translating into other formats (default = NULL) |
| character_reliabilities | A list of the 'reliabilities' of the characters (default = NULL) |
| key_states | A list of numeric character states to be divided into ranges for use as states in identification keys (default = NULL) |
| abase | Sets the base of the logarithmic item abundance scale (default = 2) |
| rbase | Sets the base of the logarithmic character-reliability scale (default = 1.4) |
| reuse | Attempt to minimize the number of different characters used in the key (default = 1.01) |
| varywt | Sets the value of a parameter that determines the treatment of intra-taxon variability in the process of character selection (default = 0.8) |
| print_width | Maximum line length for output of the keys (default = 80) |
| exclude_items | A list of items to be excluded (default = NULL) |
| exclude_characters | A list of characters to be excluded (default = NULL) |
| remove_temp | Logical; if TRUE, remove temporary files after conversion (default = TRUE) |

## Details

The function assumes that the CONFOR and KEY program executables are available in the system PATH or in the current working directory.

## Value

Invisibly returns the key as a character vector on success

## Examples

```
# Requires the CONFOR and KEY executables on the PATH.

try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  key(delta)
  key(delta, abase = 2, rbase = 1.4,
      no_tabular_key = TRUE, number_of_confirmatory_characters = 2)
})
```

---

# load_delta *Load a DELTA dataset*

---

## Description

Create a DeltaParser object from DELTA format files.

## Usage

```
load_delta(chars_file, items_file, specs_file = NULL)
```

## Arguments

| | |
|---|---|
| chars_file | Path to the DELTA characters file |
| items_file | Path to the DELTA items file |
| specs_file | Optional path to the DELTA specifications file |

## Value

A DeltaParser Rcpp module object

## Examples

```
# Load the packaged example dataset
delta <- tryCatch(
  load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
```

```
  ),
  error = function(e) NULL
)

if (!is.null(delta)) {
  char_count(delta)
  char_feature(delta, 1)
}
```

---

`matches` *Check if item matches values*

---

## Description

Check if item matches values

## Usage

```
matches(delta, itemnum, charnum, values, strict = TRUE, with_extrval = TRUE)
```

## Arguments

| | |
|---|---|
| `delta` | A DeltaParser object |
| `itemnum` | Item number |
| `charnum` | Character number |
| `values` | Numeric vector of values |
| `strict` | Logical; strict comparison |
| `with_extrval` | Logical; include extreme values |

## Value

Logical indicating if item matches

---

`natlan` *Translate DELTA-format data into natural language descriptions via CONFOR*

---

## Description

This function replicates the TRANSLATE INTO NATURAL LANGUAGE command in the DELTA system. It creates the necessary directives file and runs the CONFOR program to generate natural language descriptions.

## Usage

```
natlan(
  delta,
  heading = NULL,
  replace_angle_brackets = TRUE,
  omit_character_numbers = TRUE,
  omit_inapplicables = TRUE,
  omit_comments = TRUE,
  omit_inner_comments = TRUE,
  omit_final_comma = TRUE,
  translate_implicit_values = FALSE,
  omit_lower_for_characters = NULL,
  omit_or_for_characters = NULL,
  omit_period_for_characters = NULL,
  new_paragraphs_at_characters = NULL,
  item_subheadings = NULL,
  link_characters = NULL,
  replace_semicolon_by_comma = NULL,
  print_width = 80,
  exclude_items = NULL,
  exclude_characters = NULL,
  vocabulary = NULL,
  remove_temp = TRUE
)
```

## Arguments

| | |
|---|---|
| `delta` | A Delta dataset object (created by `load_delta`) |
| `heading` | Heading for the descriptions (default = NULL, uses heading from specs file) |
| `replace_angle_brackets` | Replace angle brackets with parentheses (default = TRUE) |
| `omit_character_numbers` | Omit character numbers from descriptions (default = TRUE) |
| `omit_inapplicables` | Omit 'or not applicable' from descriptions (default = TRUE) |
| `omit_comments` | Omit comments in attributes (default = TRUE) |
| `omit_inner_comments` | Omit inner comments in character list and attributes (default = TRUE) |
| `omit_final_comma` | Omit final comma before 'and' (default = TRUE) |
| `translate_implicit_values` | Output implicit values in descriptions (default = FALSE) |
| `omit_lower_for_characters` | List of numeric characters to omit lower values |
| `omit_or_for_characters` | List of characters to omit 'or' between alternatives |

`omit_period_for_characters`
: List of characters where period would normally terminate a sentence

`new_paragraphs_at_characters`
: List of characters where new paragraphs should start

`item_subheadings`
: List of subheadings for items (delimiter/character/value)

`link_characters`
: List of characters specifying how attributes are combined

`replace_semicolon_by_comma`
: List of characters to use comma instead of semicolon

`print_width`
: Maximum line length for output (default = 80)

`exclude_items`
: List of items to exclude

`exclude_characters`
: List of characters to exclude

`vocabulary`
: List for custom vocabulary (default = NULL)

`remove_temp`
: Logical; if TRUE, remove temporary files after conversion (default = TRUE)

## Details

The function assumes that the CONFOR program executable is available in the system PATH or in the current working directory.

## Value

Invisibly returns the descriptions as a character vector

## Examples

```
# Requires the CONFOR executable on the PATH.

try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  natlan(delta)
  natlan(delta,
         heading = "Flora of Region X - species descriptions",
         item_subheadings = list(
           "87" = "Transverse section of lamina",
           "96" = "Leaf epidermis"
         ))
})
```

## next_matching *Find next matching item*

### Description

Find next matching item

### Usage

```
next_matching(delta, charnum, values, strict = TRUE, with_extrval = TRUE)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |
| values | Numeric vector of values |
| strict | Logical; strict comparison |
| with_extrval | Logical; include extreme values |

### Value

Integer item number or 0 if none

## parse_attribute *Parse a DELTA attribute string*

### Description

Parse a DELTA attribute string

### Usage

```
parse_attribute(attr_string)
```

### Arguments

| | |
|---|---|
| attr_string | A DELTA attribute string |

### Value

A list with parsed components: `charnum` (integer), `comment` (character), and `alternatives` (list of parsed value lists).

## parse_characters *Parse characters file*

### Description

Parse characters file

### Usage

```
parse_characters(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Logical indicating success

## parse_items *Parse items file*

### Description

Parse items file

### Usage

```
parse_items(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Logical indicating success

## parse_specs *Parse specifications file*

### Description

Parse specifications file

### Usage

```
parse_specs(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Logical indicating success

## pcoa *Principal Coordinates Analysis*

### Description

Performs principal coordinates analysis (PCoA) on a distance matrix generated by the dist() function. This function reads the distance matrix and item names from the DIST output files and creates a PCoA scatterplot.

### Usage

```
pcoa(notu)
```

### Arguments

| | |
|---|---|
| notu | Number of operational taxonomic units (items) in the dataset. |

### Value

Invisibly returns a list with the following components:

**points** A two-column matrix of the coordinates of the items on the first two principal coordinate axes.

**eig** The eigenvalues of the PCoA solution, as returned by cmdscale.

**variance** A numeric vector of length 2 giving the proportion of total variance explained by Axis 1 and Axis 2 (values between 0 and 1).

**cumulative** A single numeric value giving the cumulative proportion of variance explained by the first two axes combined.

**cmdscale** The full result object returned by cmdscale, for users who need access to all components.

## Examples

```
# Requires the CONFOR and DIST executables on the PATH.

try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  dist(delta)
  res <- pcoa(delta$get_items_nb())
  res$variance
  res$cumulative
})
```

---

# retrieve_all *Retrieve ALL information for debugging*

---

## Description

This matches the C++ retrieve_all() method from the original tDelta library, combining characters, items, and specifications output.

## Usage

```
retrieve_all(delta)
```

## Arguments

| | |
|---|---|
| delta | A DeltaParser object |

## Value

Character string with all debug information

---

# retrieve_all_chars *Retrieve all character information for debugging*

---

## Description

This matches the C++ retrieve_all() method from the original tDelta library.

## Usage

```
retrieve_all_chars(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Character string with debug information

---

## retrieve_all_items *Retrieve all item information for debugging*

---

### Description

This matches the C++ retrieve_all() method from the original tDelta library.

### Usage

```
retrieve_all_items(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Character string with debug information

---

## retrieve_all_specs *Retrieve all specifications information for debugging*

---

### Description

This matches the C++ retrieve_all() method from the original tDelta library.

### Usage

```
retrieve_all_specs(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Character string with debug information

save_delta *Save a Delta dataset to disk*

## Description

This function saves a Delta dataset object to disk as three plain text files: "chars", "items", and "specs" in the DELTA format.

## Usage

```
save_delta(delta, dir, prefix = "", overwrite = FALSE, width = 79)
```

## Arguments

| | |
|---|---|
| delta | A Delta dataset object (Rcpp_DeltaParser from load_delta()) |
| dir | Path to the directory where files should be saved. This argument is required: the function never writes to the current working directory by default. Use `tempdir()` if you only need a scratch location. |
| prefix | Optional prefix for filenames (default: "") |
| overwrite | Whether to overwrite existing files (default: FALSE) |
| width | Maximum line width for output files (default: 79, range: 40-132) |

## Value

Invisibly returns the input delta object

## Examples

```
try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )

  # Write to a temporary directory
  save_delta(delta, dir = tempdir())

  # Write to a user-chosen directory with a prefix
  save_delta(delta, dir = tempdir(), prefix = "my_", overwrite = TRUE)

  # Use a wider output width
  save_delta(delta, dir = tempdir(), width = 120)
})
```

## set_char_type *Set character type*

### Description

Set character type

### Usage

```
set_char_type(delta, charnum, chartype)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |
| chartype | Integer character type code |

### Value

No return value, called for side effects. The character type of charnum is modified in delta in place.

## set_char_type_by_name *Set character type by name*

### Description

Set character type by name

### Usage

```
set_char_type_by_name(delta, charnum, type_name)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |
| type_name | Character type name |

### Value

No return value, called for side effects. The character type of charnum is modified in delta in place.

## set_chars_filename *Set characters filename*

### Description

Set characters filename

### Usage

```
set_chars_filename(delta, fname, parse = TRUE)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| fname | New filename |
| parse | Logical, parse the file immediately |

### Value

No return value, called for side effects. The characters filename stored in `delta` is updated.

## set_filename *Set filename (C++ style)*

### Description

Set filename (C++ style)

### Usage

```
set_filename(delta, fname, parse = TRUE)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| fname | New filename |
| parse | Logical, parse the file immediately |

### Value

No return value, called for side effects. The characters filename stored in `delta` is updated.

## set_items_file *Set items filename*

### Description

Set items filename

### Usage

```
set_items_file(delta, fname, parse = TRUE)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| fname | New filename |
| parse | Logical, parse the file immediately |

### Value

No return value, called for side effects. The items filename stored in `delta` is updated.

## set_items_filename *Set items filename (C++ style)*

### Description

Set items filename (C++ style)

### Usage

```
set_items_filename(delta, fname, parse = TRUE)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| fname | New filename |
| parse | Logical, parse the file immediately |

### Value

No return value, called for side effects. The items filename stored in `delta` is updated.

## set_specs_file *Set specifications filename*

### Description

Set specifications filename

### Usage

```
set_specs_file(delta, fname, parse = TRUE)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| fname | New filename |
| parse | Logical, parse the file immediately |

### Value

No return value, called for side effects. The specifications filename stored in `delta` is updated.

## set_specs_filename *Set specifications filename (C++ style)*

### Description

Set specifications filename (C++ style)

### Usage

```
set_specs_filename(delta, fname, parse = TRUE)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| fname | New filename |
| parse | Logical, parse the file immediately |

### Value

No return value, called for side effects. The specifications filename stored in `delta` is updated.

## specs_filename *Get specifications filename*

### Description

Get specifications filename

### Usage

```
specs_filename(delta)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |

### Value

Character string with the filename

## state_count *Get number of states for a character*

### Description

Get number of states for a character

### Usage

```
state_count(delta, charnum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |

### Value

Integer number of states

## state_name *Get state name*

### Description

Get state name

### Usage

```
state_name(delta, charnum, statenum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |
| statenum | State number |

### Value

Character string with state name

## states *Get all states for a character*

### Description

Get all states for a character

### Usage

```
states(delta, charnum)
```

### Arguments

| | |
|---|---|
| delta | A DeltaParser object |
| charnum | Character number |

### Value

Character vector of state names

## strip_comments *Remove comments from a DELTA string*

### Description

Remove comments from a DELTA string

### Usage

```
strip_comments(src)
```

### Arguments

| | |
|---|---|
| src | A string with DELTA comments |

### Value

A character string with all DELTA comments (delimited by < and >) removed.

## summary *Generate a statistical summary of taxonomic characters using CONFOR*

### Description

This function creates a directives file for the CONFOR program to generate a statistical summary of characters for a DELTA dataset. The summary includes character types, states, and frequency distributions.

### Usage

```
summary(delta, heading = NULL, remove_temp = TRUE)
```

### Arguments

| | |
|---|---|
| delta | A Delta dataset object (created by load_delta) |
| heading | Heading for the summary (default = NULL, uses heading from specs file) |
| remove_temp | Logical; if TRUE, remove temporary files after conversion (default = TRUE) |

### Details

The function assumes that the CONFOR program executable is available in the system PATH or in the current working directory.

## Value

Invisibly returns the path to the generated summary file

## Examples

```
# Requires the CONFOR executable on the PATH.

try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  summary(delta)
  summary(delta, heading = "Genre Viola")
})
```

---

uncoded *Generate a list of uncoded characters using CONFOR*

---

## Description

This function creates a directives file for the CONFOR program to generate a list of characters that are not coded (i.e., have no data) in a DELTA dataset. This is useful for identifying missing data and gaps in taxonomic descriptions.

## Usage

```
uncoded(delta, heading = NULL, remove_temp = TRUE)
```

## Arguments

| | |
|---|---|
| `delta` | A Delta dataset object (created by `load_delta`) |
| `heading` | Heading for the output (default = NULL, uses heading from specs file) |
| `remove_temp` | Logical; if TRUE, remove temporary files after conversion (default = TRUE) |

## Details

The function assumes that the CONFOR program executable is available in the system PATH or in the current working directory.

## Value

Invisibly returns the path to the generated uncoded characters file

## Examples

```
# Requires the CONFOR executable on the PATH.

try({
  delta <- load_delta(
    system.file("extdata", "chars_viola", package = "RDelta"),
    system.file("extdata", "items_viola", package = "RDelta"),
    system.file("extdata", "specs_viola", package = "RDelta")
  )
  uncoded(delta)
  uncoded(delta, heading = "Genre Viola - Missing data report")
})
```

# Index


attribute_count, 4
attribute_string, 4

char_count, 5
char_feature, 5
char_type, 6
char_type_name, 6
char_unit, 7
characters_df, 7
chars_filename, 8
check, 8
checkc, 9
checki, 10
cluster, 11
cmdscale, *49*

debug_all, 12
delta2sliks, 13
delta_char_types, 12
delta_version, 13
DeltaParser, 14
dependent_characters, 15
dependent_count, 15
dist, 16

find_matching, 17
first_matching, 18

get_all_depchar, 18
get_all_dependencies, 19
get_all_implicit_values, 20
get_attribute, 20
get_attributes_nb, 21
get_char_feature, 21
get_char_type, 22
get_char_unit, 22
get_chars_debug, 23
get_chars_nb, 23
get_depchar, 24
get_depchar_nb, 25
get_example_files, 26
get_filename, 26
get_implicit_value, 27
get_item, 28
get_item_name, 28
get_items_debug, 29
get_items_filename, 29
get_items_nb, 30
get_specs_debug, 30
get_specs_filename, 31
get_state, 31
get_states_nb, 32
get_version, 32

has_specifications, 33
hclust, *11*

implicit_value, 33
implicit_values, 34
is_chars_parsed, 34
is_dependent, 35
is_items_parsed, 36
is_parsed, 36
is_specs_parsed, 37
item_attributes, 37
item_count, 38
item_data, 38
item_name, 39
item_names, 40
items_df, 40
items_filename, 41

key, 41

load_delta, *8–10*, *13*, *16*, *42*, 43, *45*, *59, 60*

matches, 44

natlan, 44
next_matching, 47

parse_attribute, 47
parse_characters, 48
parse_items, 48
parse_specs, 49
pcoa, 49

retrieve_all, 50
retrieve_all_chars, 50
retrieve_all_items, 51
retrieve_all_specs, 51

save_delta, 52
set_char_type, 53
set_char_type_by_name, 53
set_chars_filename, 54
set_filename, 54
set_items_file, 55
set_items_filename, 55
set_specs_file, 56
set_specs_filename, 56
specs_filename, 57
state_count, 57
state_name, 58
states, 58
strip_comments, 59
summary, 59

uncoded, 60