# Package ‘DEXiR’

January 20, 2025

**Title** 'DEXi' Library

**Version** 1.0.2

**Description** A software package for using 'DEXi' models. 'DEXi' models are
hierarchical qualitative multi-criteria decision models developed according to the method
DEX (Decision EXpert, <https://dex.ijs.si/documentation/DEX_Method/DEX_Method.html>),
using the program 'DEXi' (<https://kt.ijs.si/MarkoBohanec/dexi.html>) or
'DEXiWin' (<https://dex.ijs.si/dexisuite/dexiwin.html>).
A typical workflow with 'DEXiR' consists of:
(1) reading a '.dxi' file, previously made using the 'DEXi' software (function read_dexi()),
(2) making a data frame containing input values of one or more decision alternatives,
(3) evaluating those alternatives (function evaluate()),
(4) analyzing alternatives (selective_explanation(), plus_minus(), compare_alternatives()),
(5) drawing charts.
'DEXiR' is restricted to using models produced externally by the 'DEXi' software and does not
provide functionality for creating and/or editing 'DEXi' models directly in 'R'.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Suggests** testthat (>= 3.0.0), grDevices, ggplot2, GGally, fmsb

**Config/testthat/edition** 3

**Imports** utils, xml2, methods, stringr, graphics

**Collate** 'DEXiR.R' 'DexiClasses.R' 'DexiAlternatives.R' 'DexiUtils.R'
'DexiAnalysis.R' 'DexiFunctions.R' 'DexiScales.R'
'DexiAttributes.R' 'DexiCharts.R' 'DexiData.R' 'DexiEvaluate.R'
'DexiModels.R' 'DexiValues.R'

**NeedsCompilation** no

**Author** Marko Bohanec [aut, cre] (<https://orcid.org/0000-0003-4317-2833>)

**Maintainer** Marko Bohanec <marko.bohanec@ijs.si>

**Repository** CRAN

**Date/Publication** 2024-09-17 16:30:09 UTC

# Contents

---

`DEXiR-package` *DEXiR: A package for using DEXi models in R*

---

## Description

DEXiR is a software package for using DEXi models in R. The main function is evaluating decision alternatives using a model previously developed by DEXi software.

## DEXi Models

DEXi models are hierarchical qualitative rule-based multi-criteria decision models developed using the method DEX (Decision EXpert, `https://en.wikipedia.org/wiki/Decision_EXpert`), using the program DEXi (`https://kt.ijs.si/MarkoBohanec/dexi.html`) or DEXiWin (`https://dex.ijs.si/dexisuite/dexiwin.html`).

In general, a DEXi model consists of a hierarchy of qualitative (symbolic linguistic, discrete) variables, called *attributes*. Each attribute represents some observable property (such as Price or Performance) of decision alternatives under study. An attribute can take values from a set of words (such as "low; medium; high" or "unacc; acc; good; exc"), which is usually small (up to five elements) and preferentially ordered from "bad" to "good" values.

The *hierarchy* of attributes represents a decomposition of a decision problem into sub-problems, so that higher-level attributes depend on the lower-level ones. Consequently, the terminal nodes represent inputs, and non-terminal attributes represent the outputs of the model. Among these, the most important are one or more root attributes, which represent the final evaluation(s) of the alternatives.

The *evaluation* of decision alternatives (i.e., hierarchical aggregation of values from model inputs to outputs) is governed by *decision rules*, defined for each non-terminal attribute by the creator of the model (usually referred to as a "decision maker").

## Terminological remarks

**DEX** DEX (Decision EXpert) refers to a general multi-attribute decision modeling method, characterized by using qualitative attribute hierarchies and decision tables. For further information, see (Trdin, Bohanec, 2018) and (Bohanec, 2022).

**DEXi** DEXi ("DEX for instruction") refers to DEXi software. DEXi implements a subset of DEX, for instance, it is restricted to set-based evaluation methods. DEXi supports the creation and editing of *DEXi models*, which are saved on `.dxi` files and subsequently read by DEXiR for processing in R. For further information on DEXi, see `https://kt.ijs.si/MarkoBohanec/dexi.html`.

**DEXiWin** A new backward-compatible implementation of DEXi, aimed at gradually replacing it in the future. For further information on DEXiWin and related software, see `https://dex.ijs.si/dexisuite/dexisuite.html`.

**DEXiR** DEXiR is this R package. It is capable of reading and processing DEXi models with some extensions towards the full DEX (for example, using value distributions).

## DEXiR Functionality

Models developed using the DEXi software are stored in XML-formatted `.dxi` files. In order to use DEXi models in R, DEXiR supports the following tasks:

1. Reading DEXi models from `.dxi` files into the R environment, using `read_dexi`.
2. Making data frames containing data (both input and output) about considered decision alternatives, using `set_alternative`.
3. Evaluating decision alternatives, using `evaluate`.
4. Analyzing alternatives (`selective_explanation`, `plus_minus`, `compare_alternatives`).
5. Drawing charts.

By default, evaluation is based on sets, which is a standard evaluation procedure of DEXi. DEXiR extends this by supporting:

- evaluations using probabilistic and fuzzy value distributions (see `evaluate`);
- "pruned" evaluation, when the evaluation starts from selected non-terminal attribute(s) upwards.

## Limitations

DEXiR has been designed to facilitate *using* DEXi models in R produced externally by the DEXi software. DEXiR does not provide any explicit means for creating and/or editing DEXi models in R.

## A typical DEXiR workflow

This example uses a simple DEXi model for evaluating cars, which is distributed together with the DEXi software (including DEXiR) and is used throughout DEX literature to illustrate the methodological approach (`https://en.wikipedia.org/wiki/Decision_EXpert`).

First, this model is loaded into R and printed as follows:

```
> Car <- read_dexi("data/Car.dxi")
> Car
DEXi Model:  CAR_MODEL
Description: Car demo
index id          structure          scale                    funct
  [1] CAR_MODEL   CAR_MODEL
  [2] CAR         +- CAR             unacc; acc; good; exc (+) 12 3x4
  [3] PRICE         |- PRICE         high; medium; low (+)     9 3x3
  [4] BUY.PRICE     | |- BUY.PRICE   high; medium; low (+)
  [5] MAINT.PRICE   | +- MAINT.PRICE high; medium; low (+)
  [6] TECH.CHAR.    +- TECH.CHAR.    bad; acc; good; exc (+)   9 3x3
  [7] COMFORT         |- COMFORT     small; medium; high (+)   36 3x4x3
  [8] X.PERS          | |- #PERS     to_2; 3-4; more (+)
  [9] X.DOORS         | |- #DOORS    2; 3; 4; more (+)
 [10] LUGGAGE         | +- LUGGAGE   small; medium; big (+)
 [11] SAFETY          +- SAFETY      small; medium; high (+)
```

Rows in the table correspond to individual attributes. The columns represent the following:

`index` Indices of attributes.

`id` Unique attribute names, generated by DEXiR from original DEXi names, in order to provide syntactically correct variable names in R and allow unambiguous referencing of attributes.

`structure` The hierarchical structure of attributes, named as in the original DEXi model.

`scale` Value scales associated with each attribute. The symbol "(+)" indicates that the corresponding scale is ordered preferentially in increasing order.

`funct` Information about the size (number of rules) and dimensions of the corresponding decision tables.

Looking at the structure of attributes, please notice that the attribute at index `[1]` is virtual and does not actually appear in the original DEXi model. It is necessary in DEXiR to facilitate models that have multiple root attributes. The "real" root of the Car model is actually `[2]` CAR. It depends on two lower-level attributes, PRICE and TECH.CHAR. These are decomposed further. Overall, the model consists of

- six input (*basic*) attributes: BUY.PRICE, MAINT.PRICE, X.PERS, X.DOORS, LUGGAGE and SAFETY, and
- four output (*aggregate*) attributes: CAR, PRICE, TECH.CHAR. and COMFORT.

Among the latter, CAR is the most important and represents the overall evaluation of cars.

The next step usually consists of defining a data frame representing decision alternatives (i.e., cars in this case). The Car model already comes with a data table about two cars:

```
> Car$alternatives
  name CAR PRICE BUY.PRICE MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFETY
1 Car1   4     3         2           3          4       3      3       3       3      3
2 Car2   3     2         2           2          3       3      3       3       3      2
```

In this data frame, attribute values are represented by ordinal numbers w.r.t. the corresponding scales. A more readable output can be made using `DexiModel$as_character`:

```
> Car$as_character(Car$alternatives)
  name  CAR  PRICE BUY.PRICE MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFETY
1 Car1  exc    low    medium         low        exc    high   more       4     big   high
2 Car2 good medium    medium      medium       good    high   more       4     big medium
```

This data can be edited using common R data.frame functions. Also, DEXiR provides the method `DexiModel$alternative` for defining a single decision alternative, for example:

```
> alt <- Car$alternative("MyCar1",
         BUY.PRICE="low", MAINT.PRICE=2, X.PERS="more", X.DOORS="4",
         LUGGAGE=2, SAFETY="medium")
> alt
    name CAR PRICE BUY.PRICE MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFETY
1 MyCar1  NA    NA         3           2         NA      NA      3       3       2      2
```

Finally, such data tables can be evaluated using `DexiModel$evaluate`:

```
> eval <- Car$evaluate(alt)
> eval
    name CAR PRICE BUY.PRICE MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFETY
1 MyCar1   4     3         3           2          3       3      3       3       2      2
> Car$as_character(eval)
    name CAR PRICE BUY.PRICE MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFETY
1 MyCar1 exc   low       low      medium       good    high   more       4  medium medium
```

## Analysis of alternatives

Once defined and evaluated, alternatives can be analysed further. DEXiR provides three analysis methods:

`selective_explanation` Exposing particular weak and strong points of alternatives.

`plus_minus` **analysis** Exploring effects of changing individual attributes to evaluation results.

`compare_alternatives` Comparison of an alternative with other alternatives.

Examples:

```
> Car$selective_explanation(1)

Selective explanation of Car1

Weak points:
None

Strong points:
 id          structure         Car1
 CAR.1       +-CAR             4
 PRICE         |-PRICE         3
 MAINT.PRICE   | +-MAINT.PRICE 3
 TECH.CHAR.    +-TECH.CHAR.    4
 COMFORT         |-COMFORT     3
 X.PERS          | |-#PERS     3
 LUGGAGE         | +-LUGGAGE   3
 SAFETY          +-SAFETY      3

> Car$plus_minus(1, as_character = TRUE)
 id          structure         -2    -1    CAR.1=exc 1
 BUY.PRICE     | |-BUY.PRICE   [     unacc medium    exc
 MAINT.PRICE   | +-MAINT.PRICE unacc exc   low       ]
 X.PERS          | |-#PERS     unacc exc   more      ]
 X.DOORS         | |-#DOORS    unacc exc   4         exc
 LUGGAGE         | +-LUGGAGE   unacc exc   big       ]
 SAFETY          +-SAFETY      unacc exc   high      ]

> Car$compare_alternatives(1, as_character = TRUE)
 id          structure         Car1   Car2
 CAR         CAR               NULL   NULL
 CAR.1       +-CAR             exc    > good
 PRICE         |-PRICE         low    > medium
 BUY.PRICE     | |-BUY.PRICE   medium
 MAINT.PRICE   | +-MAINT.PRICE low    > medium
 TECH.CHAR.    +-TECH.CHAR.    exc    > good
 COMFORT         |-COMFORT     high
 X.PERS          | |-#PERS     more
 X.DOORS         | |-#DOORS    4
 LUGGAGE         | +-LUGGAGE   big
 SAFETY          +-SAFETY      high    > medium
```

## Charts

Evaluation results can be drawn on charts. DEXiR provides four charts that display multiple alternatives:

`plotalt1` with respect to a single attribute, drawing a scatterplot "alternatives by attribute-values"

`plotalt2` with respect to two attributes, drawing a scatterplot "attribute1 by attribute2"

`plotalt_parallel` with respect to multiple attributes, drawing evaluation results using parallel axes

`plotalt_radar` with respect to multiple attributes, drawing evaluation results on a radar chart

The latter two plots scale evaluation results to the `[0:1]` interval. Evaluation values represented by sets or distributions are plotted either as intervals (`aggregate = "minmax"`) or are aggregated to a single value (`aggregate = "min"`, `"max"` or `"mean"`).

Examples:

```
Plot all Car alternatives with respect to Car$first() ("CAR.1"))
> plotalt1(Car)

Plot evaluation results of all Car alternatives with respect to attribute "PRICE"
> plotalt1(Car, "PRICE")

Draw "TECH.CHAR." by "PRICE" scatterplot of all Car alternatives
> plotalt2(Car, "TECH.CHAR.", "PRICE")

Draw a "TECH.CHAR." by "PRICE" scatterplot of the second Car alternative
> plotalt2(Car, "TECH.CHAR.", "PRICE", 2)

Draw all Car alternatives on parallel axes
> plotalt_parallel(Car)

Draw all Car alternatives on a radar chart
> plotalt_radar(Car)
```

## On the use of values in DEXi models

*DEXi values* are used throughout DEXi models. They provide input values and carry results of evaluations in data frames that contain data about decision alternatives. Values are also used in definitions of DexiFunctions and are returned by `DexiFunction$evaluate` when evaluating some function for a given set of arguments.

In DEXi, values are always bound to the context provided by a DexiScale. Since each fully defined DexiAttribute is associated with some scale, we can generalize the scale context to attributes and speak about "assigning some value to an attribute".

The scale type determines the type and possible range of values that can be assigned to an attribute. DEXiR implements two scale types: DexiContinuousScale and DexiDiscreteScale. Regarding the values, the former is really simple: it allows assigning any single real number to the corresponding attribute. In other words, continuous DEXi values are of type numeric(1).

DexiDiscreteScale is the main scale type used throughout DEXi models and supports a wider range of value types.

The "normal" and most common discrete value is a "single qualitative value". For illustration, let us use the scale composed of four qualitative values: `"unacc"`, `"acc"`, `"good"`, `"exc"`. Then, "a single qualitative value" denotes one of these words. Internally in DEXiR, such values are not represented by character strings, but rather by ordinal numbers, so that ord(`"unacc"`) = 1, ord(`"acc"`) = 2, etc. Some DEXiR functions can convert between the two representations, see `DexiModel$as_character` and `set_alternative()`.

In order to cope with missing, incomplete or uncertain data, DEX extends the concept of single values to value *sets* and *distributions*. In DEXiR, wherever it is possible to use a single qualitative value, it is also possible to use a value set or distribution. This is the main reason that all DEXiR data structures related to DEXi values are represented by lists rather than plain vectors. This includes all data frames that represent decision alternatives and all functions that return qualitative values. Also note that while sets are fully implemented in the current DEXi software, distributions are not and are thus considered extensions towards the full DEX method.

A *DEXi value set* is a subset of the full range of a DexiDiscreteScale values. For the above example, the full range of ordinal values is `1:4`, and some possible subsets are `c(2)`, `c(2, 4)`, `c(1, 2, 3)` and `1:4`. Internally, sets are represented by plain integer vectors or plain numeric vectors containing integer numbers.

A *DEXi value distribution* associates each DexiDiscreteScale value with some number, generally denoted $p$ and normally expected to be in the [0,1] interval. Depending on the context and used evaluation method (see `evaluate()`), $p$ can be interpreted as *probability* or *fuzzy set membership*. In DEXiR, value distributions are represented using the S3 class "distribution" (see distribution). For example, `distribution(0.5, 0, 0.2, 0.3)` represents a value distribution over the above scale example, assigning $p = 0.5$ to `"unacc"`, $p = 0.0$ to `"acc"`, $p = 0.2$ to `"good"` and $p = 0.3$ to `"exc"`.

Remarks:

- The value `distribution(0.5, 0, 0.2, 0.3)` is internally represented as `c(0.5, 0, 0.2, 0.3)`, whose `class()` is `"distribution"`.
- Using a special class for distributions is necessary to distinguish them from sets. For instance, the notation `c(1, 1)` is ambiguous and would be interpreted differently as a set or distribution.
- Some DEXiR functions (see `DexiModel$as_character` and `set_alternative()`) support the formulation of distributions in the form of named vectors or lists, for instance `list(unacc=0.5, good=0.2, exc=0.3)`.
- In data frames that contain data about decision alternatives, numeric vectors that contain non-integer values are implicitly interpreted as distributions rather than sets.

## Examples of using value sets and distributions

First, let us consider a car for which we have no evidence about its possible maintenance costs. For the value of `MAINT.PRICE`, we may use `"*"`, which denotes the full range of the corresponding attribute values (equivalent to `1:3` or `c(1, 2, 3)` in this case). Notice how the evaluation method considers all the possible values of `MAINT.PRICE` and propagates them upwards.

```
alt <- Car$alternative("MyCar1a",
     BUY.PRICE="low", MAINT.PRICE="*", X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY=2)
Car$evaluate(alt)
   name  CAR PRICE BUY.PRICE MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFETY
1 MyCar1a 1, 4  1, 3         3     1, 2, 3          3       3      3       3       2      2
```

The above evaluation result is not really useful, as the car turns out to be `c(1, 4)`, that is, either `"unacc"` or `"exc"`, depending on maintenance costs. Thus, let us try using value distribution for MAINT.PRICE, telling DEXiR that low maintenance costs are somewhat unexpected ($p = 0.1$) and that medium costs ($p = 0.6$) are more likely than high ($p = 0.3$). Using the evaluation method `"prob"` (where $p$'s are interpreted as probabilities) gives the following results:

```
alt <- Car$alternative("MyCar1b",
       BUY.PRICE="low", MAINT.PRICE=distribution(0.1, 0.6, 0.3),
       X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY=2)
Car$evaluate(alt, method = "prob")
   name            CAR       PRICE BUY.PRICE  MAINT.PRICE TECH.CHAR. COMFORT X.PERS X.DOORS LUGGAGE SAFE
1 MyCar1b 0.1, 0.0, 0.0, 0.9 0.1, 0.0, 0.9      3 0.1, 0.6, 0.3 0, 0, 1, 0 0, 0, 1    3     3      2    2
```

In this case, the final evaluation of `CAR` is `distribution(0.1, 0.0, 0.0, 0.9)`, that is, `list(unacc=0.1, exc=0.9)`. It is much more likely that `MyCar1b` is `"exc"` than `"unacc"`.

## References

- *Decision EXpert*. Wikipedia, `https://en.wikipedia.org/wiki/Decision_EXpert`.
- Trdin, N., Bohanec, M.: Extending the multi-criteria decision making method DEX with numeric attributes, value distributions and relational models. *Central European Journal of Operations Research*, 1-24, 2018 doi:10.1007/s1010001704689.
- Bohanec, M.: DEX (Decision EXpert): A Qualitative Hierarchical Multi-criteria Method. In: Kulkarni, A.J. (ed.): *Multiple Criteria Decision Making: Techniques, Analysis and Applications*. Singapore: Springer, 39-78, 2022 doi:10.1007/9789811674143_3.
- *DEXi: A Program for Multi-Attribute Decision Making*. `https://kt.ijs.si/MarkoBohanec/dexi.html`.
- Bohanec, M.: *DEXi: Program for Multi-Attribute Decision Making, User's Manual, Version 5.04*. IJS Report DP-13100, Jožef Stefan Institute, Ljubljana, 2020. `https://kt.ijs.si/MarkoBohanec/pub/DEXiManual504.pdf`.
- Bohanec, M.: *DEXiWin: DEX Decision Modeling Software, User's Manual, Version 1.2*. IJS Report DP-14741, Jožef Stefan Institute, Ljubljana, 2024. `https://kt.ijs.si/MarkoBohanec/pub/2024_DP14747_DEXiWin.pdf`.
- *DEX Software*. `https://dex.ijs.si`.

## Author(s)

**Maintainer**: Marko Bohanec <`marko.bohanec@ijs.si`> (ORCID)

## alt_values *alt_values*

### Description

Make a list of alternative's values corresponding to attributes.

### Usage

```
alt_values(alt, attributes, as_character = TRUE, round = NULL)
```

### Arguments

| | |
|---|---|
| alt | data.frame representing a single alternative. |
| attributes | A vector of DexiAttribute objects. |
| as_character | logical(1).. Determines whether to represent alternative values numerically ("internal representation") (FALSE) or as character strings (using value_text()) (TRUE). |
| round | A single integer. An optional argument to value_text(). |

### Value

character(length(attributes)). String representation of alt's values.

### See Also

value_text()

### Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

unlist(alt_values(Car$alternatives[1,], Car$attributes, as_character = TRUE))
# c("NULL", "exc", "low", "medium", "low", "exc", "high", "more", "4", "big", "high")
```

## `and_function` *and_function*

### Description

Determine the function to be used in the conjunctive aggregation step of `evaluate()`.

### Usage

```
and_function(method = EnumEvalMethod, and = NULL)
```

### Arguments

| | |
|---|---|
| `method` | One of: `"set"` (default), `"prob"`, `"fuzzy"` or `"fuzzynorm"`. |
| `and` | Some conjunctive aggregation function of the form `function(num_vector)`, or `NULL`. |

### Value

Returns the function `and` if not `NULL`. Otherwise, it determines the result depending on `method`:

`"set"`: `function(x) 0`

`"prob"`: `prod`

`"fuzzy"`: `min`

`"fuzzynorm"`: `min`

Fails with an error if the result is not an R function.

### See Also

`evaluate`, `or_function()`.

## `attribute_effect` *attribute_effect*

### Description

Given a single `alternative`, determine the effects of varying `attribute` on `target` attribute.

### Usage

```
attribute_effect(model, attribute, alternative, target = NULL, seq = NULL, ...)
```

## Arguments

| | |
|---|---|
| `model` | A DexiModel object. Required. |
| `attribute` | A DexiAttribute with assigned discrete or continuous scale. |
| `alternative` | A `data.frame` containing a single alternative. |
| `target` | Target DexiAttribute. Defaults to `model$first()`. |
| `seq` | A sequence of `attribute`'s numeric values for which to evaluate `alternative`. For discrete scales: Must be a sequence of integers. Defaults to `attribute$scale$full_range()`. For continuous scales: `seq` is required. |
| `...` | Optional parameters passed to `evaluate_attribute()`. |

## Value

A list of `target` evaluation results, indexed by the values of `seq`.

## See Also

`evaluate_attribute()`

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

alt <- Car$alternative("MyCar",
      BUY.PRICE="low", MAINT.PRICE=2, X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY="medium")
# Determine the effect of changing "SAFETY" balues on "CAR.1"
attribute_effect(Car, "SAFETY", alt)
# Returns a list of "CAR.1" values corresponding to consecutive values of "SAFETY"
attribute_effect(Car, "LUGGAGE", alt, "TECH.CHAR.")
# Returns a list of "TECH.CHAR." values corresponding to consecutive values of "LUGGAGE"
```

---

# `att_names` *att_names*

---

## Description

Return names or IDs of DexiAttribute objects.

## Usage

```
att_names(atts, use_id = TRUE)
```

## Arguments

| | |
|---|---|
| `atts` | A vector of DexiAttributes. |
| `use_id` | Determines whether to return attribute IDs or original DEXi names. |

### Value

A character vector of attribute IDs or names.

## bounded_scale_value *bounded_scale_value*

### Description

bounded_scale_value is a wrapper around scale_value() that makes sure that the resulting values lie within the bounds set up by the scale.

### Usage

```
bounded_scale_value(value, scale)
```

### Arguments

| | |
|---|---|
| value | Any DEXi value, including value sets and distributions. |
| scale | A DexiScale or derived object. |

### Value

For continuous scales, value is returned "as is". For discrete scales, all elements of value that lie outside of scale$full_range() are removed. If this results in an empty value set or distribution, NULL is returned.

### See Also

scale_value()

### Examples

```
scl <- DexiDiscreteScale(values = c("low", "med", "high"))
bounded_scale_value(NA, scl)                              # NA
bounded_scale_value(1, scl)                               # 1
bounded_scale_value(4, scl)                               # NULL
bounded_scale_value(c(0, 1, 3,  4, 5), scl)               # c(1, 3)
bounded_scale_value(distribution(0.1, 0.2, 0.3, 0.4), scl) # distribution(0.1, 0.2, 0.3)
```

## compare_alternatives *compare_alternatives*

### Description

Compare Alternatives Analysis: Compare `alternative` with each of `alternatives`. Display only values that differ and, optionally when `compare = TRUE`, include preference-relational operators.

### Usage

```
compare_alternatives(
  model,
  alternative,
  alternatives = NULL,
  root = NULL,
  compare = TRUE,
  deep = TRUE,
  print = TRUE,
  as_character = FALSE,
  round = NULL,
  id = NULL,
  evaluate = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| `model` | A DexiModel object. Required. |
| `alternative` | Either a `data.frame` representing a single alternative or an integer index to `model$alternatives`. |
| `alternatives` | Either a `data.frame` representing one or more alternatives, or an integer numeric vector representing indices to `model$alternatives`. By default, `alternatives` are set to `model$alternatives`, possibly excluding `alternative` when indexed. |
| `root` | Optional DexiAttribute object. When specified, only attributes that affect `root` are included in the analysis. Otherwise, all `model$attributes` are included. |
| `compare` | `logical(1)`. Whether or not preference relations `"<"`, `">"`, `"<="`, `">="` are included in results. |
| `deep` | `logical(1)`. Whether of not "deep" comparison (see `compare_two_alternatives()`) is carried out. |
| `print` | `logical(1)`. When `TRUE`, pretty print (left justify) the results. |
| `as_character` | `logical(1)`. Whether to represent alternative values numerically (`FALSE`) or using text (`TRUE`). |
| `round` | An integer number, argument to `value_text()`. |

| | |
|---|---|
| `id` | `character(1)`. Determines the contents of the first or first two columns of the resulting `data.frame`:<br>`"id"` Attribute ID.<br>`"structure"` Attribute `$structure() + $name`.<br>**anything else** Equivalent to both `"id"` and `"structure"`. |
| `evaluate` | `logical(1)`. Whether or not to evaluate `alternative` and `alternatives` beforehand. |
| `...` | Optional parameters for `evaluate()`. |

## Value

Returns or prints a `data.frame` consisting of columns: `id` (if requested), `structure` (if requested), values of `alternative` and comparison results for each alternative from `alternatives`.

## See Also

`compare_two_alternatives()`, `evaluate()`

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# Extend Car$alternatives
car3 <- set_alternative(Car, Car$alternatives[2,], name = "Car3", LUGGAGE = 2)
Car$alternatives[3,] <- car3
car4 <- set_alternative(Car, Car$alternatives[2,], name = "Car4", LUGGAGE = 1)

# Compare Car1 with the other two, varying some arguments
compare_alternatives(Car, 1, evaluate=TRUE, compare=FALSE)
compare_alternatives(Car, 1, evaluate=TRUE, compare=TRUE)
compare_alternatives(Car, 1, evaluate=TRUE, compare=TRUE, deep=FALSE)

# Compare Car2 with Car1
compare_alternatives(Car, 2, 1)

# Compare car3 with Car1 and Car2
compare_alternatives(Car, car3, 1:2)

# Compare car4 with Car$alternatives
compare_alternatives(Car, car4)

# Compare Car$alternatives[1,] with car3
compare_alternatives(Car, 1, car3)
compare_alternatives(Car, Car$alternatives[1,], car3)
```

---

`compare_two_alternatives` *compare_two_alternatives*

---

## Description

Compare alternatives `alt1` and `alt2` with respect to `attributes`.

## Usage

```
compare_two_alternatives(alt1, alt2, attributes, deep = TRUE)
```

## Arguments

| | |
|---|---|
| `alt1` | `data.frame`. First alternative. |
| `alt2` | `data.frame`. Second alternative. |
| `attributes` | Vector of DexiAttribute objects. |
| `deep` | `logical(1)`. When `TRUE` and compared values are equal, input attributes are additionally investigated for possible preferential differences. |

## Value

`numeric(length(attributes))`. Each element represents the outcome of comparison w.r.t. the corresponding attribute. Possible outcomes:

`0` Values are equal.

`-1` `alt1`'s value is worse than `alt2`'s.

`+1` `alt1`'s value is better than `alt2`'s.

`NA` Values are incomparable.

When `deep = TRUE`, the so-called deep comparison is performed: when the compared attribute's values are equal, subordinate attributes are checked for differences, possibly returning `-0.5` (indicating the weak preference relation "<=") or `+0.5` (indicating the weak preference relation ">=").

## See Also

`compare_values_on_scale()`

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

compare_two_alternatives(Car$alternatives[1,], Car$alternatives[2,], Car$attributes)
# c(NA, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1)
```

## compare_values *compare_values*

### Description

Compare two DEXi values. Internal representation is assumed for `value1` and `value2`, i.e., a single number, an integer vector representing a set or `distribution()`. Distributions are compared as sets.

### Usage

```
compare_values(value1, value2)
```

### Arguments

| | |
|---|---|
| `value1` | First value. |
| `value2` | Second value. |

### Value

`0` if values are equal, `-1` if `value1` < `value2`, `+1` if `value1` > `value2` and `NA` if values are incomparable. Values are incomparable if they are of a non-DEXiValue type or if they represent two overlapping sets.

### Examples

```
compare_values(c(1,2), c(1,2))       # 0
compare_values(c(1,2), c(1,3))       # NA
compare_values(c(1,2), c(3,4))       # -1
compare_values(c(1,2), c(2,4))       # NA
compare_values(c(1,2), c(2.1,4))     # -1
compare_values(c(1,2.05), c(2.1,4))  # -1
compare_values(c(3,4), c(3,4))       # 0
compare_values(c(5,5), c(3,4))       # +1
compare_values(c(5,5), 2)            # +1
compare_values(c(5,2), 2)            # NA
compare_values(c(5,3), 2)            # +1
compare_values(distribution(5,3), 2) # NA
compare_values(distribution(5,3), 5) # -1
```

---

`compare_values_by_preference`
*compare_values_by_preference*

---

## Description

Compare values, considering preference `order`. For value arguments, see `compare_values()`.

## Usage

```
compare_values_by_preference(value1, value2, order = EnumOrder)
```

## Arguments

| | |
|---|---|
| `value1` | First value. |
| `value2` | Second value. |
| `order` | `EnumOrder`, i.e., one of the strings `"ascending"`, `"descending"`, `"none"`. |

## Value

`compare_values()` result, modified according to `order`. Results `0` (equal values) and `NA` (incomparable values) are always retained. Results `-1` and `+1` are retained when `order="ascending"` and reversed when `order="descending"`. When `order="none"`, non-equal values return `NA`.

## See Also

`compare_values()`

## Examples

```
compare_values_by_preference(1, 1, "none")       # 0
compare_values_by_preference(1, 2, "none")       # NA
compare_values_by_preference(3, 2, "none")       # NA
compare_values_by_preference(1, 1, "ascending")  # 0
compare_values_by_preference(1, 2, "ascending")  # -1
compare_values_by_preference(3, 2, "ascending")  # +1
compare_values_by_preference(1, 1, "descending") # 0
compare_values_by_preference(1, 2, "descending") # +1
compare_values_by_preference(3, 2, "descending") # -1
```

## compare_values_on_scale

*compare_values_on_scale*

### Description

Compare values value1 and value2 considering scale$order. Internal DEXi representation is assumed for values, i.e., a single number, an integer vector representing a set or distribution(). Distributions are compared as sets.

### Usage

```
compare_values_on_scale(value1, value2, scale, force_compare = FALSE)
```

### Arguments

| | |
|---|---|
| value1 | First value. |
| value2 | Second value. |
| scale | Normally a DEXiScale object or a DexiAttribute object with defined $scale. |
| force_compare | logical(1). Applies when scale is NULL or anything other than expected. When force_compare = TRUE, comparison is enforced, assuming "ascending" scale order. When force_compare = FALSE, NA is returned. |

### Value

compare_values() result, modified according to scale$order.

### See Also

compare_values(), compare_values_by_preference()

### Examples

```
compare_values_on_scale(1, 2, NULL)                       # NA
compare_values_on_scale(2, 1, "")                         # NA
compare_values_on_scale(1, 2, NULL, force_compare = TRUE) # -1
compare_values_on_scale(2, 1, "", force_compare = TRUE)   # +1

scl <- DexiDiscreteScale(values = c("a", "b", "c"))
compare_values_on_scale(1, 1, scl)             # 0
compare_values_on_scale(1, 2, scl)             # -1
compare_values_on_scale(3, 2, scl)             # +1
compare_values_on_scale(c(1, 2), c(1, 2), scl) # 0
compare_values_on_scale(c(1, 2), c(2, 3), scl) # NA

scl <- DexiDiscreteScale(order = "descending", values = c("a", "b", "c"))
compare_values_on_scale(1, 1, scl)             # 0
compare_values_on_scale(1, 2, scl)             # +1
compare_values_on_scale(3, 2, scl)             # -1
compare_values_on_scale(c(1, 2), c(1, 2), scl) # 0
compare_values_on_scale(c(1, 2), c(2, 3), scl) # NA

scl <- DexiDiscreteScale(order = "none", values = c("a", "b", "c"))
compare_values_on_scale(1, 1, scl)             # 0
compare_values_on_scale(1, 2, scl)             # NA
compare_values_on_scale(3, 2, scl)             # NA
compare_values_on_scale(c(1, 2), c(1, 2), scl) # 0
compare_values_on_scale(c(1, 2), c(2, 3), scl) # NA
```

---

## `convert_alternatives` *convert_alternatives*

---

### Description

Converts a `data.frame` of alternatives' data to another `data.frame`. The conversion generally involves: aggregating DEXi values originally represented by sets or distributions, scaling aggregated values to a given interval and/or reversing values assigned to "descending" DexiScales.

### Usage

```
convert_alternatives(
  model,
  alternatives = NULL,
  interpret = c("set", "distribution", "none"),
  aggregate = min,
  omin = 0,
  omax = 1,
  map_values = TRUE,
  reverse_descending = TRUE,
  verbatim = "name",
  skip = NULL,
  continuous = convert_data_continuous,
  discrete = convert_data_discrete
)
```

### Arguments

| | |
|---|---|
| `model` | A DexiModel object. Required. |
| `alternatives` | A `data.frame` of alternatives (normally an output of `evaluate()`) or indices to `model$alternatives`. The default value `NULL` selects `model$alternatives`. |
| `interpret` | `character(1)`. Determines how the original values in `alternatives` are interpreted, i.e., converted prior to submitting them to `aggregate()`:<br>`"set"` As a set of values. Any distribution-type value is converted to a set, thus discarding the numeric membership information. |

"distribution" As a value distribution, i.e., a numeric vector of membership values.

"none" No conversion.

Values corresponding to continuous attributes are not converted nor affected by these settings.

| | |
|---|---|
| aggregate | A function accepting the interpreted DEXi value (see interpret) and converting it to become part of the output data frame. Normally, this function is assumed to accept a numeric vector argument and aggregate it in a single numeric value. The default aggregation function is min(). Typical alternatives include max() and mean(). |
| omin | numeric(1). Lower bound of the output value interval (see map_values). Default: 0. |
| omax | numeric(1). Upper bound of the output value interval (see map_values). Default: 1. |
| map_values | logical(1). When TRUE, values produced by aggregate() are further scaled to the interval [omin:omax]. Input bounds are determined from the corresponding attribute scales (for discrete attributes) or as minimum/maximum values from alternatives (for continuous attributes). |
| reverse_descending | logical(1). Whether or not to reverse the values of attributes whose scales are of a "descending" preference order. |
| verbatim | character(). Names of alternatives' data columns that are included in the output without conversion. Default: "name". |
| skip | character(). Names of alternatives' data columns that are ignored in the process. Default: NULL. |
| continuous | A function converting a data column that corresponds to a continuous attribute. Default: convert_data_continuous(). Setting continuous to NULL excludes all continuous attributes from conversion. |
| discrete | A function converting a data column that corresponds to a discrete attribute. Default: convert_data_discrete(). Setting discrete to NULL excludes all discrete attributes from conversion. |

## Details

The rationale for convert_alternatives() is that data frames representing alternatives, particularly those produced by evaluate(), generally contain DEXi values of various and mixed data types, such as numbers and numeric vectors (sets and distributions). As such, this data is difficult to work with in R, as most R functions expect simpler and more uniform data structures. convert_alternatives() produces data frames that are more suitable for standard R data analysis and graph drawing. However, as the conversion generally involves aggregation and mapping of DEXi values, it may distort or lose information along the way.

## Value

A converted data.frame.

## See Also

`convert_data_continuous()`, `convert_data_discrete()`, `scale_alternatives()`, DEXiR-package notes on values in DEXi models.

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# Map Car$alternatives' values to the [0, 1] interval.
convert_alternatives(Car)

#  name    CAR.1 PRICE BUY.PRICE MAINT.PRICE TECH.CHAR. COMFORT X.PERS  X.DOORS LUGGAGE SAFETY
# 1 Car1 1.0000000   1.0      0.5        1.0  1.0000000      1      1 0.6666667      1    1.0
# 2 Car2 0.6666667   0.5      0.5        0.5  0.6666667      1      1 0.6666667      1    0.5
```

---

`convert_data_continuous`     *convert_data_continuous*

---

## Description

A helper function for converting individual columns of alternatives' data. It is assumed that `data` contains numeric data corresponding to a continuous DexiAttribute. During conversion, values are optionally converted from some interval to another, using `lin_map()`, and/or reversed using `reverse_value()` for scales whose `$order = "descending"`.

## Usage

```
convert_data_continuous(
  data,
  scale,
  imin = NULL,
  imax = NULL,
  omin = 0,
  omax = 1,
  map_values = TRUE,
  reverse_descending = TRUE
)
```

## Arguments

| | |
|---|---|
| `data` | A vector containing floating point numbers. Typically a `data.frame` column of DEXi alternatives' data. |
| `scale` | A DexiContinuousScale object or a continuous DexiAttribute object. |

| | |
|---|---|
| `imin` | Lower input bound. Default: determined as `min(data)`. |
| `imax` | Upper input bound. Default: determined as `max(data)`. |
| `omin` | Lower output bound for `lin_map()` value scaling. |
| `omax` | Upper output bound for `lin_map()` value scaling. |
| `map_values` | `logical(1)`. Whether or not to perform value scaling using `lin_map()`. |
| `reverse_descending` | `logical(1)`. Whether or not to reverse values of a "descending" scale. |

## Value

`numeric()`. Vector of converted values.

## See Also

`lin_map()`, `reverse_value()`

## Examples

```
scl <- DexiContinuousScale()
convert_data_continuous(c(1, 2, 5), scl) # c(0.0, 0.25, 1.00)
convert_data_continuous(c(1, 2, 5), scl, imin = 0, imax = 10, omin = 0, omax = 100)
# c(10, 20, 50)
```

---

## `convert_data_discrete` *convert_data_discrete*

---

## Description

#' A helper function for converting individual columns of alternatives' data. It is assumed that `data` contains data corresponding to a discrete DexiAttribute. During conversion, `data` elements are converted either to sets or distributions, and function `aggregate` if applied on them. When `interpret = "set"`, values are also optionally converted to the interval `[omin:omax]`, and reversed using `reverse_value()` for scales whose `$order = "descending"`.

## Usage

```
convert_data_discrete(
  data,
  scale,
  interpret = c("set", "distribution", "none"),
  aggregate = min,
  omin = 0,
  omax = 1,
  map_values = TRUE,
  reverse_descending = TRUE
)
```

## Arguments

| | |
|---|---|
| `data` | A vector containing DEXi values: single numbers, integer vectors or distribuions. Typically a `data.frame` column of DEXi alternatives' data. |
| `scale` | A DexiDiscreteScale object or a discrete DexiAttribute object. |
| `interpret` | Either `"set"` (default), `"distribution"` or `"none"`. Determines how are individual `data` elements interpreted: as sets or distributions. Actually, each element is converted either to a set or distribution prior do applying `aggregate()`. When `interpret = "none"`, just `aggregate()` is applied on the original `value` from `data`, without any value scaling or reversal. |
| `aggregate` | A function applied on each interpreted `data` element. Normally a function that maps a numeric vector (set or distribution) to a single number. Default: `min()`. |
| `omin` | Lower output bound for `lin_map()` value scaling. Applies only when `interpret = "set"`. |
| `omax` | Upper output bound for `lin_map()` value scaling Applies only when `interpret = "set"`. |
| `map_values` | `logical(1)`. Whether or not to perform value scaling using `lin_map()`. Applies only when `interpret = "set"`. |
| `reverse_descending` | `logical(1)`. Whether or not to reverse values of a "descending" scale. |

## Value

Vector of converted values.

## See Also

`lin_map()`, `reverse_value()`

## Examples

```
scla <- DexiDiscreteScale(values = c("L", "M", "H"))
scld <- DexiDiscreteScale(values = c("L", "M", "H"), order = "descending")
convert_data_discrete(c(1, 2, 3), scla)    # 0.0 0.5 1.0
convert_data_discrete(c(1, 2, 3), scld)    # 1.0 0.5 0.0
convert_data_discrete(list(1, 2, 3), scla) # 0.0 0.5 1.0
convert_data_discrete(list(1, 2, 3), scld) # 1.0 0.5 0.0
convert_data_discrete(list(1, 2, 3), scld, omax=10) # 10  5  0
data <- list(1, c(1,2), distribution(0.2, 0, 0.8), NA)
convert_data_discrete(data, scla, omax=10) #  0  0  0 NA
convert_data_discrete(data, scld, omax=10) # 10 10 10 NA
convert_data_discrete(data, scla, aggregate=max, omax=10)  #  0  5 10 NA
convert_data_discrete(data, scla, aggregate=mean, omax=10) # 0.0 2.5 5.0  NA
```

## `default_quality` *default_quality*

### Description

Make a default discrete scale quality vector depending on the scale's `order` and `nvals`.

### Usage

```
default_quality(order = EnumOrder, nvals)
```

### Arguments

| | |
|---|---|
| `order` | ‘character(1)1, one of "ascending", "descending" or "none". |
| `nvals` | `integer(1)`. The number of qualitative values of considered DexiDiscreteScale. |

### Value

character vector of length `nvals`, containing `"bad"`, `"none"` or `"good"`.

### Examples

```
default_quality("ascending", 5)
default_quality("descending", 5)
default_quality("none", 5)
default_quality("ascending", 2)
default_quality("ascending", 1)
```

## `DexiAttribute-class` *DexiAttribute*

### Description

`DexiAttribute` is a RC class representing a DEXi attribute in R.

### Details

In a DEXi model, attributes are variables that represent observed properties of decision alternatives. Attributes are structured in a tree, so each attribute may, but need not, have one or more direct descendants (lower-level attributes) in the tree. Attributes without descendants are called *basic* and serve as model inputs. Attributes with one or more descendants are called *aggregate* and represent model outputs. In order to represent attribute hierarchies rather than plain trees, some attributes may be *linked*: two attributes of which one links to another one collectively represent, in a conceptual sense, a single attribute in the hierarchy.

When completely defined, each attribute is associated with a value scale represented by a DexiScale object. An object DexiFunction is also defined for each aggregate attribute, aimed at defining the aggregation of the attribute's inputs to values of that attribute.

## Fields

`name` character. Name of the attribute as defined in the original DEXi model. Notice that such names may not be unique and may contain characters that cannot be used for variable names in R.

`id` character. A unique identification of the attribute in the model. Derived from `name` so that it can be used as a variable name in R.

`description` character. An optional textual description of the attribute.

`inputs` list of DexiAttributes. A list of immediate descendants of this attribute in the tree/hierarchy. `NULL` for basic attributes.

`link` DexiAttribute. `NULL` or a link to another DexiAttribute

`scale` DexiScale. Value scale associated with this attribute, or `NULL`.

`funct` DexiFunction. Aggregation function associated with this attribute, or `NULL`.

`parent` DexiAttribute or DexiModel (only for `DexiModel$root`). Parent attribute of this attribute in the tree/hierarchy. The `DexiModel$root`'s parent is the DexiModel, which contains all those attributes.

`.alternatives` list. An internal field providing temporary storage for names or values of alternatives while reading them from a `.dxi` file.

## Methods

`affects(ant)` `ant` (as "antecedent") is some `DexiAttribute`. The function returns `TRUE` if `ant` lies on the path leading from this attribute towards the root, and is therefore affected by this attribute.

`count()` Return the number of `inputs` of this attribute.

`dim()` Dimensions of the value space determined by this attribute's `inputs`. Result: a numeric vector of length equal to `ninp()`, containing `DexiScale$count()` of all descendant attributes, or `NA` for attributes without associated scales. For basic attributes, `dim()` returns `NULL`.

`initialize( name = "", description = "", inputs = list(), id = "", link = NULL, scale = NULL, funct = NULL, pa` Initialize a `DexiAttribute` object.

`inp_index(inp)` Return the index of attribute `inp` in `inputs` of this attribute.

`is_aggregate()` Logical: `TRUE` for aggregate attributes (attributes whose `ninp() > 0`).

`is_basic(include_linked = TRUE)` Logical: `TRUE` for basic attributes (attributes whose `ninp() == 0`. `include_linked` determines whether linked attributes are counted as basic (`TRUE`) or not (`FALSE`).

`is_continuous()` Logical: Indicates whether or not this is a continuous attribute.

`is_discrete()` Logical: Indicates whether or not this is a discrete attribute.

`is_link()` Logical: Indicates whether or not this is a linked attribute.

`level()` Return the level of this attribute in the hierarchy. The level of `DexiModel$root` is 0.

`model()` Return the `DexiModel` that contains this attribute.

`ninp()` Return the number of `inputs` of this attribute.

`structure()` Make an indentation string for this attribute, used for printing it in `show()`.

`tree_indent(none = " ", thru = "|", link = "*", last = "+", line = "-")` Construct a string for representing the indentation of this attribute in the model structure. The arguments none, `thru`, `link`, `last` and `line` are character strings to be used in the construction.

`verify()` Check the correctnes of a `DexiAttribute` object and its fields. Result: `error()` or `TRUE`.

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# For example, consider attribute PRICE
att <- Car$attrib("PRICE")

# Print fields and basic properties of att
att$verify()
att$name
att$id
att$description
att_names(att$inputs)
att$link
att$scale
att$funct
att_names(att$parent)
att$is_aggregate()
att$is_basic()
att$is_link()
att$level()
att$count()
att$ninp()
att$dim()
att$model()
att$structure()

# Check if att affects attribute CAR
att$affects(Car$attrib("CAR"))

# Find the index of other attributes in att's inputs
att$inp_index(Car$attrib("MAINT.PRICE"))
att$inp_index(Car$attrib("CAR"))
```

---

`DexiContinuousScale-class` *DexiContinuousScale*

---

## Description

`DexiContinuousScale` is a RC class, derived from `DexiScale`, representing continuous value scales in R.

## Details

An attribute associated with a continuous scale can take any single numeric value from `[-Inf, +Inf]`.

`DexiContinuousScale` defines two numeric bounds, called `low_point` and `high_point`, such that `low_point <= high_point`. These values partition preferentially ordered scales in three preferential classes ("qualities"): `"bad"`, `"none"` (in the sense of `"neutral"`), and `"good"`. For a scale with `order = "ascending"`, the three corresponding intervals are `[-Inf, low_point]`, `(low_point, high_point)` and `[high_point, +Inf]`. For `order = "descending"`, the order of qualities is reversed. Scales with `order = "none"` have only one associated quality, `"none"`, for the whole range of values.

Continuous scales are supported in DEXi Suite software (DEXiWin), but not in older DEXi Classic software (DEXi).

## Fields

`low_point` numeric. A bound for the quality interval `[-Inf, low_point]`.

`high_point` numeric. A bound for the quality interval `[high_point, +Inf]`.

## Methods

`count()` Return the number of scale elements. Equal to `NA` for `DexiScale`, `0` for `DexiContinuousScale`, and equal to `nvals >= 0` for `DexiDiscreteScale`.

`equal(scl)` Check if this scale is equal to scale `scl`. Needed for attribute linking.

`initialize(order = EnumOrder, ...)` Initialize a `DexiScale` object.

`to_string()` Return a string representation of this scale for printing.

`value_quality(value)` Return the quality (preferential class) of `value` on this scale: one of the strings `"bad"`, `"none"` or `"good"`. Always `"none"` for `DexiScale` and scales with `order = "none"`.

`verify()` Check the correctnes of this scale object and its fields. Result: `error()` or `TRUE`.

---

`DexiDiscreteScale-class` *DexiDiscreteScale*

---

## Description

`DexiDiscreteScale` is a RC class, derived from DexiScale, representing qualitative (symbolic, discrete, verbal) value scales in R. Such scales are typical for DEXi models and are the only scale type supported by the DEXi software. DEXiWin software supports both continuous and discrete scales.

## Details

An attribute associated with a discrete scale can take values from a finite (and usually small) set of string values contained in the character vector `values`. Additionally, each of these values is associated with one of the qualities `"bad"`, `"none"` or `"good"`. The latter are contained in the character vector `quality`, which is of the same length as `values`.

## Fields

`values` character. Vector of qualitative scale values. Example: `scale$values <- c("low", "medium", "high")`.

`nvals` integer. Equal to `length(values)`.

`quality` character. Vector of qualities, corresponding to `values`. Should be the of the same length as `values`. Example: `scale$quality <- c("bad", "none", "good")`.

`descriptions` character. A vector of textual descriptions of the corresponding `values`. Should be of the same length as `values`.

## Methods

`count()` Return the number of scale elements. Equal to `NA` for `DexiScale`, `0` for `DexiContinuousScale`, and equal to `nvals >= 0` for `DexiDiscreteScale`.

`equal(scl)` Check if this scale is equal to scale `scl`. Needed for attribute linking.

`full_range()` Return the vector that represents the full range of values on this scale. Equal to `NA` for `DexiScale` and `DexiContinuousScale`, and `1 : scale$nvals` for `DexiDiscreteScale`.

`initialize(order = EnumOrder, ...)` Initialize a `DexiScale` object.

`is_discrete()` Logical: Is this scale discrete?

`to_string()` Return a string representation of this scale for printing.

`value_index(value)` Find the index of `value` (character(1)) on this scale. Equal to `NA` for `DexiScale` and `DexiContinuousScale`. With `DexiDiscreteScale` objects, it returns a numeric index or `NA` of `value` in `scale$values`.

`value_quality(value)` Return the quality (preferential class) of `value` on this scale: one of the strings `"bad"`, `"none"` or `"good"`. Always `"none"` for `DexiScale` and scales with `order = "none"`.

`verify()` Check the correctnes of this scale object and its fields. Result: `error()` or `TRUE`.

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# For example, consider the scale of attribute PRICE
scl <- Car$attrib("PRICE")$scale

# Print fields and basic properties of scl
scl$verify()
scl$values
scl$quality
scl$descriptions
scl$nvals
scl$count()
scl$is_discrete()
scl$is_continuous()
scl$to_string()
```

```
scl$full_range()

# Find value indices
scl$value_index("medium")
scl$value_index("med")

# Is scl equal to the scale of BUY.PRICE?
scl$equal(Car$attrib("PRICE")$scale)
```

## DexiDiscretizeFunction-class
*DexiDiscretizeFunction*

### Description

`DexiDiscretizeFunction` is a RC class, derived from DexiFunction. Functions of this type discretize numerical values of continuous attributes to qualitative values of discrete attributes. More precisely, a `DexiDiscretizeFunction` can be defined only for a discrete attribute that has exactly one continuous input. Then, the function discretizes numeric values of the input attribute and maps them to discrete values of the parent attribute.

### Details

Objects of class `DexiDiscretizeFunction` define discretization rules in terms of three lists: `values`, `bounds` and `assoc`. Using `n <- nvals()` to denote the length of `values`, the required lengths of `bounds` and `assoc` are `n - 1`.

The list bounds refers to values of the input attribute and partitions its scale in `n` intervals `[-Inf, bound[[1]]]`, `[bound[[1]], bound[[2]]]`, ..., `[bound[[n - 1]]], +Inf]`. The list `values` then defines the output values for each interval. The list `assoc` contains strings `"up"` or `"down"` that indicate to which interval, lower or higher, belong the corresponding `bounds`.

### Fields

`attribute` DexiAttribute. The attribute this function is associated with. Requirements: `attribute` must be discrete (i.e., associated with a DexiDiscreteScale) and must have exactly one continuous input attribute (i.e., associated with a DexiContinuousScale).

`values` A list of output values corresponding to each interval defined by `bounds`. List elements are in general value sets, i.e., integer vectors of value indices w.r.t. `attribute$scale`.

`bounds` A vector of numeric values that partitions the input scale in intervals.

`assoc` A vector of strings `"up"` or `"down"`. For each `i in 1:n-1`, `assoc[[i]]` indicates how to map the value of `bounds[[i]]`: to `value[[i]]` (`"down"`) or `value[[i + 1]]` (`"up"`).

## Methods

`bound_assoc(idx, default = "down")` Given `idx`, a bounds index, return the corresponing association ("down" or "up").

`evaluate(x)` A silent wrapper around `value(x)`; it returns `NULL` when `value(x)` fails with an error.

`nargs()` Return the number of function arguments.

`nvals()` Return the length of `values`.

`to_string()` Return an informative string about this function's `values` and `bounds`.

`value(x)` Return the function value for arguments `x`, where arguments are a numeric vector of length equal to `att$inputs`. Additionally, arguments of a `DexiTabularFunctions$value()` must be integer numbers, and the argument of `DexiDiscretizeFunctions$value()` must be a single number.

`verify()` Check the correctnes of this function object and its fields. Result: `error()` or `TRUE`.

## Examples

```
# Create a DexiDiscretizeFunction (without association to any attributes or scales)
fnc <- DexiDiscretizeFunction(bounds = c(-1, 2), values = list(1, 3, 5), assoc = c("up", "down"))

# Print fields and basic properties of fnc

fnc$verify()
fnc$nargs()
fnc$nvals()
fnc$to_string()

fnc$bound_assoc(1)
fnc$bound_assoc(2)

# Try some discretizations
sapply(c(-1.1, -1, 0, 1, 2, 3), fnc$evaluate)
```

---

# DexiFunction-class *DexiFunction*

---

## Description

`DexiFunction` is a base RC class for representing DEXi aggregation and discretization functions in R.

## Details

DEXi functions are generally associated with aggregate attributes. For some aggregate attribute `att`, `att$funct` defines the mapping from values of `att$inputs` to values of `att`.

DexiFunction is a base class that defines fields and methods common to all functions:

- method `value(x)`: returns the function value for arguments `x`. Arguments are assumed to be a numeric vector of length equal to `att$inputs`.
- method `evaluate(x)` is a silent wrapper around `value(x)`; it returns `NULL` when `value(x)` fails with an error.

DEXiR implements two other function classes derived from `DexiFunction`: DexiTabularFunction and DexiDiscretizeFunction.

## Methods

`evaluate(x)` A silent wrapper around `value(x)`; it returns `NULL` when `value(x)` fails with an error.

`value(x)` Return the function value for arguments `x`, where arguments are a numeric vector of length equal to `att$inputs`. Additionally, arguments of a `DexiTabularFunctions$value()` must be integer numbers, and the argument of `DexiDiscretizeFunctions$value()` must be a single number.

`verify()` Check the correctnes of this function object and its fields. Result: `error()` or `TRUE`.

---

`DexiModel-class` *DexiModel*

---

## Description

`DexiModel` is a RC class representing a DEXi model in R.

## Details

Normally, `DexiModel` objects are created by reading from a `.dxi` file, previously developed by the DEXi software. In principle, all fields of a `DexiModel` should be considered read-only. DEXiR does not provide any explicit functionality for creating and changing DEXi models in R. Of course, models can still be created and modified in R, but without integrity and consistency guarantees.

## Fields

`name` character. Name of the model.

`description` character. An optional textual description of the model.

`linking` logical. Indicates whether or not the model uses linked attributes, which are used in DEXi to represent hierarchies of attributes (i.e., directed acyclic graphs) rather than trees.

`root` DexiAttribute. The virtual root of all subtrees/hierarchies of attributes in the model.

`attributes` list. A list of all DexiAttributes that constitute the model.

`att_names` character. A list of all attribute names, as defined in the original DEXi model. Notice that these names may contain whitespace and other "strange" characters, and may not be unique.

`att_ids` character. A list of unique attribute IDs generated by DEXiR from `att_names` using `make.unique`. When using the DEXiR package, it is strongly advised to refer to attributes with their IDs rather than DEXi names.

`basic` list. A list of all basic (input) DexiAttributes in the model.

`aggregate` list. A list of all aggregate (output) DexiAttributes in the model.

`links` list. A list of all linked DexiAttributes in the model.

`basic_ids` character. A vector of all basic attributes' unique names.

`aggregate_ids` character. A vector of all aggregate attributes' unique names.

`link_ids` character. A vector of all linked attributes' unique names.

`alternatives` data.frame. A data frame representing decision alternatives contained in the `.dxi` file.

## Methods

`alternative(name = "NewAlternative", ...)` Create a data frame containing data of one decision alternative. `name`, character(1), represents the alternative's name. The arguments `...` define the alternative's values to be put in the data frame. Please see `set_alternative` for the syntax of `...`.

`as_character(alt, transpose = FALSE, structure = FALSE, round = NULL)` The argument `alt` is assumed to be a data frame containing data of one or more decision alternatives with values represented by numeric vectors. `as_character(alt)` transforms the values of `alt` into a more human-readable form using character strings. Additionally, `transpose = TRUE` transposes the data frame, so that rows correspod to attributes and columns to alternatives. `structure = TRUE` additionally displays the tree structure of attributes; the latter works only with `transpose = TRUE`. `round` denotes the number of decimal digits for printing numeric values.

`att_index(atts, use_id = TRUE)` Find the indices of attributes. `atts` is a character vector of attribute IDs (when `use_id = TRUE`) or original DEXi attribute names (when `use_id = FALSE`). Result: a numeric vector containing the set of indices. Example: `Car$att_index(c("PRICE", "TECH.CHAR."))`

`att_stat()` Count the number of all attributes (including the virtual root), as well as the number of basic, aggregate and linked attributes in the model. Result: a list of the form list(all=..., basic=..., aggregate=..., link=...).

`attrib(atts)` A general function for finding attributes in the model. `atts` is a vector or list of `DexiAttributes`, attribute indices (integer) or attribute IDs (character). Result: a list of found `DexiAttributes` (or `NA`s if not found). Example: `Car$attrib(list(5, "PRICE", "TECH.CHAR."))`

`compare_alternatives(...)` Calls `compare_alternatives(.self, ...)` to carry out Comparison of Alternatives. Please see `compare_alternatives` for the description of `...` arguments.

`convert(...)` Calls `convert_alternatives(.self, ...)` to convert decision alternatives' data. Please see `convert_alternatives` for the description of `...` arguments.

`evaluate(...)` Calls `evaluate(.self, ...)` to evaluate decision alternatives. Please see `evaluate` for the description of `...` arguments.

`first()` Return first non-virtual model attribute, i.e., first descendant of model$root.

`initialize(name = "", description = "", root = NULL, linking = FALSE, ...)` Initialize a `DexiModel` object.

`link_attributes()` Carries out the linking of attributes. DEXi attributes that have the same names and value scales, and satisfy some other constraints to prevent making cycles in the model, are linked together so that they logically represent a single attribute. In this way, a tree of attributes is conceptually turned in a hierarchy (directed acyclic graph). If `linking` = `TRUE`, `link_attributes` is called by `setup()` after reading the model.

`plus_minus(...)` Calls `plus_minus(.self, ...)` to carry out Plus-Minus Analysis. Please see `plus_minus` for the description of `...` arguments.

`scale(atts)` Find attribute scales. `atts` is a vector of `DexiAttributes`. Result: a vector of the corresponding `DexiScale`s (or `NA`s).

`selective_explanation(...)` Calls `selective_explanation(.self, ...)` to carry out Selective Explanation. Please see `selective_explanation` for the description of `...` arguments.

`setup()` Called by `initialize()` as the last step that establishes consistent internal data structures by making unique attribute IDs, linking attributes (if required), making lists of attributes and their IDs, and creating a data frame of alternatives.

`verify()` Check the correctnes of a `DexiModel` object and its fields. Result: `error()` or `TRUE`.

## See Also

`evaluate`, `set_alternative`, `read_dexi()`

## Examples

```
# Get ".dxi" file name
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")

# Read DEXi model
Car <- read_dexi(CarDxi)

# Print fields of Car
Car
Car$verify()
Car$name
Car$description
Car$linking
att_names(Car$attributes)
Car$att_names
Car$att_ids
Car$basic_ids
Car$aggregate_ids
Car$att_stat()
Car$scale(Car$aggregate)

# Find some attributes in the model
Car$first()
Car$attributes[[3]]
Car$attrib("PRICE")
Car$att_index("PRICE")

# Display alternatives loaded from "Car.dxi"
Car$alternatives
```

```
Car$as_character(Car$alternatives)
Car$as_character(Car$alternatives, transpose = TRUE)
Car$as_character(Car$alternatives, transpose = TRUE, structure = TRUE)

# Define and evaluate a decision alternative (some car)
alt <- Car$alternative("MyCar",
        BUY.PRICE="low", MAINT.PRICE=2, X.PERS=3, X.DOORS=3, LUGGAGE="medium", SAFETY=2)
Car$evaluate(alt)
Car$as_character(Car$evaluate(alt))

# Employ the set-based evaluation (notice how the value of SAFETY propagates upwards to TECH.CHAR.)
alt <- Car$alternative("MyCar",
      BUY.PRICE="low", MAINT.PRICE=2, X.PERS=3, X.DOORS=3, LUGGAGE="medium", SAFETY=c(2,3))
Car$evaluate(alt)
Car$as_character(Car$evaluate(alt))

# Analysis of alternatives
Car$selective_explanation(1)
Car$selective_explanation(alt)
Car$plus_minus(alt)
Car$compare_alternatives(alt)
Car$compare_alternatives(1, 2)
Car$compare_alternatives(1, alt)
```

## DexiScale-class *DexiScale*

### Description

DexiScale is a base RC class representing value scales in R.

### Details

A value scale defines the type and set of values that can be assigned to some DexiAttribute. DexiScale is a base scale class that defines fields and methods common to all scales:

- whether or not the scale is preferentially ordered (and in which direction),
- scale type (discrete or continuous),
- the number of scale elements, if countable,
- partition of scale elements in three preferential classes: "bad", "good" and "none",
- helper methods value_index() and full_range().

DEXiR implements two other scale classes derived from DexiScale: DexiContinuousScale and DexiDiscreteScale.

### Fields

order character. Preferential order of the scale. Possible values: "ascending", "descending" or "none".

## Methods

`count()` Return the number of scale elements. Equal to `NA` for `DexiScale`, `0` for `DexiContinuousScale`, and equal to `nvals >= 0` for `DexiDiscreteScale`.

`equal(scl)` Check if this scale is equal to scale `scl`. Needed for attribute linking.

`full_range()` Return the vector that represents the full range of values on this scale. Equal to `NA` for `DexiScale` and `DexiContinuousScale`, and `1 : scale$nvals` for `DexiDiscreteScale`.

`initialize(order = EnumOrder, ...)` Initialize a `DexiScale` object.

`is_continuous()` Logical: Is this scale continuos?

`is_discrete()` Logical: Is this scale discrete?

`to_string()` Return a string representation of this scale for printing.

`value_index(value)` Find the index of `value` (character(1)) on this scale. Equal to `NA` for `DexiScale` and `DexiContinuousScale`. With `DexiDiscreteScale` objects, it returns a numeric index or `NA` of `value` in `scale$values`.

`value_quality(value)` Return the quality (preferential class) of `value` on this scale: one of the strings `"bad"`, `"none"` or `"good"`. Always `"none"` for `DexiScale` and scales with `order = "none"`.

`verify()` Check the correctnes of this scale object and its fields. Result: `error()` or `TRUE`.

---

`DexiTabularFunction-class` *DexiTabularFunction*

---

## Description

`DexiTabularFunction` is a RC class, derived from DexiFunction. Functions of this type aggregate attribute values according to *decision rules*, defined in terms of a *decision table*.

## Details

A decision table contains as many decision rules as there are possible combinations of input attributes' values. For instance, if some `attribute` has two inputs whose discrete scales have three and four values, respectively (i.e., `attribute$dim() == c(3,4)`), then the number of rules is equal to `prod(attribute$dim()) == 12`. Each rule defines the value of `attribute` for one of the possible combinations of values of `attribute$inputs`. Thus, a decision table can be interpreted as a lookup table that, given a vector of values of `attribute$inputs` (i.e., function arguments) returns the corresponding `attribute` value.

Objects of class `DexiTabularFunction` store decision rules in `values`, a multi-dimensional list that contains rule values. In most cases, a rule value is a single integer, representing an ordinal number of some value from `attribute$scale`. In a general case, however, a rule value can be an integer vector, representing a (sub)set of values from `attribute$scale`.

## Fields

`attribute` DexiAttribute. The attribute this function is associated with. Both the attribute and its inputs are required to be discrete (i.e., associated with `DexiDiscreteScales`).

`values` A multi-dimensional list of rule values. The dimensions of the list are equal to `attribute$dim()`, and the length of the list is `nvals()` == `prod(dim)`. The list contains rule values that are in general value sets, i.e., integer vectors of value indices w.r.t. `attribute$scale`.

`args` A list of integer vectors, containing all possible combinations of values of `attribute$inputs`. `args` and `values` are of the same length and ordered so that, for each `i`, `args[[i]]` defines function arguments that map to `values[[i]]`).

## Methods

`evaluate(x)` A silent wrapper around `value(x)`; it returns `NULL` when `value(x)` fails with an error.

`nargs()` Return the number of function arguments.

`nvals()` Return the function size (number of rules).

`to_string()` Return a short informative string about the size and dimensions of `values`.

`value(x)` Return the function value for arguments `x`, where arguments are a numeric vector of length equal to `att$inputs`. Additionally, arguments of a `DexiTabularFunctions$value()` must be integer numbers, and the argument of `DexiDiscretizeFunctions$value()` must be a single number.

`verify()` Check the correctnes of this function object and its fields. Result: `error()` or `TRUE`.

## See Also

`dexi_index()`, `dexi_table()`, `make_args()`

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# For example, consider the function of attribute CAR
fnc <- Car$attrib("CAR")$funct

# Print fields and basic properties of fnc
fnc$verify()
att_names(fnc$attribute)
fnc$values
fnc$args
fnc$nargs()
fnc$nvals()
fnc$to_string()

# Try some args to value mappings
fnc$evaluate(c(1, 1))
fnc$evaluate(c(2, 2))
```

```
fnc$evaluate(c(3, 4))
fnc$evaluate(c(4, 4)) # the first argument is out of bounds, returns NULL
```

## dexi_bool *dexi_bool*

### Description

Convert a DEXi string to logical. `"TRUE"`, `"T"` and `"1"` are interpreted as `TRUE`, all other strings as `FALSE`.

### Usage

```
dexi_bool(x)
```

### Arguments

| | |
|---|---|
| `x` | character(1). |

### Value

`logical(1)`.

### Examples

```
dexi_bool("TRUE")
sapply(c("TRUE", "T", "1", TRUE, 1, "FALSE", "F", "0", NULL, NA, NaN), dexi_bool)
```

## dexi_index *dexi_index*

### Description

Return the index of argument vector `vec` in the decision space `dim`. The index is calculated according to DEXi's sorting rules, which are different to R's.

### Usage

```
dexi_index(vec, dim)
```

### Arguments

| | |
|---|---|
| `vec` | Integer vector, representing arguments of some decision rule. |
| `dim` | Integer vector, representing dimensions of the corresponding decision space. Assumptions: `length(vec) == length(dim)` and, for each `i`, `1 <= vec[[i]] <= dim[[i]]`. |

## Value

Integer, index of vec.

## Examples

```
dexi_index(c(1,1,1), c(2,2,3))
dexi_index(c(1,1,2), c(2,2,3))
dexi_index(c(1,2,3), c(2,2,3))
```

---

## dexi_option_value *dexi_option_value*

---

## Description

Conversion of a string to a "DEXi value" (see DEXiR-package) according to "old" DEXi syntax. In .dxi files, the old syntax is used with OPTION XML tags. The reason for replacing the old with the new syntax (see dexi_value()) was that the old syntax can not unambiguously represent value distributions.

## Usage

```
dexi_option_value(x)
```

## Arguments

| | |
|---|---|
| x | character(1). Contains a sequence of characters, each of which represents an individual ordinal number. |

## Value

A numeric vector. The conversion uses rule_values(x, add = 1). For special-type parameters, the conversion results are:

```
x                              result
------------------------------+------
NULL                           NULL
a non-character object         NA
"" or "*"                      "*"
a string starting with "undef" NA
```

## See Also

DEXiR-package, dexi_value(), rule_value()

### Examples

```
dexi_option_value(NULL)
dexi_option_value(NA)
dexi_option_value("")
dexi_option_value("*")
dexi_option_value("undef")
dexi_option_value("1")
dexi_option_value("012")
```

---

## dexi_table *dexi_table*

---

### Description

Create a representation of DEXi's decision table in R.

### Usage

```
dexi_table(dim, low, high = NULL)
```

### Arguments

| | |
|---|---|
| dim | An integer vector, representing dimensions of the underlying decision space. |
| low | character(1). A string normally read from a .dxi file, representing the lower bounds of the corresponding decision rule values (assuming the order according to dexi_index()). Notice that the string contains zero-based characters, which are converted to one-based integer values used in R. |
| high | character(1) or NULL. A string representing the upper bounds of corresponding decision rule values. If high = NULL, high is assumed to be equal to low. |

### Value

length(dim)-dimensional matrix of rule values, which are normally single integer values, but might also be sets of values. Each set is represented by a numeric vector.

### Examples

```
# Converting DEXi's value strings to R's numeric vectors.
dexi_table(c(2, 3), "011012")
dexi_table(c(2, 3), "011012", "012112")
```

## dexi_value *dexi_value*

### Description

Conversion of a string to a "DEXi value" (see DEXiR-package) according to "new" DEXi syntax rules. In `.dxi` files, this syntax is used in `ALTERNATIVE` and `RULE` XML tags. Examples of possible options include:

```
x                              result
------------------------------+--------------------------------------------------
NULL or ""                     NULL
"*"                            "*"
a string starting with "undef" NA
"2"                            a single ordinal value, c(2) in this case
"2.1"                          a single number, c(2.1) in this case
"1:3"                          interval, equivalent to c(1, 2, 3)
"{0;2;3}"                      a value set, equivalent to c(0, 2, 3)
"<0;0.3;0.7>"                a value distribution, distribution(0.0, 0.3, 0.7)
```

### Usage

```
dexi_value(x, add = 0)
```

### Arguments

| | |
|---|---|
| `x` | character(1). |
| `add` | A numeric constant to be added to the result. Useful when converting DEXi's zero-based representation to one-based representation used in R, which requires the setting `add = 1`. |

### Value

A single integer or real number, an integer numeric vector, or a distribution.

### See Also

DEXiR-package, `dexi_option_value()`, distribution

### Examples

```
dexi_value("")
dexi_value(NULL)
dexi_value("*")
dexi_value("UNDEF")
dexi_value("2")
dexi_value("2.1")
dexi_value("1:3")
dexi_value("{0;2;3}")
dexi_value("{0;2;3}", add = 1)
dexi_value("<0;0.3;0.7>")
```

---

## `dexi_vector` *dexi_vector*

---

### Description

Interpret a string, composed of `";"`-separated numbers, as a numeric vector.

### Usage

```
dexi_vector(x)
```

### Arguments

| | |
|---|---|
| `x` | `character(1)`. |

### Value

Numeric vector.

### Examples

```
dexi_vector("1;2")
dexi_vector("1.2; 2.3")
```

---

## `distribution` *distribution*

---

### Description

Create an object as a S3 class `distribution`.

### Usage

```
distribution(...)
```

### Arguments

| | |
|---|---|
| `...` | Expected a comma-separated list of numeric values. |

### Value

An object, call it `obj`, such that `all(obj == c(...))` and `class(obj) == "distribution"`.

## See Also

DEXiR-package, `set_to_distr()`, `distr_to_set()`

## Examples

```
distribution(0.1, 0.2, 0.7)
```

---

## `distr_to_set` *distr_to_set*

---

## Description

Convert a DEXi value distribution to a DEXi value set.

## Usage

```
distr_to_set(distr, eps = .Machine$double.eps)
```

## Arguments

| | |
|---|---|
| `distr` | An S3 object of class `distribution`. |
| `eps` | A numeric value representing the threshold value of $p$ (see DEXiR-package) above which the corresponding elements are considered set members. |

## Value

A numeric vector determined as `which(distr > eps)`. Notice that `distr_to_set` is generally a lossy conversion, so that multiple different `distr`s are converted to the same sets.

## See Also

DEXiR-package, distribution, `set_to_distr()`

## Examples

```
distr_to_set(distribution(0.2, 0, 0.5, 0.3))
distr_to_set(distribution(0.1, 0, 0.7, 0.2))
distr_to_set(distribution(0.1, 0, 0.7, 0.2), eps = 0.5)
```

## `equal_scales` *equal_scales*

### Description

Check if two scales are equal. `NULL` arguments, indicating undefined scales, are allowed. Two `NULL` scales are considered equal.

### Usage

```
equal_scales(scl1, scl2)
```

### Arguments

| | |
|---|---|
| `scl1` | A DexiScale (or derived) object, or `NULL`. |
| `scl2` | A DexiScale (or derived) object, or `NULL`. |

### Value

`logical(1)`.

## `evaluate` *evaluate*

### Description

Evaluates decision alternatives. Essentially, this is a bottom-up aggregation method: starting with basic attributes (or `pruned` aggregate attributes), values of each alternative are gradually aggregated towards the `root` attribute, according to `evaluation_order()`. The aggregation at each individual DexiAttribute is governed by the corresponding `DexiAttribute$funct`. When alternative values are sets or distributions (see DEXiR-package), then `evaluate()` tries all possible combinations of values of the descendant attributes.

### Usage

```
evaluate(
  model,
  alternatives = model$alternatives,
  root = model$root,
  method = EnumEvalMethod,
  bounding = FALSE,
  prune = list(),
  norm = NULL,
  and = NULL,
  or = NULL
)
```

## Arguments

| | |
|---|---|
| `model` | DexiModel. |
| `alternatives` | A data frame containing data of one or more decision alternatives. |
| `root` | DexiAttribute. Default: `model$root`. |
| `method` | One of: `"set"` (default), `"prob"`, `"fuzzy"` or `"fuzzynorm"`. |
| `bounding` | `logical(1)`. When `TRUE`, evaluation results are additionally subjected to `bounded_scale_value()` to keep them in the bounds set up by the corresponding scale. |
| `prune` | `character()`, containing IDs of aggregate attributes that should be treated as evaluation inputs (rather than basic attributes). |
| `norm` | Some normalization function of the form `function(num_vector)`, or `NULL`. |
| `and` | Some conjunctive aggregation function of the form `function(num_vector)`, or `NULL`. |
| `or` | Some disjunctive aggregation function of the form `function(num_vector)`, or `NULL`. |

## Details

`evaluate()` implements four aggregation methods: `"set"`, `"prob"`, `"fuzzy"` and `"fuzzynorm"`.

The `"set"` method interprets DEXi values as sets. The output value assigned to some `attribute` is composed of the union of all `attribute$funct` evaluations for all possible combinations of values of `attribute$inputs`.

The remaining three methods interpret DEXi values as value distributions. They follow the same algorithm, but use different functions (see `evaluation_parameters()`) in three algorithm steps: normalization, and conjunctive and disjunctive aggregation. All values distributions involved in calculations are normalized by the function `norm()`. All combinations of `attribute$input` values are individually evaluated by the corresponding tabular function `attribute$funct`. The value $p$ of each set of `attribute$funct` arguments is determined by the conjunctive aggregation function `and()` over $p$'s of individual arguments. Finally, the $p$ of some output value `val` is determined by the disjunctive aggregation function `or()`, applied on the $p$'s of all partial evaluations that map to `val`.

For the mathematical background and more details about aggregation in DEX, please see (Trdin, Bohanec, 2018). For default normalization and aggregation functions, see `normalize_function()`, `and_function()` and `or_function()`.

## Value

A data frame containing both input and output (evaluated) values of `alternatives`.

## See Also

`evaluation_parameters()`, `normalize_function()`, `norm_none()`, `norm_max()`, `norm_sum()`, `and_function()`, `or_function()`, `bounded_scale_value()`.

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

alt <- Car$alternative("MyCar_set",
      BUY.PRICE="low", MAINT.PRICE=2, X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY="medium")
Car$evaluate(alt)

# Try the set-based evaluation using the default "set" method
alt <- Car$alternative("MyCar2",
       BUY.PRICE="low", MAINT.PRICE="*", X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY=2)
Car$evaluate(alt)

# Use value distributions and try the methods "prob", "fuzzy" and "fuzzynorm"
alt <- Car$alternative("MyCar_distr",
       BUY.PRICE="low", MAINT.PRICE=distribution(0.1, 0.6, 0.3),
       X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY=2)
Car$evaluate(alt, method = "prob")
Car$evaluate(alt, method = "fuzzy")
Car$evaluate(alt, method = "fuzzynorm")
```

---

## `evaluate_attribute` *evaluate_attribute*

---

### Description

Evaluate `alternative` for a sequence of `attribute` values.

### Usage

```
evaluate_attribute(model, attribute, alternative, seq = NULL, ...)
```

### Arguments

| | |
|---|---|
| `model` | A DexiModel. |
| `attribute` | A DexiAttribute with an assigned discrete or continuous scale. |
| `alternative` | A `data.frame` containing a single alternative. |
| `seq` | A sequence of `attribute` numeric values for which to evaluate `alternative`. For discrete scales: Must be a sequence of integers. Defaults to `attribute$scale$full_range()`. For continuous scales: `seq` is required. |
| `...` | Optional parameters passed to `evaluate()`. |

### Value

A list of evaluated alternatives for consecutive `attribute` values from `seq`.

### See Also

evaluate()

### Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

alt <- Car$alternative("MyCar",
      BUY.PRICE="low", MAINT.PRICE=2, X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY="medium")
safety <- Car$attrib("SAFETY")
# Evaluate alt for all values of att
evaluate_attribute(Car, safety, alt)
# Returns a list of three alternatives for values SAFETY=c("small", "medium", "high")
```

## evaluate_attributes *evaluate_attributes*

### Description

Apply `evaluate_attribute()` for all discrete `attributes`.

### Usage

```
evaluate_attributes(model, alternative, attributes = NULL, ...)
```

### Arguments

| | |
|---|---|
| `model` | A DexiModel object. Required. |
| `alternative` | A `data.frame` containing a single alternative. |
| `attributes` | List of attributes or vector of attribute names, ID's or indices. Default: All basic attributes of `model`. |
| `...` | Optional parameters passed to `evaluate_attribute()`. |

### Value

A list of `evaluate_attribute()` results for each `attribute`

### See Also

evaluate_attribute()

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

alt <- Car$alternative("MyCar",
      BUY.PRICE="low", MAINT.PRICE=2, X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY="medium")
safety <- Car$attrib("SAFETY")
# Perform evaluate_attribute() for all basic attributes of CarDxi
evaluate_attributes(Car, alt)
# Returns a list of evaluate_attribute() results corresponding to all basic attributes,
# indexed by attribute id
```

---

# `evaluation_order` *evaluation_order*

---

## Description

Determine the evaluation order of attributes. Interpreted as a sequence, the order guarantees that whenever some attribute is reached as a candidate for evaluation, all the previous attributes have been already evaluated.

## Usage

```
evaluation_order(att, prune = list())
```

## Arguments

| | |
|---|---|
| `att` | DexiAttribute. The starting point of evaluation. |
| `prune` | A character vector. May contain IDs of aggregate attributes at which the evaluation should stop, treating them as if they were basic attributes. |

## Value

A character vector of attribute IDs.

## See Also

`evaluate()`

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# Full evaluation order, starting with Car$root and without pruning
evaluation_order(Car$root)
```

```
# Evaluation order, starting with the TECH.CHAR. attribute
evaluation_order(Car$attrib("TECH.CHAR."))

# evaluation order, starting with Car$root and pruned at "PRICE"
evaluation_order(Car$root, prune = "PRICE")
```

## evaluation_parameters *evaluation_parameters*

### Description

Make a list containing parameters of DEXi evaluation. The parameters determine which method and normalization/aggregation functions should be used by `evaluate()`.

### Usage

```
evaluation_parameters(
  method = EnumEvalMethod,
  norm = NULL,
  and = NULL,
  or = NULL
)
```

### Arguments

| | |
|---|---|
| `method` | One of: `"set"` (default), `"prob"`, `"fuzzy"` or `"fuzzynorm"`. |
| `norm` | Some normalization function of the form `function(num_vector)`, or `NULL`. |
| `and` | Some conjunctive aggregation function of the form `function(num_vector)`, or `NULL`. |
| `or` | Some disjunctive aggregation function of the form `function(num_vector)`, or `NULL`. |

### Value

`list(method, norm, and, or)`. For `NULL` `norm`, `and`, and `or` arguments, defaults are taken depending on the `method`.

### See Also

evaluate, `normalize_function()`, `norm_none()`, `norm_max()`, `norm_sum()`, `and_function()`, `or_function()`.

### Examples

```
evaluation_parameters("prob", norm = norm_none)
```

## expand_value_to_points *expand_value_to_points*

### Description

Expand a DEXi `value` to a sequence of individual elements (points). Particularly aimed for graphic functions that display DEXi values with dots of different sizes and colors.

### Usage

```
expand_value_to_points(value, scale, colors = c("red", "black", "green"))
```

### Arguments

| | |
|---|---|
| `value` | A DEXi value: a single value (integer or float), a set (integer vector) or a distribution. |
| `scale` | A DexiScale object. |
| `colors` | `numeric(3)` representing colors to display "bad", "neutral" and "good" values, respectively. |

### Value

A `data.frame` consisting of:

`points` `numeric()`. `value` expanded to a vector of ordinal values.

`sizes` `numeric()`. Numeric values assigned to each corresponding ordinal values. Normally `1.0` for set elements and in the `(0,1]` interval for distribution membership values.

`colors` Colors assigned to corresponding value qualities.

### Examples

```
scl <- DexiDiscreteScale(values = c("L", "M", "H"))

expand_value_to_points(c(1, 3), scl)
# points sizes colors
# 1      1     1    red
# 2      3     1  green

expand_value_to_points(distribution(0.1, 0, 0.9), scl)
# points sizes colors
# 1      1   0.1    red
# 2      3   0.9  green
```

## export_alternatives *export_alternatives*

### Description

Convert alternatives' data to a data frame formatted so that it can be imported by DEXi/DEXiWin software.

### Usage

```
export_alternatives(model, alternatives = NULL)
```

### Arguments

| | |
|---|---|
| model | A DexiModel object. Required. |
| alternatives | A data.frame of alternatives (normally an output of evaluate()) or indices to model$alternatives. The default value NULL selects model$alternatives. |

### Details

In order to import the output of export_alternative() in DEXi/DEXiWin software, proper Import/Export settings must be ensured in these programs:

**DEXi** Option values: "base 1", Attributes: "all", Orientation: "normal", Indent: "indent".

**DEXiWin** Option values: "Base 1", Attributes: "All", Orientation: "Attributes \ Alternatives", Indent: "Indent tree levels", CSV Format: "Invariant" when format = "csv" and "Local" when format = "csv2".

If alternatives contain value distributions, they can be imported only by DEXiWin and not by DEXi.

### Value

A data frame consisting of character strings that can be further written out by write_alternatives().

### See Also

write_alternatives()

### Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

export_alternatives(Car)    # export both alternatives from Car
export_alternatives(Car, 1) # export only the first alternative
```

## export_dexi_value *export_dexi_value*

### Description

Convert a DEXi `value` to a character string that is understood by DEXi/DEXiWin software while importing data about alternatives.

### Usage

```
export_dexi_value(value)
```

### Arguments

| | |
|---|---|
| `value` | A DEXi value: `NA`, `NULL`, a single number, integer vector (a set) or a distribution. |

### Value

A string representation of `value`.

### Examples

```
export_dexi_value(NULL)                          # "<undefined>"
export_dexi_value(NA)                            # "<undefined>"
export_dexi_value(1)                             # "1"
export_dexi_value(3.2)                           # "3.2"
export_dexi_value(c(1, 3, 5))                    # "1;3;5"
export_dexi_value(distribution(0.1, 0.9))        # "1/0.1;2/0.9"
export_dexi_value(distribution(0, 0.1, 0, 0.9, 0)) # "2/0.1;4/0.9"
```

## flat_text *flat_text*

### Description

"Flatten" the function argument using `c(value)`, concatenate the elements and separate them by a single space.

### Usage

```
flat_text(value)
```

### Arguments

| | |
|---|---|
| `value` | Any object that can occur as an argument of `c()` and `as.character()`. |

### Value

character(1).

---

ggplot_parallel *ggplot_parallel*

---

### Description

Makes a basic ggplot2 chart for displaying DEXi alternatives using parallel axes. Generally, axes are uniformly scaled to the [0,1] interval.

### Usage

```
ggplot_parallel(
  model,
  alternatives = NULL,
  attids = NULL,
  aggregate = c("minmax", "min", "max", "mean", "none"),
  name = "name",
  shift = 0.01
)
```

### Arguments

| | |
|---|---|
| model | A DexiModel object. Required. |
| alternatives | A data.frame of alternatives (normally an output of evaluate()) or indices to model$alternatives. The default value NULL selects the whole model$alternatives. |
| attids | character(). A character vector of DexiAttribute IDs to be included in the result. Default: all model attributes. |
| aggregate | One of "minmax", "min", "max", "mean" or "none". Determines how to aggregate alternatives' values that are represented by sets or distributions. |
| name | character(1), The name of the column in alternatives that contains alternatives' names. Default: "name". |
| shift | numeric(1). Used to "shift" numeric values by a small amount to avoid overlapping lines in charts. Default: 0.01. You may want to experiment with charts to determine the right value, |

### Details

Uses GGally::ggparcoord() and requires package "GGally" to be installed. Data presented in the chart is prepared by scale_alternatives().

### Value

A basic 'ggplot2' chart. Generally, this chart needs to be further enhanced by graph layers, such as themes, labels, geom_points() and geom_line(). See plotalt_parallel() that already provides some such layers.

## See Also

`scale_alternatives()`, `plotalt_parallel()`

## Examples

```
if (requireNamespace("GGally", quietly = TRUE)) {

# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# Plot all Car$alternatives with points and lines
ggplot_parallel(Car) + ggplot2::geom_line(linewidth = 2) + ggplot2::geom_point(size = 3)
}
```

---

## `has_bad` *has_bad*

---

## Description

has_bad

## Usage

```
has_bad(value, scale)
```

## Arguments

| | |
|---|---|
| `value` | A DEXi value. |
| `scale` | A DexiScale or derived object. |

## Value

`logical(1)`. Whether or not `value_qualities(value, scale)` contains `"bad"`.

---

## `has_good` *has_good*

---

## Description

has_good

## Usage

```
has_good(value, scale)
```

### Arguments

| | |
|---|---|
| `value` | A DEXi value. |
| `scale` | A DexiScale or derived object. |

### Value

`logical(1)`. Whether or not `value_qualities(value, scale) contains "good"`‘.

---

## `has_none` *has_none*

---

### Description

has_none

### Usage

```
has_none(value, scale)
```

### Arguments

| | |
|---|---|
| `value` | A DEXi value. |
| `scale` | A DexiScale or derived object. |

### Value

`logical(1)`. Whether or not `value_qualities(value, scale)` contains `"none"`.

---

## `has_quality` *has_quality*

---

### Description

has_quality

### Usage

```
has_quality(quality = EnumQuality, value, scale)
```

### Arguments

| | |
|---|---|
| `quality` | A character string from `EnumQuality`. |
| `value` | A DEXi value. |
| `scale` | A DexiScale or derived object. |

### Value

`logical(1)`. Whether or not `value_qualities(value, scale)` contains `quality`.

## `is_distribution` *is_distribution*

### Description

Checks whether `value` is of `DexDistributionClass` or not.

### Usage

```
is_distribution(value)
```

### Arguments

| | |
|---|---|
| `value` | Any value or object to be checked. |

### Value

`logical(1)`. Returns `TRUE` if `value` is distribution.

### Examples

```
is_distribution(NULL)
is_distribution(3)
is_distribution("text")
is_distribution(c(1,2,3))
is_distribution(distribution(1,0,2))
```

## `is_in_range` *is_in_range*

### Description

Check whether or not `x` lies the specified range.

### Usage

```
is_in_range(x, lb, hb, lassoc = c("up", "down"), hassoc = c("down", "up"))
```

### Arguments

| | |
|---|---|
| `x` | Any object type, but using a non-numeric argument always returns `FALSE`. |
| `lb` | `numeric(1)`. Lower bound of the interval. |
| `hb` | `numeric(1)`. Ipper bound of the interval. |
| `lassoc` | `"up"` or `"down"`, indicating whether `lb` is included in the `[lb:hb]` interval (`"up"`) or not (`"down"`). The default is `"up"`. |
| `hassoc` | `"up"` or `"down"`, indicating whether `hb` is included in the `[lb:hb]` interval (`"down"`) or not (`"up"`). The default is `"down"`. |

## Value

`logical(1)`, indicating whether or not `x` lies in the interval `[lb:hb]` according to function arguments.

## Examples

```
is_in_range(3, 2, 5)
is_in_range(7, 2, 5)
is_in_range(3, 3, 5)
is_in_range(3, 3, 5, lassoc = "down")
```

---

`lin_map` *lin_map*

---

## Description

Map value `x` linearly from interval `[imin:imax]` to `[omax:omax]`.

## Usage

```
lin_map(x, imin, imax, omin = 0, omax = 1)
```

## Arguments

| | |
|---|---|
| `x` | `numeric()`. Value(s) to be mapped. |
| `imin` | `numeric()`. Lower bound of the input range. |
| `imax` | `numeric()`. Upper bound of the input range. |
| `omin` | `numeric()`. Lower bound of the output range. |
| `omax` | `numeric()`. Upper bound of the output range. |

## Value

`numeric()`. Mapped value(s).

## Examples

```
lin_map(2, 1, 3) # 0.5
```

## `make_args` *make_args*

### Description

Make a list of all possible combinations of values in a decision space defined by `dim`.

### Usage

```
make_args(dim)
```

### Arguments

| | |
|---|---|
| `dim` | A numeric vector containing upper bounds of the corresponding decision space dimensions. For example, `dim = c(3, 4)` defines the space of `3 * 4 == 12` combinations. |

### Value

A list containing all possible value combinations. List elements are numeric vectors of length equal to `length(dim)`.

### Examples

```
make_args(c(3, 4))
```

## `normalize_function` *normalize_function*

### Description

Determine the function to be used in the normalization step of `evaluate()`.

### Usage

```
normalize_function(method = EnumEvalMethod, norm = NULL)
```

### Arguments

| | |
|---|---|
| `method` | One of: `"set"` (default), `"prob"`, `"fuzzy"` or `"fuzzynorm"`. |
| `norm` | Some normalization function of the form `function(num_vector)`, or `NULL`. |

## Value

Returns function `norm` if not `NULL`. Otherwise, it determines the result depending on `method`:

`"set"`: `norm_none()`

`"prob"`: `norm_sum()`

`"fuzzy"`: `norm_none()`

`"fuzzynorm"`: `norm_max()`

Fails with an error if the result is not an R function.

## See Also

`evaluate`, `norm_none()`, `norm_max()`, `norm_sum()`,

---

`norm_max` *norm_max*

---

## Description

Normalize `values` so that `max(values)` == `max`.

## Usage

```
norm_max(values, max = 1)
```

## Arguments

| | |
|---|---|
| `values` | A numeric vector. |
| `max` | `numeric(1)`. |

## Value

`values` normalized so that `max(result)` == `max`. Returns unchanged `values` when `max(values)` == `0`.

## See Also

`norm_none()`, `norm_sum()`

## Examples

```
norm_max(c(0, 0.5, 0.7))
```

## `norm_none` *norm_none*

### Description

A "do nothing" normalization function.

### Usage

```
norm_none(values)
```

### Arguments

| | |
|---|---|
| `values` | A numeric vector. |

### Value

Returns unchanged `values`.

### See Also

`norm_max()`, `norm_sum()`

### Examples

```
norm_none(c(0, 0.5, 0.7))
```

## `norm_sum` *norm_sum*

### Description

Normalize `values` so that `sum(values) == sum`.

### Usage

```
norm_sum(values, sum = 1)
```

### Arguments

| | |
|---|---|
| `values` | A numeric vector. |
| `sum` | `numeric(1)`. |

### Value

`values` normalized so that `sum(result) == sum`. Returns unchanged `values` when `sum(values) == 0`

## See Also

`norm_none()`, `norm_max()`

## Examples

```
norm_sum(c(0, 0.5, 0.7))
```

---

`or_function` *or_function*

---

## Description

Determine the function to be used in the disjunctive aggregation step of `evaluate()`.

## Usage

```
or_function(method = EnumEvalMethod, or = NULL)
```

## Arguments

| | |
|---|---|
| `method` | One of: `"set"` (default), `"prob"`, `"fuzzy"` or `"fuzzynorm"`. |
| `or` | Some disjunctive aggregation function of the form `function(num_vector)`, or `NULL`. |

## Value

Returns the function `or` if not `NULL`. Otherwise, it determines the result depending on `method`:

`"set"`: `function(x) 1`

`"prob"`: `sum`

`"fuzzy"`: `max`

`"fuzzynorm"`: `max`

Fails with an error if the result is not an R function.

## See Also

`evaluate`, `and_function()`.

## plotalt1 *plotalt1*

### Description

Plot `alternatives` with respect to a single `attribute`.

### Usage

```
plotalt1(
  model,
  attribute = model$first(),
  alternatives = NULL,
  colors = c("red", "black", "green"),
  pch = 20,
  size = 5,
  linetype = 2,
  margins = NULL,
  lm = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| `model` | A DexiModel object. Required. |
| `attribute` | A single DexiAttribute selector. It may be an DexiAttribute object or an argument to `model$attrib()`. `attribute$scale` must be defined. Default: `model$first()`. |
| `alternatives` | A `data.frame` of alternatives (normally an output of `evaluate()`) or indices to `model$alternatives`. The default value `NULL` selects the whole `model$alternatives`. |
| `colors` | `character(3)` representing colors corresponding to "bad", "neutral" and "good" scale values, respectively. Default: `c("red", "black", "green")`. |
| `pch` | Plotting character, see `graphics::points()`. Default: `20`. |
| `size` | `numeric(1)`. Multiplication size factor for drawing individual points. Base point size depends on `pch`. |
| `linetype` | `integer()`. Line type for drawing chart grid. Default: `2`. |
| `margins` | `numeric(4)`. Chart margins, passed to `graphics::par()` prior to drawing. |
| `lm` | `numeric(1)`. Left chart margin. May be used to adjust the display of alternatives' names. |
| `...` | Optional parameters passed to `graphics::plot()`. |

### Details

Standard scatterplot base::plot is used.

## Value

Draws a chart.

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# Plot all Car$alternatives with respect to "TECH.CHAR." attribute
plotalt1(Car, "TECH.CHAR.")

# Plot the first Car alternative with respect to "MAINT.PRICE" attribute
plotalt1(Car, "MAINT.PRICE", 1)
```

---

`plotalt2` *plotalt2*

---

## Description

Draw a scatterpolot of `alternatives` with `attribute1` and `attribute2` on the $x$ and $y$ axis, respectively.

## Usage

```
plotalt2(
  model,
  attribute1,
  attribute2,
  alternatives = NULL,
  colors = NULL,
  pch = 20,
  size = 5,
  margins = NULL,
  lm = NULL,
  pos = 4,
  offset = 1,
  ...
)
```

## Arguments

| | |
|---|---|
| `model` | A DexiModel object. Required. |
| `attribute1` | First attribute. It may be an DexiAttribute object or an argument to `model$attrib()`. The attribute must be discrete. |

| | |
|---|---|
| `attribute2` | Second attribute. It may be an DexiAttribute object or an argument to `model$attrib()`. The attribute must be discrete. |
| `alternatives` | A `data.frame` of alternatives (normally an output of `evaluate()`) or indices to `model$alternatives`. The default value `NULL` selects the whole `model$alternatives`. |
| `colors` | `character()`. Colors for displaying subsequent alternatives. |
| `pch` | Plotting character, see `graphics::points()`. Default: `20`. |
| `size` | `numeric(1)`. Multiplication size factor for drawing individual points. Base point size depends on `pch`. |
| `margins` | `numeric(4)`. Chart margins, passed to `graphics::par()` prior to drawing. |
| `lm` | `numeric(1)`. Left chart margin. May be used to adjust the display of `attribute2`'s values. |
| `pos` | A position specifier for legent text, see `graphics::text()`. Default: `4`. |
| `offset` | When `pos` is specified, this value controls the distance of the text label from the specified coordinate in fractions of a character width. Default: `1`. |
| `...` | Optional parameters passed to `graphics::plot()`. |

## Details

Standard scatterplot `graphics::plot()` is used. Continuous attributes are not supported.

## Value

Draws a chart.

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# Plot all Car$alternatives with respect to "PRICE" and "TECH.CHAR." attributes
plotalt2(Car, "PRICE", "TECH.CHAR.")

# Plot the first Car alternative with respect to "BUY.PRICE" and "MAINT.PRICE" attributes
plotalt2(Car, "BUY.PRICE", "MAINT.PRICE", 1)
```

---

`plotalt_parallel` *plotalt_parallel*

---

## Description

Makes and plots DEXi alternatives on parallel axes, corresponding to attributes. Generally, axes are uniformly scaled to the `[0,1]` interval.

## Usage

```
plotalt_parallel(
  model,
  alternatives = NULL,
  attids = NULL,
  aggregate = c("minmax", "min", "max", "mean", "none"),
  name = "name",
  shift = 0.01,
  linewidth = 2,
  pointsize = 3,
  split = c("no", "h", "v")
)
```

## Arguments

| | |
|---|---|
| `model` | A DexiModel object. Required. |
| `alternatives` | A `data.frame` of alternatives (normally an output of `evaluate()`) or indices to `model$alternatives`. The default value `NULL` selects the whole `model$alternatives`. |
| `attids` | `character()`. A character vector of DexiAttribute IDs to be included in the result. Default: all `model` attributes. |
| `aggregate` | One of `"minmax"`, `"min"`, `"max"`, `"mean"` or `"none"`. Determines how to aggregate `alternatives` values that are represented by sets or distributions. |
| `name` | `character(1)`, The name of the column in `alternatives` that contains alternatives' names. Default: `"name"`. |
| `shift` | `numeric(1)`. Used to "shift" numeric results by a small amount to avoid overlapping lines in charts. Default: `0.01`. You may want to experiment with charts to determine the right value, |
| `linewidth` | `numeric(1)`. Widths of lines drawn. |
| `pointsize` | `numeric(1)`. Size of points drawn. |
| `split` | One of:<br>`"no"` Draw all alternatives on the same chart.<br>`"v"` Split the chart vertically and draw alternatives separately.<br>`"h"` Split the chart horizontally and draw alternatives separately. |

## Details

Data presented in the chart is prepared by `scale_alternatives()`. `plotalt_parallel()` invokes `ggplot_parallel()` to make a basic chart and then enhances it with graphic layers that are suitable for presenting DEXi alternatives.

## Value

A 'ggplot2' chart, enhanced with additional graph layers.

## See Also

`scale_alternatives()`, `ggplot_parallel()`

## Examples

```
if (requireNamespace("GGally", quietly = TRUE)) {

# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# Plot all Car$alternatives with points and lines
plotalt_parallel(Car)

# Show alternatives on two separate chart segments, shown one above the other.
plotalt_parallel(Car, split = "v")

alts3 <- structure(
list(
  name = c("MyCar", "MyCar2", "MyCar1b"),
    CAR.1 = list(4L, 4L, c(1L, 4L)),
    PRICE = list(3L, 3L, c(1L, 3L)),
    BUY.PRICE = list(3L, 3L, 3L),
    MAINT.PRICE = list(2, 1, structure(c(0.1, 0.6, 0.3), class = "distribution")),
    TECH.CHAR. = list(3L, 3:4, 3L),
    COMFORT = list(3L, 2, 3L),
    X.PERS = list(3, 3, 3L),
    X.DOORS = list(3, 3, 3L),
    LUGGAGE = list(2L, 2L, 2),
    SAFETY = list(2, c(2, 3), 2)
    ),
    row.names = c(NA, -3L),
    class = "data.frame"
 )

# Plot `alts2` with points and lines.
# Notice the "minmax" aggregation of sets and distributions.
plotalt_parallel(Car, alts3)
plotalt_parallel(Car, alts3, split = "v")

# Now with "mean" aggregation
plotalt_parallel(Car, alts3, split = "v", aggregate = "mean")
}
```

---

## `plotalt_radar` *plotalt_radar*

---

## Description

Plots DEXi alternatives on a radar chart. Generally, axes are uniformly scaled to the `[0,1]` interval.

## Usage

```
plotalt_radar(
  model,
  alternatives = NULL,
  attids = NULL,
  aggregate = c("minmax", "min", "max", "mean", "none"),
  name = "name",
  shift = 0.01,
  linewidth = 2,
  ptype = 16,
  colors = NULL,
  unicolors = NULL,
  fillcolors = NULL,
  transparency = 85,
  circular = FALSE,
  split = FALSE,
  fill = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `model` | A DexiModel object. Required. |
| `alternatives` | A `data.frame` of alternatives (normally an output of `evaluate()`) or indices to `model$alternatives`. The default value `NULL` selects the whole `model$alternatives`. |
| `attids` | `character()`. A character vector of DexiAttribute IDs to be included in the result. Default: all `model` attributes. |
| `aggregate` | One of `"minmax"`, `"min"`, `"max"`, `"mean"` or `"none"`. Determines how to aggregate `alternatives` values that are represented by sets or distributions. |
| `name` | `character(1)`, The name of the column in `alternatives` that contains alternatives' names. Default: `"name"`. |
| `shift` | `numeric(1)`. Used to "shift" numeric values by a small amount to avoid overlapping lines in charts. Default: `0.01`. You may want to experiment with charts to determine the right value, |
| `linewidth` | `numeric(1)`. Widths of lines drawn. |
| `ptype` | A vector to specify point symbol: Default `16` (closed circle). Should be 32 to not plot the points. This vector is repeatedly used for data series. |
| `colors` | Colors to be used (repeatably) for data series. Default `1:8`. |
| `unicolors` | A vector of one or two colors to be used for displaying the minimum and maximum data series, respectively. Applies only when `split = TRUE`. |
| `fillcolors` | A vector of color codes for filling polygons. Applies only when `fill = TRUE`. |
| `transparency` | A number between `0` and `100` representing the transparency of colors used for filling polygons. |
| `circular` | `logical(1)`. Whether to make a circular (using `fmsb::radarchartcirc()`) or polygonal (`fmsb::radarchart()`) radar grid. |

| | |
|---|---|
| `split` | `logical(1)`. Whether to plot all alternatives on a single chart (`FALSE`, default) or make a series of plots of individual alternatives (`TRUE`). |
| `fill` | `logical(1)`. Whether or not to fill polygons using `fillcolors`. |
| `...` | Optional parameters passed to `fmsb::radarchart()`. |

## Details

Uses `fmsb::radarchart()` and requires package "fmsb" to be installed. Data presented in the chart is prepared by `scale_alternatives()`.

## Value

Draws a chart or, when `split = TRUE` a series of charts corresponding to individual alternatives.

## See Also

`scale_alternatives()`, `fmsb::radarchart()`

## Examples

```
if (requireNamespace("fmsb", quietly = TRUE)) {

# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# Plot all Car$alternatives with points and lines
plotalt_radar(Car)

# Use different colors and fill polygons
plotalt_radar(Car, colors = c("blue", "brown"), fill = TRUE)
plotalt_radar(Car, colors = c("blue", "brown"), fillcolors = c("green", "red"), fill = TRUE)

# Draw separate charts
plotalt_radar(Car, split = TRUE)

# Draw separate charts, using the same color settings on all charts
plotalt_radar(Car, split = TRUE, unicolors = c("green", "red"))
plotalt_radar(Car, split = TRUE, unicolors = c("green", "red"), circular = TRUE)

alts3 <- structure(
list(
  name = c("MyCar", "MyCar2", "MyCar1b"),
    CAR.1 = list(4L, 4L, c(1L, 4L)),
    PRICE = list(3L, 3L, c(1L, 3L)),
    BUY.PRICE = list(3L, 3L, 3L),
    MAINT.PRICE = list(2, 1, structure(c(0.1, 0.6, 0.3), class = "distribution")),
    TECH.CHAR. = list(3L, 3:4, 3L),
    COMFORT = list(3L, 2, 3L),
    X.PERS = list(3, 3, 3L),
    X.DOORS = list(3, 3, 3L),
    LUGGAGE = list(2L, 2L, 2),
    SAFETY = list(2, c(2, 3), 2)
    ),
    row.names = c(NA, -3L),
    class = "data.frame"
 )

# The same chart types as above, but using more varied alternatives data
# Plot all Car$alternatives with points and lines
plotalt_radar(Car, alts3)

# Use different colors and fill polygons
plotalt_radar(Car, alts3, colors = c("blue", "brown", "purple"), fill = TRUE)
plotalt_radar(Car, alts3, colors = c("blue", "brown", "purple"),
  fillcolors = c("green", "red", "yellow"), fill = TRUE)

# Draw separate charts
plotalt_radar(Car, alts3, split = TRUE)
plotalt_radar(Car, alts3, split = TRUE, fill = TRUE)

# Draw separate charts, using the same color settings on all charts
plotalt_radar(Car, alts3, split = TRUE, unicolors = c("red", "green"))
plotalt_radar(Car, alts3, split = TRUE, unicolors = c("green", "darkgreen"), fill = TRUE)
plotalt_radar(Car, alts3, split = TRUE, unicolors = c("red", "green"), circular = TRUE)

}
```

## plus_minus *plus_minus*

### Description

Plus-Minus Analysis: Investigate the effects of changing single attributes values on the evaluation of `alternative`. The values of discrete basic attributes ("input attributes") are changed, one attribute at a time, by a particular number of steps downwards (`minus`) and upwards (`plus`), while observing the changes of the `target` attribute values.

### Usage

```
plus_minus(
  model,
  alternative,
  target = model$first(),
  minus = .Machine$integer.max,
  plus = .Machine$integer.max,
  print = TRUE,
  as_character = FALSE,
  round = NULL,
  id = NULL,
  evaluate = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `model` | A DexiModel object. |
| `alternative` | Either a `data.frame` representing a single alternative or an index to `model$alternatives`. |
| `target` | The attribute on which effects are observed. Default: `model$first()`. |
| `minus` | The maximum number of downward steps to be made for each input attribute. Default: `.Machine$integer.max`. The actual `minus` value is further determined with respect to `alternative` values and involved attributes' scales. |
| `plus` | The maximum number of upward steps to be made for each input attribute. Default: `.Machine$integer.max`. The actual `plus` value is further determined with respect to `alternative` values and involved attributes' scales. |
| `print` | `logical(1)`. When `TRUE`, pretty print (left justify) the results. |
| `as_character` | `logical(1)`. Whether to represent alternative values numerically (`FALSE`) or using text (`TRUE`). |
| `round` | An integer number, argument to `value_text()`. |
| `id` | `character(1)`. Determines the contents of the first or first two columns of the resulting `data.frames`: `"id"` Attribute ID. `"structure"` Attribute `$structure() + $name`. **anything else** Equivalent to both `"id"` and `"structure"`. |
| `evaluate` | `logical(1)`. Whether or not to evaluate `alternative` beforehand. |
| `...` | Optional parameters for `evaluate()`. |

## Value

A data frame consisting of columns:

`id` IDs of input attributes (unless excluded by the `id` argument).

`structure` Structure and names of input attributes (unless excluded by the `id` argument).

**'For** `-minus` **to** `-1` Evaluation value of `target` when decreasing the corresponding attribute value by the corresponding number of steps.

`target$id` Original `alternative` value assigned to the corresponding attribute `id`.

**For** `1` **to** `plus` Evaluation value of `target` when increasing the corresponding attribute value by the corresponding number of steps.

Special values `"["` and `"]"` denote that it is not possible to decrease of increase, respectively, the corresponding attributes value further.

## See Also

`evaluate()`, `value_text()`

### Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

alt <- Car$alternative("MyCar",
    BUY.PRICE="low", MAINT.PRICE="*", X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY=c(1, 3))
alte <- Car$evaluate(alt)

# Default plus-minus analysis, evaluating `alt`.
plus_minus(Car, alt, evaluate = TRUE)

# Plus-minus analysis of `alte`, using character strings,
# no pretty-printing and excluding structure info.
plus_minus(Car, alte, as_character=TRUE, print=FALSE, id = "id")

# Plus-minus analysis on `target="PRICE"`, using character strings.
plus_minus(Car, alt, target="PRICE", as_character=TRUE, evaluate=TRUE)
```

---

## `plus_minus_setup` *plus_minus_setup*

---

### Description

A helper function: Initializes a data frame for `plus_minus()`.

### Usage

```
plus_minus_setup(evaluated, attributes, minus, plus)
```

### Arguments

| | |
|---|---|
| `evaluated` | An evaluated alternative. |
| `attributes` | Vector of DexiAttribute objects involved in `plus-minus` analysis. |
| `minus` | A single integer: Maximum steps down. |
| `plus` | A single integer: Maximum steps up. |

### Value

A data frame consisting of columns:

`"id"` Attribute IDs.

`"structure"` Attribute `$structure() + $name`.

`counts` Attributes' scale sizes.

`low_bounds` Low bounds of attributes' values.

`high_bounds` High bounds of attributes' values.

| | |
|---|---|
| `low_diff` | Maximum possible value decrease given `low_bound` and attribute scale. |
| `high_diff` | Maximum possible value increase given `high_bound` and attribute scale. |
| `evals` | Alternative evaluation for the corresponding attribute (from `evaluated`). |
| `sets` | `evals` represented as value sets. |

### See Also

`plus_minus()`

## `print_selective_explanation` *print_selective_explanation*

### Description

A helper function for `selective_explanation()`: Pretty-prints its results.

### Usage

```
print_selective_explanation(explanation)
```

### Arguments

| | |
|---|---|
| `explanation` | A list of lists, containing selective explanation results produced by `selective_explanation()`. |

### Value

`NULL`. Pretty-prints the contents of `explanation`.

## `read_dexi` *read_dexi*

### Description

`read_dexi()` reads a definition of a DEXi model from a `.dxi` file or XML string.

### Usage

```
read_dexi(dxi)
```

### Arguments

| | |
|---|---|
| `dxi` | character(1). A `.dxi` file name or XML string. |

### Value

A DexiModel RC object.

### See Also

DexiModel

### Examples

```
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)
```

---

## reverse_value *reverse_value*

---

### Description

Numeric value(s) `x` are assumed to lie within the `[lb:hb]` interval. The function "reverses" `x` linearly so that `x = lb` maps to `hb` and `x = hb` maps to `lb`. In DEXiR, this function is used to reverse values defined on a DexiScale from "ascending" to "descending" order or vice versa.

### Usage

```
reverse_value(x, lb, hb)
```

### Arguments

| | |
|---|---|
| `x` | `numeric()`. Value(s) to be reversed. |
| `lb` | `numeric()`. Lower interval bound(s). |
| `hb` | `numeric()`. Upper interval bound(s). |

### Value

`numeric()`. Reversed `value`.

### Examples

```
reverse_value(1, 1, 5) # 5
reverse_value(3, 1, 5) # 3
reverse_value(5, 1, 5) # 1
reverse_value(c(1, 3, 5), 1, 5) # c(5, 3, 1)
```

## `rule_value` *rule_value*

### Description

Values of decision rules are in `.dxi` files encoded using character strings, where each individual character encodes some function value. The encoding is zero-based, so that `"0"` represents the lowest ordinal number on the corresponding discrete scale. `rule_value(char)` converts a single character to the corresponding ordinal value.

### Usage

```
rule_value(ch)
```

### Arguments

| | |
|---|---|
| `ch` | A single character, such as `"3"` or `"Z"`. |

### Value

Corresponding integer value.

### Examples

```
rule_value("1")
rule_value("Z")
```

## `rule_values` *rule_values*

### Description

Values of decision rules are in `.dxi` files encoded using character strings, where each individual character encodes some function value. The encoding is zero-based, so that the character `"0"` represents the lowest ordinal number on the corresponding discrete scale. Encoding of characters is according to ASCII, starting with `"0"`. `rule_values(str)` converts the character string to a numeric vector of corresponding ordinal values.

### Usage

```
rule_values(str, add = 0)
```

## Arguments

| | |
|---|---|
| str | character(1), a DEXi encoding of a vector of ordinal numbers. |
| add | An integer constant to be added to the resulting vector. The default is add = 0, however DEXi's ordinal numbers should normally be converted to R's using add = 1. |

## Value

A numeric vector of the same length as str.

## Examples

```
rule_values("01122:")
rule_values("01122:", add = 1)
```

---

scale_alternatives *scale_alternatives*

---

## Description

A helper function for preparing alternatives' data for charts that involve multiple attributes (such as plotalt_parallel()) and plotalt_radar()). scale_alternatives() carries out three main operations:

1. Aggregates DEXi values, represented by sets and distributions, into single numeric values, using one of the aggregate operators: "minmax", "min", "max" or "mean",
2. scales the aggregated values to the [0,1] interval so that they can be drawn uniformly on multiple chart axes,
3. optionally "shifts" the values by a small amount to avoid overlapping chart lines.

## Usage

```
scale_alternatives(
  model,
  alternatives = NULL,
  attids = NULL,
  aggregate = c("minmax", "min", "max", "mean", "none"),
  name = "name",
  shift = 0.01
)
```

## Arguments

| | |
|---|---|
| `model` | A DexiModel object. Required. |
| `alternatives` | A `data.frame` of alternatives (normally an output of `evaluate()`) or indices to `model$alternatives`. The default value `NULL` selects the whole `model$alternatives`. |
| `attids` | `character()`. A character vector of DexiAttribute IDs to be included in the result. Default: all `model` attributes. |
| `aggregate` | Determines how to aggregate DEXi values that are represented/interpreted as sets in `alternatives`:<br>`"min"` Uses the function `min()` to take the minimal set element.<br>`"max"` Uses the function `max()` to take the maximal set element.<br>`"mean"` Uses the function `mean()` to take the average set value.<br>`"minmax"` **(default)** Takes both `"min"` and `"max"`, so that each alternative appears in the result twice.<br>`"none"` No aggregation.<br>Any distributions that appear in `alternatives` are interpreted as sets prior to aggregation. The default operator `"minmax"` is suitable particularly for alternatives containing non-single-values (sets and/or distributions). For alternatives containing only single numeric values, any of the other three operators is preferred. |
| `name` | `character(1)`, The name of the column in `alternatives` that contains alternatives' names. Default: `"name"`. |
| `shift` | `numeric(1)`. Used to "shift" numerical values by a small amount to avoid overlapping lines in charts. Default: `0.01`. |

## Value

A list containing the elements:

`data` A data frame containing the aggregated/scaled/shifted numeric values.

`nalt` The number of alternatives. Notice that with `aggregate = "minmax"`, `data` contains twice as many rows.

`groups` A numeric vector mapping `data` rows to `alternatives`' indices.

`altnames` Names of alternatives.

## See Also

`plotalt_parallel()`), `plotalt_radar()`)

## `scale_of` *scale_of*

### Description

scale_of

### Usage

```
scale_of(obj)
```

### Arguments

| | |
|---|---|
| `obj` | A DexiAttribute or DexiScale. |

### Value

A DexiScale associated with `obj`, or `NA` for an undefined scale.

## `scale_value` *scale_value*

### Description

Check and interpret `value` on `scale`.

### Usage

```
scale_value(value, scale)
```

### Arguments

| | |
|---|---|
| `value` | A wide range of possible value types, including integer, double, character and list vectors. |
| `scale` | A DexiScale or derived object. |

### Value

The result is produced depending on `value` and `scale` according to the following tables. For any scale type:

```
value                          result
------------------------------+------------------
NULL                           NULL
length(value == 0)             NULL
NA                             scale$full_range()
other types                    ERROR
value contains any NULL or NA  ERROR
------------------------------+------------------
```

For continuous scales:

```
value                          result
------------------------------+------------------
length(value != 1)             ERROR
character                      ERROR
named object                   ERROR
length(value == 1)             unclass(value)
------------------------------+------------------
```

For discrete scales:

```
value                          result
------------------------------+------------------
distribution class             value
all-integer numeric vector     value
non all-integer numeric vector distribution(value)
"*" or "undef"...              scale$full_range()
list of value names            matched value set
list of name=p                 distribution(value)
------------------------------+------------------
```

## Examples

```
# Examples of successfully checked (witout error) values on a continuous scale
scl <- DexiContinuousScale()
scale_value(NULL, scl)              # NA
scale_value(c(), scl)               # NA
scale_value(list(), scl)            # NA
scale_value(character(), scl)       # NA
scale_value(NA, scl)                # NA
scale_value(c(NA), scl)             # NA
scale_value(15.5, scl)              # 15.5
scale_value(distribution(15.5), scl) # 15.5

# Examples of successfully checked (without error) values on a discrete scale
scl <- DexiDiscreteScale(values = c("low", "med", "high"))
scale_value(NULL, scl)                       # NA
scale_value(c(), scl)                        # NA
scale_value(list(), scl)                     # NA
scale_value(NA, scl)                         # NA
scale_value("*", scl)                          # 1:3
scale_value("Undefined", scl)                  # 1:3
scale_value(2, scl)                            # 2
scale_value(c(-1, 2, 4), scl)                  # c(-1, 2, 4))
scale_value(distribution(c(-1, 2, 4)), scl)    # distribution(c(-1, 2, 4)))
scale_value(c(-1, 2.2, 4), scl)                # distribution(c(-1, 2.2, 4)))
scale_value("high", scl)                       # 3
scale_value(c("low", "high"), scl)             # c(1,3))
v <- c(0.5, 0.4)
names(v) <- c("low", "high")
scale_value(v, scl)                            # distribution(c(0.5, 0, 0.4)))
scale_value(list(high = 1.1, low = 2.2), scl) # distribution(c(2.2, 0, 1.1)))
```

---

## scale_values *scale_values*

---

### Description

A vectorized version of `scale_value`.

### Usage

```
scale_values(values, scale)
```

### Arguments

| | |
|---|---|
| `values` | A list of values. For possible value types, see `scale_value()`. |
| `scale` | A DexiScale or derived object. |

### Value

A list determined as `lapply(values, function (v) scale_value(v, scale))`.

### See Also

`scale_value()`

## `selective_explanation` *selective_explanation*

### Description

Selective Explanation: Displays subtrees of alternatives' values in which values are particularly weak (value quality is "bad") and particularly strong (value quality is "good").

### Usage

```
selective_explanation(
  model,
  alternatives = NULL,
  print = TRUE,
  as_character = FALSE,
  round = NULL,
  id = NULL,
  evaluate = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| `model` | A DexiModel object. Required. |
| `alternatives` | A `data.frame` of alternatives or indices to `model$alternatives`. The default value `NULL` selects `model$alternatives`. |
| `print` | `logical(1)`. When `TRUE`, pretty print (add headings and left justify) the results, using `print_selective_explanation()`. |
| `as_character` | `logical(1)`. Whether to represent alternative values numerically (`FALSE`) or using text (`TRUE`). |
| `round` | An integer number, argument to `value_text()`. |
| `id` | `character(1)`. Determines the contents of the first or first two columns of the resulting `data.frames`:<br>`"id"` Attribute IDs.<br>`"structure"` Attribute `$structure() + $name`.<br>**anything else** Equivalent to both `"id"` and `"structure"`. |
| `evaluate` | `logical(1)`. Whether or not to evaluate `alternatives` beforehand. |
| `...` | Optional parameters for `evaluate()`. |

### Value

A list of lists: For each alternative contains a list of two data.frames, corresponding to `"bad"` and `"good"` qualities, respectively. May be pretty-printed using `print_selective_explanation()`.

### See Also

value_qualities(), value_text(), print_selective_explanation(), evaluate()

### Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# Print selective explanation of two Car$alternatives.
selective_explanation(Car)

alt <- Car$alternative("MyCar",
    BUY.PRICE="low", MAINT.PRICE="*", X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY=c(1, 3))
alte <- Car$evaluate(alt)

# Print selective explanation of `alte`.
selective_explanation(Car, alte)

# Print selective explanation of both `alt` and `alte`.
selective_explanation(Car, rbind.data.frame(alt, alte))
```

---

## `select_quality` *select_quality*

---

### Description

Select from `alt` only those attributes whose values have the given `quality`. Used primarily in `selective_explanation()`.

### Usage

```
select_quality(model, alt, quality)
```

### Arguments

| | |
|---|---|
| `model` | A DexiModel object. |
| `alt` | `data.frame`. A single DEXi alternative. |
| `quality` | Requested `EnumQuality`: `"bad"`, `"good"` or `"none"`. |

### Value

`alt` containing only values that have the requested quality.

### See Also

value_qualities(), selective_explanation()

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

alt <- Car$alternative("MyCar",
    BUY.PRICE="low", MAINT.PRICE="*", X.PERS="more", X.DOORS="4", LUGGAGE=2, SAFETY=c(1, 3))
alte <- Car$evaluate(alt)
alts <- select_quality(Car, alte, "bad")
names(alts)
# c("CAR", "PRICE", "MAINT.PRICE", "TECH.CHAR.", "SAFETY")
alts <- select_quality(Car, alte, "none")
names(alts)
# c("MAINT.PRICE", "X.DOORS", "LUGGAGE")
alts <- select_quality(Car, alte, "good")
names(alts)
# c("CAR", "PRICE", "BUY.PRICE", "MAINT.PRICE", "TECH.CHAR.", "COMFORT", "X.PERS", "SAFETY")
```

---

## `set_alternative` *set_alternative*

---

### Description

Set values of a single decision alternative and represent it with a data frame. Usually, only input values are set in this way. The data frame can then be evaluated to set the values of output attributes.

### Usage

```
set_alternative(model, alternative, ...)
```

### Arguments

| | |
|---|---|
| `model` | A DexiModel object. Required. |
| `alternative` | `character(1)` or `data.frame`. The first form sets the name of the newly created decision alternative. The second form copies values from `alternative[1, ]` to initialize the corresponding columns of the resulting data frame. |
| `...` | A list of parameters specifying the values of the newly created decision alternative. Each parameter is expected to be in the form `attribute_id=attribute_value`, or is a list of elements of the same form.<br>There are several possible ways to specify `attribute_value`. Taking the scale `CAR` = {`"unacc"`; `"acc"`; `"good"`; `"exc"`} as an example, the options are:<br>`CAR="unacc"` A single qualitative value.<br>`CAR=2` An ordinal number, indicating `"acc"` in this case.<br>`CAR=c("good", "exc")` A set of qualitative values.<br>`CAR=c(3, 4)` A set of ordinal numbers, equivalent to the above.<br>`CAR=list("good", 4)` A set specified by a mixture of qualitative values and ordinal numbers. |

`CAR="*"` A full range of ordinal numbers, in this case equivalent to `1:4`.

`CAR=distribution(0, 0, 0.7, 0.3)` A value distribution.

`CAR=list("good"=0.7, "exc"=0.3)` A value distribution, equivalent to the above.

`CAR="undef"` An unknown value, interpreted as `NA`.

For attributes associated with continuous scales, only `numeric(1)` `attribute_values` are allowed.

## Value

A one-row data frame with columns corresponding to `model`'s attributes, collectively representing a single decision alternative. The columns not copied from `alternative` (as a data frame) nor set by any parameter contain `NA`s.

## See Also

DEXiR-package notes on values in DEXi models.

---

`set_to_distr` *set_to_distr*

---

## Description

Convert a DEXi value set to DEXi value distribution.

## Usage

```
set_to_distr(set, length = 0)
```

## Arguments

| | |
|---|---|
| `set` | Normally a numeric vector containing integer numbers. |
| `length` | The required length of the resulting distribution vector. The actual length is determined as `max(length, max(set))`, so the `length` is extended when too small to hold the whole distribution. |

## Value

A distribution object of length `length`. Arguments that are already distributions are returned "as is". Input vectors of length 0 and other types of objects return `NA`.

## See Also

DEXiR-package, distribution, `distr_to_set()`

### Examples

```
set_to_distr(c(1, 3, 4))
set_to_distr(c(1, 3, 4), length = 5)
set_to_distr(c(1, 3, 4), length = 0)
```

---

## transparent_colors *transparent_colors*

---

### Description

A helper function for making `colors` transparent.

### Usage

```
transparent_colors(colors, percent = 50)
```

### Arguments

| | |
|---|---|
| `colors` | A vector of color numbers or names. |
| `percent` | Required color transparency, in the range `[0:100]`. |

### Details

Requires installed package "grDevices".

### Value

A vector of colors of the same length as `colors`.

### Examples

```
transparent_colors(c("red", "green", "blue"), 50)
# c("#FF00007F", "#00FF007F", "#0000FF7F")
```

## unique_names *unique_names*

### Description

Convert names strings to ID strings that are unique and conformant with R's syntactic rules for variable names.

### Usage

```
unique_names(names, reserved = c())
```

### Arguments

| | |
|---|---|
| `names` | `character()`. Names to be converted to IDs. |
| `reserved` | `character()`. Reserved names that should not be used as IDs. |

### Value

`character()`.

### See Also

`base::make.unique()`]

## values_to_str *values_to_str*

### Description

Convert numbers to a DEXi string. Implements the reverse operation of `rule_values()`.

### Usage

```
values_to_str(vals, add = 0)
```

### Arguments

| | |
|---|---|
| `vals` | Numeric vector, containing ordinal values. |
| `add` | An integer constant to be added to `vals` prior to conversion. |

### Value

A string representing DEXi's representation of ordinal values. Fails when `vals` + `add` contains negative numbers.

### Examples

```
values_to_str(c(0, 1, 1, 2, 2, 10, 12))
values_to_str(c(1, 2, 2, 3, 3, 11, 13), -1)
```

---

## value_qualities *value_qualities*

---

### Description

Returns a vector of qualities corresponding to consecutive elements of `value`. In contrast with `DexiScale$value_quality(value)`, which can handle only single values, this function can handle `value` arguments that contain multiple elements, such as value sets and distributions.

### Usage

```
value_qualities(value, scale)
```

### Arguments

| | |
|---|---|
| `value` | A DEXi value, internal representation: numeric value or vector, or distribution. |
| `scale` | A DexiScale or derived object. |

### Value

A vector consisting of `EnumQuality` elements corresponding to individual `value` elements.

### Examples

```
scl <- DexiDiscreteScale(values = c("low", "med", "high"))
value_qualities(1, scl)       # "bad"
value_qualities(1:3, scl)     # c("bad", "none", "good")
value_qualities(c(3, 2), scl) # c("good", "none")
```

---

## value_text *value_text*

---

### Description

Converts a DEXi `value` to a human-readable character string that can be printed. Used, for instance, by `DexiModel$as_character()`.

### Usage

```
value_text(value, scale, round = NULL)
```

## Arguments

| | |
|---|---|
| `value` | Any DEXi value type (see DEXiR-package). |
| `scale` | A DexiScale or derived object. |
| `round` | An integer number. Indicates the number of decimals for rounding numeric values prior to printing. If `NULL`, no rounding takes place. |

## Value

character.

## Examples

```
scl <- DexiDiscreteScale(values = c("low", "med", "high"))
value_text(NA, scl)
value_text(1, scl)
value_text(c(1, 3), scl)
value_text(distribution(0.1, 0.2, 0.3), scl)
```

---

`value_to_set` *value_to_set*

---

## Description

value_to_set

## Usage

```
value_to_set(value, scale)
```

## Arguments

| | |
|---|---|
| `value` | A DEXi value, internal representation: numeric value or vector, or distribution. |
| `scale` | A DexiScale or derived object. |

## Value

An integer vector or `NA` for: non-discrete scale, `NA`/`NULL` value(s), non-integer value(s).

## Examples

```
scl <- DexiDiscreteScale(values = c("low", "med", "high"))
value_to_set(1, scl)                      # 1
value_to_set(1:2, scl)                    # c(1, 2)
value_to_set(c(1,3), scl)                 # c(1, 3)
value_to_set(distribution(1, 0, 0.5), scl) # c(1, 3)
```

---

`write_alternatives` *write_alternatives*

---

## Description

Write out `alternatives`' data. First convert DEXi alternatives to a data frame using `export_alternatives()` and then write it to a file.

## Usage

```
write_alternatives(
  model,
  alternatives = NULL,
  file = "",
  quote = FALSE,
  format = c("tab", "csv", "csv2"),
  ...
)
```

## Arguments

| | |
|---|---|
| `model` | A DexiModel object. Required. |
| `alternatives` | A `data.frame` of alternatives (normally an output of `evaluate()`) or indices to `model$alternatives`. The default value `NULL` selects `model$alternatives`. |
| `file` | Write the data frame contents to a file. When `file = ""`, the contents is written to the console (default). `file = "clipboard"` might also work to copy the contents to the clipboard. |
| `quote` | `logical(1)`. Whether or not to quote output character strings. |
| `format` | One of `"tab"`, `"csv"` or `"csv2"` to invoke `write.table()`, `write.csv()` or `write.csv2()`, respectively. |
| `...` | Optional parameters to `write.table()` functions. |

## Value

Writes a "tab"- or "csv"-formatted `alternatives`' data to a file, console or clipboard. This data is meant to be subsequently imported to 'DEXi' software.

## See Also

`export_alternatives()`, `write.table()`

## Examples

```
# Load "Car.dxi"
CarDxi <- system.file("extdata", "Car.dxi", package = "DEXiR")
Car <- read_dexi(CarDxi)

# Write both Car alternatives to console
write_alternatives(Car, file = "")
```

# Index