Package ‘struct’

May 3, 2024

Type Package

Title Statistics in R Using Class-based Templates

Version 1.16.0

Description Defines and includes a set of class-based templates for developing and implementing data processing and analysis workflows, with a strong emphasis on statistics and machine learning. The templates can be used and where needed extended to 'wrap' tools and methods from other packages into a common standardised structure to allow for effective and fast integration. Model objects can be combined into sequences, and sequences nested in iterators using overloaded operators to simplify and improve readability of the code. Ontology lookup has been integrated and implemented to provide standardised definitions for methods, inputs and outputs wrapped using the class-based templates.

License GPL-3

Encoding UTF-8

Collate 'generics.R' 'ontology_term_class.R' 'struct_class.R'
'parameter_class.R' 'chart_class.R' 'stato_class.R'
'DatasetExperiment_class.R' 'entity_class.R'
'entity_stato_class.R' 'enum_class.R' 'enum_stato_class.R'
'output_class.R' 'model_class.R' 'example_objects.R'
'model_list_class.R' 'metric_class.R' 'iterator_class.R'
'optimiser_class.R' 'preprocess_class.R' 'resampler_class.R'
'struct-package.R' 'struct_templates.R' 'zzz.R'

RoxygenNote 7.3.1

Depends R (>= 4.0)

Suggests testthat, rstudioapi, rmarkdown, covr, BiocStyle, openxlsx, ggplot2, magick

VignetteBuilder knitr

Imports methods,ontologyIndex, datasets, graphics, stats, utils, knitr, SummarizedExperiment, S4Vectors, rols

biocViews WorkflowStep

git_url https://git.bioconductor.org/packages/struct
Contents

struct-package ................................................................. 3
.DollarNames.struct_class .................................................. 4
as.code ................................................................. 5
as.DatasetExperiment ......................................................... 6
as.DatasetExperiment.SummarizedExperiment-method .................. 7
as.SummarizedExperiment ................................................... 7
as.SummarizedExperiment.DatasetExperiment-method ................. 8
as_data_frame ............................................................ 8
c.ontology_list-method ..................................................... 9
calculate ................................................................. 9
chart ................................................................. 11
chart_names ............................................................. 11
chart_plot ............................................................. 12
citations ............................................................... 13
DatasetExperiment ........................................................... 14
entity_stato ............................................................. 15
enum ................................................................. 16
enum_stato ............................................................. 17
example_chart ............................................................ 18
example_iterator-class ..................................................... 19
example_model ............................................................ 20
export_xlsx ............................................................. 21
get_description ........................................................... 21
iris_DatasetExperiment ..................................................... 22
is_output ............................................................... 22
is_param ............................................................... 23
libraries ................................................................. 24
max_length ............................................................... 24
model ................................................................. 26
models ................................................................. 28
model_apply ............................................................. 29
model_predict ............................................................ 30
model_reverse ............................................................ 30
model_seq ............................................................... 31
model_train ............................................................ 33
struct-package

**Description**

Defines and includes a set of class-based templates for developing and implementing data processing and analysis workflows, with a strong emphasis on statistics and machine learning. The templates can be used and where needed extended to "wrap" tools and methods from other packages into a common standardised structure to allow for effective and fast integration. Model objects can be combined into sequences, and sequences nested in iterators using overloaded operators to simplify and improve readability of the code. Ontology lookup has been integrated and implemented
to provide standardised definitions for methods, inputs and outputs wrapped using the class-based templates.

**Author(s)**

**Maintainer:** Gavin Rhys Lloyd <g.r.lloyd@bham.ac.uk>

Authors:

- Ralf Johannes Maria Weber <r.j.weber@bham.ac.uk>

---

**Description**

This function returns slotnames for autocompletion when using $ syntax.

**Usage**

```r
## S3 method for class 'struct_class'
.DollarNames(x, pattern = "")

## S4 method for signature 'struct_class'
.DollarNames(x, pattern = "")

## S3 method for class 'chart'
.DollarNames(x, pattern = "")

## S4 method for signature 'chart'
.DollarNames(x, pattern = "")

## S3 method for class 'DatasetExperiment'
.DollarNames(x, pattern = "")

## S4 method for signature 'DatasetExperiment'
.DollarNames(x, pattern = "")

## S3 method for class 'model'
.DollarNames(x, pattern = "")

## S4 method for signature 'model'
.DollarNames(x, pattern = "")

## S3 method for class 'metric'
.DollarNames(x, pattern = "")
```
as.code

```r
## S4 method for signature 'metric'
.DollarNames(x, pattern = "")

## S3 method for class 'iterator'
.DollarNames(x, pattern = "")

## S4 method for signature 'iterator'
.DollarNames(x, pattern = "")

## S3 method for class 'optimiser'
.DollarNames(x, pattern = "")

## S4 method for signature 'optimiser'
.DollarNames(x, pattern = "")

## S3 method for class 'preprocess'
.DollarNames(x, pattern = "")

## S4 method for signature 'preprocess'
.DollarNames(x, pattern = "")

## S3 method for class 'resampler'
.DollarNames(x, pattern = "")

## S4 method for signature 'resampler'
.DollarNames(x, pattern = "")
```

**Arguments**

- `x` a struct_class object
- `pattern` the text used to compare against the slot names

**Value**

A vector of slot names

---

**Description**

Prints a block of code that can be used to replicate the input object.
Usage

```r
as.code(M, start = "M = ", mode = "compact", quiet = FALSE)
```

```r
## S4 method for signature 'struct_class'
as.code(M, start = "M = ", mode = "compact", quiet = FALSE)
```

```r
## S4 method for signature 'model_seq'
as.code(M, start = "M = ", mode = "compact", quiet = FALSE)
```

```r
## S4 method for signature 'iterator'
as.code(M, start = "M = ", mode = "compact", quiet = FALSE)
```

Arguments

- **M**: a struct model, model_seq or iterator object
- **start**: text prepended to the code. Default is "M = "
- **mode**: "compact" will use the least amount of lines, "expanded" will put each object and input on a new line. "neat" will produce an output somewhere between "compact" and "expanded".
- **quiet**: TRUE or FALSE to print code to console

Value

A string of code to reproduce the input object.

- a string of code to reproduce the model
- a string of code to reproduce the model sequence
- a string of code to reproduce the iterator

Examples

```r
M = example_model(value_1 = 10)
as.code(M)
M = example_model()
as.code(M)
M = example_model()
as.code(M)
M = example_model()
as.code(M)
```

---

**as.DatasetExperiment**  
*Convert a SummarizedExperiment to DatasetExperiment*

Description

Converts a SummarizedExperiment to DatasetExperiment. The assay data is transposed, and colData and rowData switched to match. struct specific slots such as "name" and "description" are extracted from the metaData.
Usage

```r
as.DatasetExperiment(obj)
```

Arguments

- `obj`: a SummarizedExperiment object

Value

- a DatasetExperiment object

Description

Convert a SummarizedExperiment to DatasetExperiment

The assay data is transposed, and colData and rowData switched to match. struct specific slots such as "name" and "description" are extracted from the metaData if available. NB Any additional metadata will be lost during this conversion.

Usage

```r
## S4 method for signature 'SummarizedExperiment'
as.DatasetExperiment(obj)
```

Arguments

- `obj`: a SummarizedExperiment object

Value

- a DatasetExperiment object

Description

Converts a DatasetExperiment to SummarizedExperiment. The assay data is transposed, and colData and rowData switched to match. struct specific slots such as "name" and "description" are stored in the metaData.
Usage

as.SummarizedExperiment(obj)

Arguments

obj a DatasetExperiment object

Value

a SummarizedExperiment object

---

as.SummarizedExperiment,DatasetExperiment-method

*Convert a DatasetExperiment to SummarizedExperiment*

---

Description

Converts a DatasetExperiment to SummarizedExperiment. The assay data is transposed, and col-
Data and rowData switched to match. struct specific slots such as "name" and "description" are
stored in the metaData.

Usage

```r
## S4 method for signature 'DatasetExperiment'
as.SummarizedExperiment(obj)
```

Arguments

obj a DatasetExperiment object

Value

a SummarizedExperiment object

---

as_data_frame convert to data.frame

---

Description

Most often used with univariate statistics to gather all the different outputs in a consistent format.

Usage

```r
as_data_frame(M, ...)
```
c,ontology_list-method

Arguments

- M  a struct object
- ... other inputs passed through this function

Value

a data.frame containing outputs from an object

c,ontology_list-method
catenate ontology_lists

Description

ontology_lists can be catenated with other ontology lists or with ontology_items.

Usage

## S4 method for signature 'ontology_list'
c(x, ...)

Arguments

- x  an ontology_list()
- ... any number of ontology_list() or ontology_item() objects to catenate

Value

an ontology_list()

calculate  Calculate metric

Description

A class for metrics to assess performance of e.g. models, iterators. Not intended to be called directly, this class should be inherited to provide functionality for method-specific classes.
Usage

\[ \text{calculate}(\text{obj}, \ldots) \]

\[ \text{value}(\text{obj}) \]

\[ \text{value}(\text{obj}) \leftarrow \text{value} \]

\[ \text{max\_length}(\text{obj}) \leftarrow \text{value} \]

\[ \text{metric}(\ldots) \]

\#
S 4 method for signature 'metric'
\[ \text{calculate}(\text{obj}, \text{Y}, \text{Yhat}) \]

\#
S 4 method for signature 'metric'
\[ \text{value}(\text{obj}) \]

\#
S 4 replacement method for signature 'metric'
\[ \text{value}(\text{obj}) \leftarrow \text{value} \]

Arguments

\begin{itemize}
    \item \textbf{obj} \quad \text{a metric object}
    \item \ldots \quad \text{named slots and their values.}
    \item \textbf{value} \quad \text{value}
    \item \textbf{Y} \quad \text{the true class labels}
    \item \textbf{Yhat} \quad \text{the predicted class labels}
\end{itemize}

Value

\[ \text{value} \quad \text{the calculated value of a metric} \]

a metric object

Examples

\begin{verbatim}
MET = \text{metric}()
calculate(MET)
MET = \text{metric}()
M = \text{metric}()
calculate(M, \text{Y}, \text{Yhat})
MET = \text{metric}()
\text{value}(\text{MET})
MET = \text{metric}()
\text{value}(\text{MET}) = 10
\end{verbatim}
chart

Constructor for struct chart objects

Description
A base class in the struct package. Should not be called directly.

Usage
chart(...)

Arguments
... named slots and their values that get passed to struct_class

Details
The chart class provides a template for figures, charts and plots associated with other objects. For example, a DatasetExperiment object could have a histogram plotted for a specified column.
Charts can have parameters but not outputs (other than the figure itself), as chart objects are not intended to be used for calculations. The chart_plot method can be used to display a chart for an object, and chart_names can be used to list all chart objects associated with an object.
Classes that inherit the stato class have STATO integration enabled, allowing stato_id to be set and formal names and descriptions pulled from the STATO ontology database.

Value
a chart object
a struct_class object

Examples
C = example_chart()

chart_names

chart names

Description
Returns a list of valid charts for a struct object

Usage
chart_names(obj, ret = "char")

## S4 method for signature 'struct_class'
chart_names(obj, ret = "char")
Arguments

obj  An object derived from the struct_class object
ret  A string indicating whether a list of objects ('obj') or a list of chart names ('char') is returned. 'char' is default.

Details

The chart_names method searches for chart objects associated with the input object.

Value

list of chart names, or a list of chart objects

Examples

M = example_model()
chart_names(M)  # 'example_chart'
chart_names(M,'char')  # as above
chart_names(M,'obj')  # returns a list of chart objects

Description

Plots a chart object

Usage

chart_plot(obj, dobj, ...)

## S4 method for signature 'chart,ANY'
chart_plot(obj, dobj)

Arguments

obj  A chart object
dobj  An object derived from struct_class
...  optional inputs

Details

The optional optional inputs depend on the input object/chart, but might include an additional dataset object or a second model object, for example.

Value

a plot object
Methods (by class)

- chart_plot(obj = chart, dobj = ANY):

Examples

C = example_chart()
chart_plot(C, iris_DatasetExperiment())

citations

Citations for an object

Description

All struct objects have a "citations" slot, which is a list of references in bibtex format. The citations method gathers citations from an object and all struct objects that it inherits to generate a complete list.

Usage

citations(obj)

## S4 method for signature 'struct_class'
citations(obj)

Arguments

obj a struct object

Value

a character array of citations

Examples

D = iris_DatasetExperiment()
D$citations # the list specifically defined for this object
citations(D) # the list for this object and all inherited ones
DatasetExperiment

DatasetExperiment class

Description
An object for holding raw data and associated meta data

Usage
DatasetExperiment(
  data = data.frame(),
  sample_meta = data.frame(),
  variable_meta = data.frame(),
  ...
)

## S4 method for signature 'DatasetExperiment'
x$name

## S4 replacement method for signature 'DatasetExperiment'
x$name <- value

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>A data frame with samples in rows and features in columns</td>
</tr>
<tr>
<td>sample_meta</td>
<td>A data frame with samples in rows and meta data in columns</td>
</tr>
<tr>
<td>variable_meta</td>
<td>A data frame with features in rows and meta data in columns</td>
</tr>
<tr>
<td>...</td>
<td>named slot values to pass through to struct_class</td>
</tr>
<tr>
<td>x</td>
<td>A DatasetExperiment object</td>
</tr>
<tr>
<td>name</td>
<td>DatasetExperiment slot to get/set</td>
</tr>
<tr>
<td>value</td>
<td>the value to assign to the named slot</td>
</tr>
</tbody>
</table>

Details

The DatasetExperiment object is an extension of the SummarizedExperiment object from the SummarizedExperiment package (found on Bioconductor). It incorporates the basic functionality of struct objects, containing fields such as Description, Name and Type with features of SummarizedExperiment such as subsetting.

There are some important differences between DatasetExperiment and SummarizedExperiment:

- In DatasetExperiment data is stored as Samples (rows) x Features (columns)
- DatasetExperiment currently only supports a single assay
- length(DatasetExperiment) returns the number of samples
**Value**

DatasetExperiment

**Slots**

- **name**: Name of the dataset
- **description**: Brief description of the dataset
- **type**: The type of dataset e.g. single_block

---

**entity_stato**

**entity_stato class**

---

**Description**

A base class in the **struct** package. Should not be called directly.

**Usage**

```r
entity_stato(
     name,
     description = character(0),
     type = "character",
     value = NULL,
     max_length = Inf,
     stato_id
)
```

**Arguments**

- **name**: the name of the object
- **description**: a description of the object
- **type**: the type of the struct object
- **value**: The value of the parameter/outputs
- **max_length**: Maximum length of value vector (default 1)
- **stato_id**: The STATO ID for the entity

**Details**

Extends the entity class to include stato functionality.

**Value**

an entity_stato object
See Also

Refer to entity and stato for further info.

Examples

```r
E = entity_stato(
    name = 'example',
    description = 'this is an example',
    type = 'numeric',
    value = 1,
    stato_id='XYZ000001'
)
```

---

### enum

**Enum objects**

Description

A base class in the struct package. Not normally called directly.

Usage

```r
def enum(
    name,
    description = character(0),
    type = "character",
    value = character(0),
    max_length = 1,
    allowed,
    ...
)
```

```r
# S4 replacement method for signature 'enum'
value(obj) <- value
```

Arguments

- **name**: the name of the object
- **description**: a description of the object
- **type**: the type of the struct object
- **value**: value of the enum
- **max_length**: Maximum length of value vector (default 1)
- **allowed**: A list of allowed values
- **...**: additional inputs to the struct_class object
- **obj**: an enum object
Details

An enum object is a special type of entity object that ensures the value must be one from a list of allowed values.

Enum objects are usually defined in the prototype of another object, but can be extracted using `param_obj` and `output_obj`.

Value

an enum object

Examples

```r
# Create a new enum object
E = enum(
    name = 'example',
    description = 'this is an example',
    type = 'character',
    value = 'hello',
    allowed = c('hello', 'world')
)

# Get/set the value of the entity object
value(E)
value(E) = 'world'
```

---

enum_stato enum_stato class

Description

A base class in the `struct` package. Should not be called directly.

Usage

```r
enum_stato(
    name,
    description = character(0),
    type = "character",
    value = character(0),
    max_length = 1,
    allowed,
    stato_id
)
```
Arguments

name  the name of the object
description  a description of the object
type  the type of the struct object
value  The value of the parameter/outputs
max_length  Maximum length of value vector (default 1)
allowed  A list of allowed values
stato_id  The STATO ID for the entity

Details

Extends the \texttt{enum} class to include stato functionality.

Value

an \texttt{enum\_stato} object

See Also

Refer to \texttt{enum} and \texttt{stato} for further info.

Examples

\begin{verbatim}
E = enum\_stato(
name='example',
allowed=list('choice\_1','choice\_2'),
value='choice\_1',
type='character',
stato\_id='XYZ000001'
)
\end{verbatim}

---

**example\_chart**  
example chart object

Description

an example of a chart object for documentation purposes

Usage

\begin{verbatim}
example\_chart(…)
\end{verbatim}

## S4 method for signature 'example\_chart,example\_model'
chart\_plot(obj, dobj)
**Arguments**

... named slots and their values.

- **obj** a chart object
- **dobj** an example_model object

**Value**

- a chart object

**Examples**

```r
C = example_chart()
chart_plot(C, example_model())
```

---

**example_iterator-class**

*Example iterator*

**Description**

An example iterator for testing runs the example iterator, which just returns a value of 3.142

**Usage**

```r
## S4 method for signature 'example_iterator,DatasetExperiment,metric'
run(I, D, MET)
```

**Arguments**

- **I** example_iterator object
- **D** dataset object
- **MET** metric object

**Value**

- test iterator object
- dataset object

**Examples**

```r
I = example_iterator()
I = example_iterator()
D = iris_DatasetExperiment()
MET = metric()
I = run(I, D, MET)
```
**example_model**  

**Example model**

**Description**

An example model for testing. Training this model adds value_1 to a data set, and prediction using this model adds value_2.

- trains the example model, which adds value_1 to the raw data of a dataset
- predicts using the example model, which adds value_2 to the raw data of a dataset

**Usage**

```r
example_model(value_0 = 0, value_1 = 10, value_2 = 20, ...)
```

```r
## S4 method for signature 'example_model,DatasetExperiment'
model_train(M, D)
```

```r
## S4 method for signature 'example_model,DatasetExperiment'
model_predict(M, D)
```

**Arguments**

- **value_0** a numeric value
- **value_1** a numeric value
- **value_2** a numeric value
- **...** named slots and their values.
- **M** A struct model object
- **D** A DatasetExperiment object

**Value**

- modified example_model object
- dataset object
- dataset object

**Examples**

```r
M = example_model()
M = example_model(value_1 = 10, value_2 = 20)
D = iris_DatasetExperiment()
M = example_model(value_1 = 10, value_2 = 20)
M = model_train(M, D)
D = iris_DatasetExperiment()
M = example_model(value_1 = 10, value_2 = 20)
M = model_predict(M, D)
```
**export_xlsx**

**write a dataset object to file**

**Description**

Exports a dataset object to an excel file with sheets for data, sample_meta and variable_meta

**Usage**

```r
export_xlsx(object, outfile, transpose = TRUE)
```

## S4 method for signature 'DatasetExperiment'

```r
export_xlsx(object, outfile, transpose = TRUE)
```

**Arguments**

- **object**: a dataset object
- **outfile**: the filename (including path) to write the data to
- **transpose**: TRUE (default) or FALSE to transpose the output data

**Value**

an excel file with sheets for data and meta data

**Examples**

```r
## Not run:
D = iris_DatasetExperiment() # example dataset
export_xlsx(D,'iris_DatasetExperiment.xlsx')

## End(Not run)
```

**get_description**

**Get struct object help description**

**Description**

This function is to help developers including struct objects in their own R packages, and isn’t intended for general use. Use with roxygen 2 '@eval' tags this function generates a detailed description of a struct object generated by extracting names, descriptions etc from slots in a suitable format.

**Usage**

```r
get_description(id)
```
Arguments

id (character) the name of a struct object to generate documentation for

Value

a character string of roxygen formatted documentation for the object

Examples

get_description('example_model')

---

iris_DatasetExperiment

*Fisher’s Iris data*

---

Description

Fisher’s Iris data as a DatasetExperiment object

Usage

iris_DatasetExperiment()

Value

DatasetExperiment object

Examples

D = iris_DatasetExperiment()

---

is_output

*Verify output*

---

Description

Verify that the name of a output is valid for an object

Usage

is_output(obj, name)

### S4 method for signature 'struct_class'

is_output(obj, name)
**is_param**

**Arguments**

- **obj**  
  A model or iterator object derived from the *struct* class
- **name**  
  Name of output

**Value**

TRUE if output name is valid, FALSE if not

**Examples**

```r
M = example_model()
is_output(M, 'result_1')  # TRUE
is_output(M, 'result_0')  # FALSE
```

---

**is_param**  
**Verify parameter**

**Description**

Verify that the input name is a valid input parameter for an object

**Usage**

```r
is_param(obj, name)
```

---

## S4 method for signature 'struct_class'

```r
is_param(obj, name)
```

**Arguments**

- **obj**  
  An object derived from struct_class
- **name**  
  Name of parameter

**Value**

TRUE if parameter name is valid, FALSE if not

**Examples**

```r
M = example_model()
is_param(M, 'value_1')  # TRUE
is_param(M, 'alpha')   # FALSE
```
libraries | Libraries for an object

Description

All struct objects have a "libraries" slot, which is a character array of libraries required to use the object. The libraries method gathers libraries from an object and all struct objects that it inherits to generate a complete list.

Usage

libraries(obj)

## S4 method for signature 'struct_class'
libraries(obj)

Arguments

obj a struct object

Value

a character array of R packages needed by the object

Examples

M = example_model()
libraries(M)

max_length | get the max value vector length for an entity

Description

A base class in the struct package. Not normally called directly. An entity object is used to store information about a parameter or output. The standard 'name', 'description', and 'type' slots are included, along with 'value' for storing the value of the parameter and 'max_length' for restricting the length of 'value' if needed.
Usage

max_length(obj)

describe_entity(
    name,
    description = character(0),
    type = "character",
    value = NULL,
    max_length = Inf,
    ...
)

## S4 method for signature 'entity'
value(obj)

## S4 replacement method for signature 'entity'
value(obj) <- value

## S4 method for signature 'entity'
max_length(obj)

## S4 replacement method for signature 'entity'
max_length(obj) <- value

Arguments

obj An entity object
name the name of the object
description a description of the object
type the type of the struct object
value The value of the parameter/outputs
max_length Maximum length of value vector (default 1)
... additional inputs to the struct_class object

Details

Entity objects are usually defined in the prototype of another object, but can be extracted using param_obj and output_obj.

Value

max value vector length for an entity
An entity object
Examples

```r
# Create a new entity object
E = entity(
  name = 'example',
  description = 'this is an example',
  type = 'numeric',
  value = 1
)
```

```r
# Get/set the value of the entity object
value(E)
value(E) = 10
```

---

### model

#### model class

**Description**

A class for models that can be trained/applied to datasets e.g. PCA, PLS etc. Also used for preprocessing steps that require application to test sets. not intended to be called directly, this class should be inherited to provide functionality for method-specific classes.

**Usage**

```r
model(
  predicted = character(0),
  seq_in = "data",
  seq_fcn = function(x) {
    return(x)
  },
  ...
)
```

```r
## S4 method for signature 'model,DatasetExperiment'
model_train(M, D)
```

```r
## S4 method for signature 'model,DatasetExperiment'
model_predict(M, D)
```

```r
## S4 method for signature 'model,DatasetExperiment'
model_apply(M, D)
```

```r
## S4 method for signature 'model,DatasetExperiment'
model_reverse(M, D)
```

```r
## S4 method for signature 'model'
predicted(M)
```
## S4 method for signature 'model'
seq_in(M)

## S4 replacement method for signature 'model,character'
seq_in(M) <- value

## S4 method for signature 'model'
predicted_name(M)

## S4 replacement method for signature 'model,character'
predicted_name(M) <- value

### Arguments

- **predicted**
  The name of an output slot to return when using `predicted()` (see details)

- **seq_in**
  The name of an output slot to connect with the "predicted" output of another model (see details)

- **seq_fcn**
  A function to apply to seq_in before inputting into the next model. Typically used to extract a single column, or convert from factor to char etc.

- **...**
  Named slots and their values.

- **M**
  A struct model object

- **D**
  A DatasetExperiment object

- **value**
  The value to assign

### Value

- **trained model object**
  A trained model object with test set results

- **dataset object**
  A dataset object with the reverse model applied

- **predicted output**
  The predicted output, as specified by `predicted_name`

- **id of input parameter**
  The id of the input parameter to be replaced by the predicted output of the previous model in a model sequence. Reserved keyword 'data' means that the input data used by `model_train`, `model_apply` etc is used. `seq_in = 'data'` is the default setting.

- **modified model object**
  The modified model object

- **id of output returned by predicted()**
  The id of the output returned by `predicted()`

- **modified model object**

### predicted slot

The "predicted" slot is a slot for use by users to control the flow of model sequences. The `predicted()` function is used to return a default output and from a model. Typically it is a DatasetExperiment object that is passed directly into the next model in a sequence as the data for that model.
seq_in slot

In a sequence of models (see model_seq) the "predicted" slot is connected to the DatasetExperiment input of the next model. seq_in can be used to control flow and connect the "predicted" output to the input parameter of the next model. Default is the keyword 'data', and can otherwise be replaced by any input slot from the model. The slot seq_fcn can be used to apply a transformation to the output before it is used as an input. This allows you to e.g. convert between types, extract a single column from a data.frame etc.

Examples

```r
M = model()
D = DatasetExperiment()
M = model()
M = model_train(M,D)
D = DatasetExperiment()
M = model()
M = model_train(M,D)
M = model_predict(M,D)
D = DatasetExperiment()
M = model()
M = model_apply(M,D)
D = DatasetExperiment()
M = model()
M = model_train(M,D)
M = model_predict(M,D)
M = model_reverse(M,D)
D = DatasetExperiment()
M = example_model()
M = model_train(M,D)
M = model_predict(M,D)
p = predicted(M)
D = DatasetExperiment()
M = example_model()
seq_in(M) = 'data'
M = example_model()
seq_in(M) = 'value_1'
M = example_model()
predicted_name(M)
M = example_model()
predicted_name(M) = 'result_2'
```

models

Get/set models of a model_seq

Description

Returns the list of models in a model_seq object
model_apply

Usage

models(ML)

models(ML) <- value

Arguments

ML a model_seq object
value a list containing only model objects

Value

models(ML) returns a list of models in the model sequence
models(ML)<- sets the list of models in the model sequence

Examples

# Create a model sequence
ML = model_seq()
models(ML) = list(example_model(), example_model())
models(ML)

model_apply Apply a model

Description

Applies a method to the input dataset

Usage

model_apply(M, D, ...)

Arguments

M a 'method' object
D another object used by the first
... other optional inputs

Value

Returns a modified method object

Examples

M = example_model()
M = model_apply(M, iris.DatasetExperiment())
**model_predict**  
*Model prediction*

**Description**  
Apply a model using the input dataset. Assumes the model is trained first.

**Usage**  
```r
model_predict(M, D, ...)
```

**Arguments**
- `M` a model object
- `D` a dataset object
- `...` other optional inputs

**Value**
Returns a modified model object

**Examples**
```r
M = example_model()
M = model_predict(M, iris_DatasetExperiment())
```

---

**model_reverse**  
*Reverse preprocessing*

**Description**  
Reverse the effect of a preprocessing step on a dataset.

**Usage**  
```r
model_reverse(M, D, ...)
```

**Arguments**
- `M` a model object
- `D` a dataset object
- `...` other optional inputs

**Value**
Returns a modified dataset object
Examples

M = example_model()
D = model_reverse(M, iris_DatasetExperiment())

Description

A class for (ordered) lists of models

Usage

model_seq(...)

## S4 method for signature 'model_seq,DatasetExperiment'
model_train(M, D)

## S4 method for signature 'model_seq,DatasetExperiment'
model_predict(M, D)

## S4 method for signature 'model_seq,ANY,ANY,ANY'
x[i]

## S4 replacement method for signature 'model_seq,ANY,ANY,ANY'
x[i] <- value

## S4 method for signature 'model_seq'
models(ML)

## S4 replacement method for signature 'model_seq,list'
models(ML) <- value

## S4 method for signature 'model_seq'
length(x)

## S4 method for signature 'model,model_seq'
e1 + e2

## S4 method for signature 'model_seq,model'
e1 + e2

## S4 method for signature 'model,model'
e1 + e2

## S4 method for signature 'model_seq'
predicted(M)
## S4 method for signature 'model_seq,DatasetExperiment'

```r
model_apply(M, D)
```

### Arguments

- `...`: named slots and their values.
- `M`: a model object
- `D`: a dataset object
- `x`: a model_seq object
- `i`: index
- `value`: value
- `ML`: a model_seq object
- `e1`: a model or model_seq object
- `e2`: a model or model_seq object

### Value

- model sequence
- model sequence
- model at the given index in the sequence
- model sequence with the model at index i replaced
- a list of models in the sequence
- a model sequence containing the input models
- the number of models in the sequence
- a model sequence with the additional model appended to the front of the sequence
- a model sequence with the additional model appended to the end of the sequence
- a model sequence
- the predicted output of the last model in the sequence

### Examples

```r
MS = model_seq()
MS = model() + model()
MS = example_model() + example_model()
MS = model_train(MS, DatasetExperiment())
D = DatasetExperiment()
MS = example_model() + example_model()
MS = model_train(MS, D)
MS = model_predict(MS, D)
MS = model() + model()
MS[2]
MS = model() + model()
```
model_train

Train a model

Description

Trains a model using the input dataset

Usage

model_train(M, D, ...)

Arguments

M  a model object
D  a dataset object
... other optional inputs
Value

Returns a modified model object

Examples

```r
M = example_model()
M = model_train(M, iris_DatasetExperiment())
```

---

**new_struct**

*Generate a struct object from a Class*

Description

This function creates a newly allocated object from the class identified by the first argument. It works almost identically to `new` but is specific to objects from the *struct* package and ensures that entity slots have their values assigned correctly. This function is usually called by class constructors and not used directly.

Usage

`new_struct(class, ...)`

Arguments

- `class` The class of struct object to create
- `...` named slots and values to assign

Value

An object derived from `struct_class`

Examples

```r
S = new_struct('struct_class')
```
Ontology for an object

Description

All struct objects have an "ontology" slot, which is a list of ontology items for the object. The ontology method gathers ontology items from an object and all struct objects that it inherits to generate a complete list.

A base class in the struct package. Stores ontology information e.g. term, description, id etc for struct objects and provides methods for populating these fields using the ‘rols’ package.

A base class in the struct package. Stores multiple ‘ontology_term’ objects.

Usage

ontology(obj, cache = NULL)

ontology_term(
  id,
  ontology = character(),
  label = character(),
  description = character(),
  iri = character(),
  rols = TRUE
)

ontology_list(terms = list())

# S4 method for signature 'ontology_list,ANY,ANY,ANY'
x[i]

# S4 replacement method for signature 'ontology_list,ANY,ANY,ANY'
x[i] <- value

# S4 method for signature 'ontology_list'
length(x)

# S4 method for signature 'struct_class'
ontology(obj, cache = NULL)

Arguments

obj  a struct object

cache  a named list of ontology_terms for offline use. Terms from the cache are search based on the name of the list items matching the ontology id. If cache=NULL then the OLS API is used to lookup terms.

id  (character) The ontology term id e.g. 'STATO:0000555'
ontology (character) The ontology the term is a member of e.g. 'stato'
label (character) The label for the ontology term
description (character) The description of the term
iri (character) The Internationalized Resource Identifier for the term
rots (logical) TRUE or FALSE to query the Ontology Lookup Service for missing label, description or iri if not provided as input. Default rols = TRUE
terms A list of ontology_term objects.
x the list
i The list item index
value an ontology_term() object

Value
model at the given index in the sequence
model sequence with the model at index i replaced
the number of models in the sequence

Examples
M = example_model()
ontology(M,cache=NULL)
## Not run:
OT = ontology_term(id='STATO:0000555')

## Not run:
## Not run:
OT = ontology_list(terms=list(
   ontology_term(ontology='obi',id = 'OBI:0200051'),
   ontology_term(ontology='stato',id = 'STATO:0000555')
))

## Not run:
## Not run:
OL = ontology_list('STATO:0000555')
OL[1]

## Not run:
## Not run:
OL = ontology_list('STATO:0000555')
OL[1] = ontology_term('STATO:0000302')

## Not run:
## Not run:
OL = ontology_list()
length(OL) # 0

## End(Not run)
**optimiser**

---

**optimiser**  
**optimiser class**

---

**Description**

A special class of iterator for selecting optimal parameter values not intended to be called directly, this class should be inherited to provide functionality for method-specific classes.

**Usage**

```r
optimiser(...)  
```

**Arguments**

... named slots and their values.

**Value**

an optimiser object

**Examples**

```r
OPT = optimiser()
```

---

**output_ids**  
**Output identifiers**

---

**Description**

return a list of valid output ids for an object

**Usage**

```r
output_ids(obj)
```

```r
## S4 method for signature 'struct_class'
output_ids(obj)
```

**Arguments**

`obj` A model or iterator object derived from the *struct* class

**Value**

list of output ids
Example

\begin{verbatim}
M = example_model()
output_ids(M)
\end{verbatim}

---

output_list output list

Description

get/set a named list of outputs and their current value for an object

Usage

\begin{verbatim}
output_list(obj)
output_list(obj) <- value
\end{verbatim}

## S4 method for signature 'struct_class'
output_list(obj)

## S4 replacement method for signature 'struct_class,list'
output_list(obj) <- value

Arguments

obj An object derived from struct_class
value A named list of outputs and corresponding values

Value

A named list of outputs and corresponding values
struct object

Examples

\begin{verbatim}
M = example_model()
L = output_list(M)
M = example_model()
output_list(M) = list('result_1' = DatasetExperiment(),'result_2' = DatasetExperiment())
\end{verbatim}
**output_name**

**Description**

return a the name for a output, if available

**Usage**

output_name(obj, name)

```r
## S4 method for signature 'struct_class,character'
output_name(obj, name)
```

**Arguments**

- **obj**
  - A model or iterator object derived from the *struct* class
- **name**
  - Name of output

**Value**

name of output

**Examples**

```r
M = example_model()
output_name(M,'result_1')
```

---

**output_obj**

**Description**

Gets or sets the object of an output e.g. to an entity() object.

**Usage**

output_obj(obj, name)

output_obj(obj, name) <- value

```r
## S4 method for signature 'struct_class,character'
output_obj(obj, name)
```

```r
## S4 replacement method for signature 'struct_class,character'
output_obj(obj, name) <- value
```
Arguments

- **obj**: A model or iterator object derived from the *struct* class
- **name**: Name of output
- **value**: A valid value for the output being set

Value

- `output_obj(M, name)` returns the named output as an object
- `output_obj(M, name) <- value` sets the named output of an object the modified object

Examples

```r
# get the output as an object
M = example_model()
obj = output_obj(M, 'result_1')

# set a output as an object
output_obj(M, 'result_1') = entity(value = 15, type = 'numeric', name = 'result_1')
```

Description

get/set the values for an output

Usage

```r
output_value(obj, name)

output_value(obj, name) <- value

## S4 method for signature 'struct_class,character'
output_value(obj, name)

## S4 replacement method for signature 'struct_class,character'
output_value(obj, name) <- value
```

Arguments

- **obj**: A model or iterator object derived from the *struct* class
- **name**: Name of output
- **value**: A valid value for the output being set
param_ids

Value

Value of output
struct object

Examples

M = example_model()
output_value(M, 'result_1')
M = example_model()
output_value(M, 'result_1') = DatasetExperiment()

<table>
<thead>
<tr>
<th>param_ids</th>
<th>Parameter identifiers</th>
</tr>
</thead>
</table>

Description

return a list of valid parameter ids for an object

Usage

param_ids(obj)

## S4 method for signature 'struct_class'
param_ids(obj)

Arguments

obj An object derived from struct_class

Value

list of parameter ids

Examples

M = example_model()
param_ids(M)
Description
get/set a named list of parameters and their current value for an object

Usage

param_list(obj)

param_list(obj) <- value

## S4 method for signature 'struct_class'
param_list(obj)

## S4 replacement method for signature 'struct_class,list'
param_list(obj) <- value

Arguments

obj An object derived from struct_class
value A named list of parameters and corresponding values

Value
A named list of parameters names and corresponding values

Examples

M = example_model()
L = param_list(M)

M = example_model()
param_list(M) = list('value_1' = 15,'value_2' = 20)

Description
Returns the name for a parameter, if available
Usage

param_name(obj, name)

### S4 method for signature 'struct_class,character'

param_name(obj, name)

Arguments

obj An object derived from struct_class
name Name of parameter

Value

name of parameter

Examples

M = example_model()
param_name(M, 'value_1')

param_obj

Parameter objects

Description

Gets or sets the object of a parameter e.g. to an entity() object.

Usage

param_obj(obj, name)

param_obj(obj, name) <- value

### S4 replacement method for signature 'struct_class,character'

param_obj(obj, name) <- value

### S4 method for signature 'struct_class,character'

param_obj(obj, name)

Arguments

obj An object derived from struct_class
name Name of parameter
value A valid value for the parameter being set
Value

\texttt{param_obj(M, name)} Returns the named parameter as an object
\texttt{param_obj(M, name)}<- Sets the named parameter of an object

Examples

# get the parameter as an object
M = example_model()
obj = param_obj(M, 'value_0')

# set a parameter as an object
param_obj(M, 'value_0') = entity(value = 15, type = 'numeric', name = 'value_0')

---

**param_value**  

**Parameter values**

Description

get/set the values for a parameter.

Usage

\texttt{param_value(obj, name)}

\texttt{param_value(obj, name)} <- value

### S4 method for signature 'struct_class, character'
\texttt{param_value(obj, name)}

### S4 replacement method for signature 'struct_class, character'
\texttt{param_value(obj, name)} <- value

Arguments

\begin{itemize}
  \item \texttt{obj} A model or iterator object derived from structclass
  \item \texttt{name} Name of parameter
  \item \texttt{value} A valid value for the parameter being set
\end{itemize}

Value

Value of parameter
**Examples**

```r
M = example_model()
param_value(M, 'value_1')

M = example_model()
param_value(M, 'value_1') = 0.95
```

<table>
<thead>
<tr>
<th>predicted</th>
<th>Prediction output</th>
</tr>
</thead>
</table>

**Description**

returns the prediction output for a model. This is supplied as input to the next model when used in a model_seq.

**Usage**

```r
predicted(M)
```

**Arguments**

- **M**
  - a model object

**Value**

The value returned varies depending on the output.

**Examples**

```r
M = example_model()
M = model_train(M, iris_DatasetExperiment())
M = model_predict(M, iris_DatasetExperiment())
predicted(M)
```

<table>
<thead>
<tr>
<th>predicted_name</th>
<th>Predicted output name</th>
</tr>
</thead>
</table>

**Description**

get/set the prediction output for a model. This determines which outputs from this model are supplied as inputs to the next model when used in a model_seq.

**Usage**

```r
predicted_name(M)
predicted_name(M) <- value
```
preprocess

Arguments

- M: a model object
- value: name of an output for this model

Value

- predicted_name: returns the name of the predicted output
- predicted_name<-: sets the name of the predicted output

Examples

```r
M = example_model()
predicted_name(M)
predicted_name(M) = 'result_2'
```

Description

A class used for preprocessing steps that require application to test sets. Not intended to be called directly, this class should be inherited to provide functionality for method-specific classes.

Usage

```r
preprocess(...)  
```

```r
## S4 method for signature 'preprocess,DatasetExperiment'
model_reverse(M, D)
```

Arguments

- ...: named slots and their values.
- M: a model object
- D: a dataset object

Value

- dataset object

Examples

```r
M = preprocess()  
D = DatasetExperiment()  
M = model()  
D2 = model_reverse(M, D)
```
**resampler**

**resampler class**

**Description**

A class for resampling methods such as cross-validation. not intended to be called directly.

**Usage**

`resampler(...)`

**Arguments**

... named slots and their values.

**Value**

a resampler object

**Examples**

```r
R = resampler()
```

---

**result**

**Iterator result**

**Description**

Returns the results of an iterator. This is used to control model flow in a similar way to `predict` for model and model_seq objects.

**Usage**

`result(M)`

**Arguments**

`M` an iterator object

**Value**

the returned output varies with the algorithm implemented
result_name

**Examples**

```r
D = iris.DatasetExperiment() # get some data
MET = metric() # use a metric
I = example_iterator() # initialise iterator
models(I) = example_model() # set the model
I = run(I,D,MET) # run
result(I)
```

**Description**

get/set the prediction output for a model. This determines which outputs from this model are supplied as inputs to the next model when used in a `model_seq`.

**Usage**

```r
result_name(M)
result_name(I) <- value
```

**Arguments**

- **M**: an iterator object
- **I**: an iterator object
- **value**: name of an output for iterator M

**Value**

- `result_name(M)` returns the name of the output for this iterator (equivalent to `predicted` for model objects)
- `result_name(I) <- value` sets the default output for an iterator

**Examples**

```r
I = example_iterator() # initialise iterator
result_name(I)
result_name(I) = 'result_1'
```
Run iterator

Description

Runs an iterator, applying the chosen model multiple times.

Evaluates an iterator by e.g. averaging over all iterations. May be deprecated in a future release as evaluate is applied by run anyway.

A class for iterative approaches that involve the training/prediction of a model multiple times. Not intended to be called directly, this class should be inherited to provide functionality for method-specific classes.

Usage

run(I, D, MET)

evaluate(I, MET)

iterator(...)

## S4 method for signature 'iterator,DatasetExperiment,metric'
run(I, D, MET = NULL)

## S4 method for signature 'iterator,metric'
evaluate(I, MET)

## S4 method for signature 'iterator'
models(ML)

## S4 replacement method for signature 'iterator,model_OR_iterator'
models(ML) <- value

## S4 replacement method for signature 'iterator,character'
result_name(I) <- value

## S4 method for signature 'iterator'
result(M)

## S4 method for signature 'iterator'
result_name(M)

## S4 method for signature 'iterator,model_OR_iterator'
e1 * e2

## S4 method for signature 'iterator,ANY,ANY,ANY'
x[i]
## S4 replacement method for signature 'iterator,ANY,ANY,ANY'

```r
x[i] <- value
```

### Arguments

- **I**: an iterator object
- **D**: a dataset object
- **MET**: a metric object
- **ML**: a model sequence object
- **...**: named slots and their values.
- **value**: value
- **M**: a model object
- **e1**: an iterator object
- **e2**: an iterator or a model object
- **x**: a sequence object
- **i**: index into sequence

### Details

Running an iterator will apply the iterator a number of times to a dataset. For example, in cross-validation the same model is applied multiple times to the same data, splitting it into training and test sets. The input metric object can be calculated and collected for each iteration as an output.

### Value

Modified iterator object

- Modified iterator object
- the modified model object
- model at the given index in the sequence
- iterator with the model at index i replaced

### Examples

```r
D = iris_DatasetExperiment() # get some data
MET = metric() # use a metric
I = example_iterator() # initialise iterator
models(I) = example_model() # set the model
I = run(I,D,MET) # run
D = iris_DatasetExperiment() # get some data
MET = metric() # use a metric
I = example_iterator() # initialise iterator
models(I) = example_model() # set the model
I = run(I,D,MET) # run
I = evaluate(I,MET) # evaluate
I = iterator()
```
I = iterator() * model()
D = DatasetExperiment()
MET = metric()
I = iterator() * model()
I = run(I,D,MET)

I = iterator()
result_name(I) = 'example'
MS = model() + model()
I = iterator() * MS
I[2] # returns the second model() object

MS = model() + model()
I = iterator() * MS
I[2] = model() # sets the second model to model()

---

**seq_in**

*Sequence input*

**Description**

get/set the input parameter replaced by the output of the previous model in a model sequence. Default is "data" which passes the output as the data input for methods such as model_train and model_apply.

**Usage**

seq_in(M)

seq_in(M) <- value

**Arguments**

<table>
<thead>
<tr>
<th>M</th>
<th>a model object</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>name of an output for this model</td>
</tr>
</tbody>
</table>

**Value**

seq_in returns the name of the input parameter replaced when used in a model sequence
seq_in<- sets the name of the input parameter replaced when used in a model sequence

**Examples**

M = example_model()
seq_in(M)
seq_in(M) = 'value_1'
**set_obj_method**

update method for a struct object

**Description**

a helper function to update methods for a struct object

**Usage**

```r
definition = c(class_name, "DatasetExperiment")
set_obj_method(
  class_name, method_name, definition,
  where = topenv(parent.frame()),
  signature = c(class_name, "DatasetExperiment")
)
```

**Arguments**

- `class_name` - the name of the to update the method for
- `method_name` - the name of the method to update. Must be an existing method for the object.
- `definition` - the function to replace the method with. This function will be used when the method is called on the object.
- `where` - the environment to create the object in. default where = topenv(parent.frame())
- `signature` - a list of classes that this object requires as inputs. Default is c(class_name,'DatasetExperiment')

**Value**

a method is created in the specified environment

**Examples**

```r
set_struct_obj(
  class_name = 'add_two_inputs',
  struct_obj = 'model',
  params = c(input_1 = 'numeric', input_2 = 'numeric'),
  outputs = c(result = 'numeric'),
  prototype = list(
    input_1 = 0,
    input_2 = 0,
    name = 'Add two inputs',
    description = 'example class that adds two values together')
)
```
set_obj_show

**set_obj_show**

*a helper function to update the show method for a struct object*

**Description**

a helper function to update the show method for a struct object

**Usage**

```r
description
set_obj_show(class_name, extra_string, where = toplevel(parent.frame()))
```

**Arguments**

- **class_name**: the name of the to update the method for
- **extra_string**: a function that returns an extra string using the input object as an input e.g.
  ```r
  function(object) return = 'extra_string'
  ```
- **where**: the environment to create the object in. default where = toplevel(parent.frame())

**Value**

a method is created in the specified environment

**Examples**

```r
# create an example object first
set_struct_obj(
  class_name = 'add_two_inputs',
  struct_obj = 'model',
  params = c(input_1 = 'numeric', input_2 = 'numeric'),
  outputs = c(result = 'numeric'),
  prototype = list(
    input_1 = 0,
    input_2 = 0,
    name = 'Add two inputs',
    description = 'example class that adds two values together'
  )
)

# now update the method
set_obj_show(
  class_name = 'add_two_inputs',
  extra_string = function(object) (return('The extra text'))
)
```
set_struct_obj  

**Description**

A helper function to create new struct objects

**Usage**

```
set_struct_obj(
  class_name,  # the name of the new class to create
  struct_obj,  # the struct obj to inherit e.g. 'model', 'metric' etc
  params = character(0),  # a named character vector of input parameters where each element specifies the type of value that will be in the slot e.g. c(example = 'character')
  outputs = character(0),  # a named character vector of outputs where each element specifies the type of value that will be in the slot e.g. c(example = 'character')
  private = character(0),  # a named character vector of private slots where each element specifies the type of value that will be in the slot e.g. c(example = 'character'). These are intended for internal use by the object and generally not available to the user.
  prototype = list()  # a named list with initial values for slots.
)
```

**Arguments**

- `class_name`: the name of the new class to create
- `struct_obj`: the struct obj to inherit e.g. 'model', 'metric' etc
- `params`: a named character vector of input parameters where each element specifies the type of value that will be in the slot e.g. c(example = 'character')
- `outputs`: a named character vector of outputs where each element specifies the type of value that will be in the slot e.g. c(example = 'character')
- `private`: a named character vector of private slots where each element specifies the type of value that will be in the slot e.g. c(example = 'character'). These are intended for internal use by the object and generally not available to the user.
- `prototype`: a named list with initial values for slots.

**Value**

A new class definition. To create a new object from this class use `X = new_class_name()`

---

**stato_id**  

**Description**

Get the stato_id for an object

A base class in the `struct` package. Provides several fundamental methods and should not be called directly.
Usage

stato_id(obj)
stato_name(obj)
stato_definition(obj)
stato_summary(obj)
stato(stato_id)

## S4 method for signature 'stato'
stato_id(obj)

## S4 method for signature 'stato'
stato_name(obj)

## S4 method for signature 'stato'
stato_definition(obj)

## S4 method for signature 'stato'
stato_summary(obj)

Arguments

obj An object derived from the stato object
stato_id A STATO ID e.g. OBI:0000001

Details

STATO is the statistical methods ontology. It contains concepts and properties related to statistical methods, probability distributions and other concepts related to statistical analysis, including relationships to study designs and plots (see http://stato-ontology.org/).

This class provides access to a version of the STATO ontology database that can be searched by ontology id to provide formal names and definitions for methods, models, iterators, metrics and charts.

This class makes use of the ontologyIndex package to search a copy of the STATO database included in this package.

Value

id the stato id
name the stato name
def the stato description

Value returned depends on the method used.
Examples

```r
M = example_model()
stato_id(M)
stato_name(M)
stato_definition(M)
stato_summary(M)
# an example stato object
M = example_model()

# the stato id assigned to object M
stato_id(M) # OBI:0000011

# the name associated with that id
stato_name(M)

# the STATO definition for that id
stato_definition(M)

# a summary of the STATO database entry for the id, and any parameters or
# outputs that also have stato ids.
stato_summary(M)
```

---

### struct_class

*Constructor for struct_class objects*

#### Description

Creates a new `struct_class` object and populates the slots. Not intended for direct use.

#### Usage

```r
struct_class(
    name = character(0),
    description = character(0),
    type = character(0),
    citations = list(),
    ontology = character(0)
)
```

#### Arguments

- **name**  
  the name of the object
- **description**  
  a description of the object
- **type**  
  the type of the struct object
- **citations**  
  a list of citations for the object in "bibentry" format
- **ontology**  
  a list of ontology items for the object in "ontology_item" format
**struct_class-class**

**Value**

a struct_class object

---

**struct_class-class**

**struct_class object definition**

**Description**

Defines the struct class base template. This class is inherited by other objects and not intended for direct use. It defines slots and methods common to all struct objects.

**Value**

Returns a struct object

**Public slots**

Public slots can be accessed using shorthand $ notation and are intended for users building workflows.

- **name character()** A short descriptive name of the struct object
- **description character()** A longer description of the struct object and what it does
- **type character()** A keyword that describes the type of struct object
- **libraries character()** A (read only) list of R packages used by this struct object
- **citations list of bibentry** A (read only) list of citations relevant to this struct object, in Bibtex format.

**Private slots**

Private slots are not readily accessible to users and are intended for developers creating their own struct objects. Any slot not listed within `.params` or `.outputs` is considered a private slot.

- **.params character()** A list of additional slot names that can be get/set by the user for a specific struct object. These are used as input parameters for different methods.
- **.outputs character()** a list of additional slot names that can be get by the user. These are used to store the results of a method.

**Examples**

```r
S = struct_class(name = 'Example', description = 'An example object')
```
struct_template  

**Description**

Create a struct template

**Usage**

```r
struct_template(
  template = "model",
  output,
  in_editor = TRUE,
  overwrite = FALSE
)
```

**Arguments**

- `template`: the type of object you want a template for e.g. 'model'
- `output`: the name/path of the output file
- `in_editor`: TRUE/FALSE to open the created file in the default editor
- `overwrite`: = TRUE/FALSE to overwrite file if exists already

**Value**

A template is created at the output location specified

**Examples**

```r
## Not run:
struct_template('model','example.R',FALSE)
## End(Not run)
```

test_metric-class  

**Description**

An example metric for testing

calculates a metric, which just returns a value of 3.142
Usage

```r
## S4 method for signature 'test_metric'
calculate(obj)
```

Arguments

- `obj` metric object

Value

test metric object
dataset object

Examples

```r
MET = test_metric()
MET = test_metric()
MET = calculate(MET)
```

### $,ontology_list-method

*Get/set ontology_list slots*

Description

Dollar syntax can be used to as a shortcut for getting values for ontology_list objects.

Usage

```r
## S4 method for signature 'ontology_list'
x$name
```

Arguments

- `x` An ontology_term object
- `name` The name of the slot to access

Value

Slot value
$ontology_term-method

Get/set ontology term slots

Description

Dollar syntax can be used to as a shortcut for getting values for ontology_term objects.

Usage

```r
## S4 method for signature 'ontology_term'
x$name
```

Arguments

- **x**: An ontology_term object
- **name**: The name of the slot to access

Value

Slot value

Examples

```r
## Not run:
OT = ontology_term(ontology='stato',id='STATO:0000555')

## End(Not run)
```
$.struct_class-method  Get/set parameter or output values

Description

Dollar syntax can be used as a shortcut for getting/setting input parameter and output values for struct objects.

Usage

## S4 method for signature 'struct_class'
x$name

Arguments

x  An object derived from struct_class
name The name of the slot to access

Value

Parameter/output value

Examples

M = example_model()
M$value_1 = 10
M$value_1 # 10

$<-,struct_class-method  Get/set parameter or output values

Description

Dollar syntax can be used as a shortcut for getting/setting input parameter and output values for struct objects.

Usage

## S4 replacement method for signature 'struct_class'
x$name <- value

Arguments

x  An object derived from struct_class
name The name of the slot to access
value The value to assign
Value

Parameter/output value

Examples

M = example_model()
M$\text{value}_1 = 10$
M$\text{value}_1 \# 10
Index

* internal
  struct-package, 3
*,iterator,model_OR_iterator-method (run), 49
+,model,model-method (model_seq), 31
+,model,model_seq-method (model_seq), 31
+,model_seq,model-method (model_seq), 31
..DollarNames,DatasetExperiment-method (.DollarNames.struct_class), 4
..DollarNames.chart-method (.DollarNames.struct_class), 4
..DollarNames,iterator-method (.DollarNames.struct_class), 4
..DollarNames,metric-method (.DollarNames.struct_class), 4
..DollarNames,model-method (.DollarNames.struct_class), 4
..DollarNames,optimiser-method (.DollarNames.struct_class), 4
..DollarNames,preprocess-method (.DollarNames.struct_class), 4
..DollarNames,resampler-method (.DollarNames.struct_class), 4
..DollarNames,struct_class-method (.DollarNames.struct_class), 4
..DollarNames.DatasetExperiment (.DollarNames.struct_class), 4
..DollarNames.chart (.DollarNames.struct_class), 4
..DollarNames.iterator (.DollarNames.struct_class), 4
..DollarNames.metric (.DollarNames.struct_class), 4
..DollarNames.model (.DollarNames.struct_class), 4
..DollarNames.optimiser (.DollarNames.struct_class), 4
..DollarNames.preprocess (.DollarNames.struct_class), 4
..DollarNames.resampler (.DollarNames.struct_class), 4
..DollarNames.struct_class, 4
..DollarNames.DatasetExperiment (.DollarNames.struct_class), 4
..DollarNames.chart (.DollarNames.struct_class), 4
..DollarNames.iterator (.DollarNames.struct_class), 4
..DollarNames.metric (.DollarNames.struct_class), 4
..DollarNames.model (.DollarNames.struct_class), 4
..DollarNames.optimiser (.DollarNames.struct_class), 4
..DollarNames.preprocess (.DollarNames.struct_class), 4
..DollarNames.resampler (.DollarNames.struct_class), 4
..DollarNames.struct_class (.DollarNames.struct_class), 4
..DollarNames.DatasetExperiment (.DollarNames.struct_class), 4
..DollarNames.chart (.DollarNames.struct_class), 4
..DollarNames.iterator (.DollarNames.struct_class), 4
..DollarNames.metric (.DollarNames.struct_class), 4
..DollarNames.model (.DollarNames.struct_class), 4
..DollarNames.optimiser (.DollarNames.struct_class), 4
..DollarNames.preprocess (.DollarNames.struct_class), 4
..DollarNames.resampler (.DollarNames.struct_class), 4
..DollarNames.struct_class (.DollarNames.struct_class), 4
..onto...
calculate,test_metric-method
(test_metric-class), 58
chart, 11
cart_names, 11
cart_names,struct_class-method
(cart_names), 11
cart_plot, 12
cart_plot,chart,ANY-method
(cart_plot), 12
cart_plot,example_chart,example_model-method
(example_chart), 18
citations, 13
citations,struct_class-method
(citations), 13
DatasetExperiment, 14
distance, 16
distance,(max_length), 24
distance_stato, 15
distance_stato,17
evaluate,(run), 49
evaluate_iterator,metric-method,(run),
evaluated, 49
evaluated_iterator, 18
evaluated_iterator,(example_iterator-class), 19
evaluated_iterator-class, 19
evaluated_iterator,(run), 49
export_xlsx, 21
export_xlsx,DatasetExperiment-method
(export_xlsx), 21
generate_description, 21
iris_DatasetExperiment, 22
iteration, 22
iteration,(max_length), 24
iteration,struct_class-method
(iteration), 22
iteration,param, 23
iteration,param,struct_class-method
(iteration), 23
iteration,(run), 49
length,model_seq-method(model_seq), 31
length,ontology_list-method(ontology),
length,ontology_list-method(ontology), 35
libraries, 24
libraries,struct_class-method
(libraries), 24
max_length, 24
max_length,(entity-method(max_length),
max_length<-(calculate), 9
max_length<-,entity-method
(max_length), 24
metric,(calculate), 9
model, 26
model_apply, 29
model_apply,model,DatasetExperiment-method
(model), 26
model_apply,model_seq,DatasetExperiment-method
(model_seq), 31
model_predict, 30
model_predict,example_model,DatasetExperiment-method
(example_model), 20
model_predict,model,DatasetExperiment-method
(model), 26
model_predict,model_seq,DatasetExperiment-method
(model_seq), 31
model_reverse, 30
model_reverse,model,DatasetExperiment-method
(model), 26
model_reverse,preprocess,DatasetExperiment-method
(preprocess), 46
model_seq, 31
model_train, 33
model_train,example_model,DatasetExperiment-method
(example_model), 20
model_train,model,DatasetExperiment-method
(model), 26
model_train,model_seq,DatasetExperiment-method
(model_seq), 31
models, 28
models,iterator-method,(run), 49
models,model_seq-method(model_seq), 31
models<-(models), 28
models<-,iterator,model_OR_iterator-method
(run), 49
models<-,model_seq,member-method
(model_seq), 31
new_struct, 34
ontology, 35
stato_summary, stato-method (stato_id), 54
struct (struct-package), 3
struct-package, 3
struct_class, 56, 56
struct_class-class, 57
struct_template, 58

test_metric (test_metric-class), 58
test_metric-class, 58

value (calculate), 9
value, entity-method (max_length), 24
value, metric-method (calculate), 9
value<- (calculate), 9
value<-, entity-method (max_length), 24
value<-, enum-method (enum), 16
value<-, metric-method (calculate), 9