

Example learning

Generated by Doxygen 1.8.11

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	3
2.1	File List	3
3	Data Structure Documentation	5
3.1	Attribute Struct Reference	5
3.1.1	Detailed Description	5
3.2	Enum Struct Reference	5
3.2.1	Detailed Description	6
3.3	EnumType Struct Reference	6
3.3.1	Detailed Description	6
3.4	Example Struct Reference	6
3.4.1	Detailed Description	7
3.5	Examples Struct Reference	7
3.5.1	Detailed Description	7
3.6	Interval Struct Reference	7
3.6.1	Detailed Description	8
3.7	Model Struct Reference	8
3.7.1	Detailed Description	8
3.8	ModelAttribute Struct Reference	8
3.8.1	Detailed Description	9
3.9	ModelType Struct Reference	9

3.9.1	Detailed Description	9
3.10	Object Struct Reference	10
3.10.1	Detailed Description	10
3.11	ObjectIndice Struct Reference	10
3.11.1	Detailed Description	10
3.12	OutAttribute Struct Reference	11
3.12.1	Detailed Description	11
3.13	OutEnum Struct Reference	11
3.13.1	Detailed Description	12
3.14	OutObject Struct Reference	12
3.14.1	Detailed Description	12
3.15	Solution Struct Reference	12
3.15.1	Detailed Description	13
3.16	String Struct Reference	13
3.16.1	Detailed Description	13
3.17	StringVector Struct Reference	13
3.17.1	Detailed Description	14
3.18	Tree Struct Reference	14
3.18.1	Detailed Description	14
4	File Documentation	15
4.1	app/core.h File Reference	15
4.1.1	Detailed Description	16
4.1.2	Function Documentation	16
4.1.2.1	combiOutObjectObject(Model *mdl, OutObject *oo, Object *o)	16
4.1.2.2	compareOutObjects(OutObject *oo1, OutObject *oo2)	16
4.1.2.3	genAllCombi(Model *mdl, Examples *exp)	17
4.1.2.4	genAllRelations(Solution *s, Examples *e, Model *m)	18
4.1.2.5	genGeneralisation(Solution *s)	18
4.1.2.6	genSpecificity(Model *mdl, OutObject *oo)	18
4.1.2.7	getIndex(Examples *exp, ObjectIndice *oi)	18

4.1.2.8	<code>initAllCombi(Model *mdl, Examples *exp)</code>	19
4.1.2.9	<code>initOutObjectWithObject(Model *mdl, Object *o)</code>	19
4.1.2.10	<code>nbCombi(Examples *exp, int explndice)</code>	19
4.2	<code>app/output.h</code> File Reference	20
4.2.1	Detailed Description	21
4.2.2	Function Documentation	21
4.2.2.1	<code>cPrint(const char *fmt,...)</code>	21
4.2.2.2	<code>extractVerbosityFromArg(const char *verbosity)</code>	21
4.2.2.3	<code>genOutput(Solution *sol, Model *mdl)</code>	22
4.2.2.4	<code>output(unsigned int level, const char *fmt,...)</code>	22
4.2.2.5	<code>setOutputImportance(unsigned int level)</code>	22
4.3	<code>parser/parsers.h</code> File Reference	22
4.3.1	Detailed Description	24
4.3.2	Function Documentation	24
4.3.2.1	<code>getAttributePosition(const char *attr, Model *m)</code>	24
4.3.2.2	<code>getIncludeFile(char const *pathname, size_t *pos)</code>	24
4.3.2.3	<code>getNextExample(FILE *f)</code>	25
4.3.2.4	<code>getRelationPosition(const char *rel, Model *m)</code>	25
4.3.2.5	<code>isValidAttrChar(char c, unsigned int first)</code>	25
4.3.2.6	<code>loadConfigFile(char const *pathname)</code>	26
4.3.2.7	<code>loadExampleFile(char const *pathname, Model *model, size_t startPos)</code>	26
4.3.2.8	<code>parseAttrName(FILE *fp, char **error)</code>	26
4.3.2.9	<code>parseAttrType(FILE *fp, char **error)</code>	27
4.3.2.10	<code>parseAttrTypeEnum(FILE *fp, char **error)</code>	27
4.3.2.11	<code>parseAttrTypeInterval(FILE *fp, char **error)</code>	27
4.3.2.12	<code>parseAttrTypeTree(FILE *fp, char **error, int *index, int indent)</code>	27
4.3.2.13	<code>parseAttrValue(FILE *fp, char **error, Model *m, attrType type, Attribute *attr, unsigned int position, struct StringVector *seenObjects)</code>	28
4.3.2.14	<code>parseConfigLine(FILE *fp, char **error, Model *out)</code>	28
4.3.2.15	<code>parseExample(FILE *fp, char **error, Example *ex, Model *m)</code>	28

4.3.2.16	<code>parseExampleObject(FILE *fp, char **error, Object *o, Model *m, struct String↵ Vector *seenObjects)</code>	29
4.3.2.17	<code>printIndent(unsigned int flag, int indent)</code>	29
4.3.2.18	<code>readFileSpaces(FILE *fp, char const *set)</code>	29
4.3.2.19	<code>readTil(FILE *fp, char const *set)</code>	30
4.4	<code>types/attribute-types.h</code> File Reference	30
4.4.1	Detailed Description	30
4.5	<code>types/attribute.h</code> File Reference	30
4.5.1	Detailed Description	31
4.6	<code>types/enum.h</code> File Reference	31
4.6.1	Detailed Description	32
4.6.2	Function Documentation	32
4.6.2.1	<code>freeEnum(Enum *enu, int freeltself)</code>	32
4.6.2.2	<code>freeEnumType(EnumType *enuty, int freeltself)</code>	32
4.7	<code>types/example.h</code> File Reference	32
4.7.1	Detailed Description	33
4.7.2	Function Documentation	33
4.7.2.1	<code>freeExample(Example *exp, int freeltself)</code>	33
4.7.2.2	<code>initExample(Example *exp)</code>	33
4.8	<code>types/examples.h</code> File Reference	33
4.8.1	Detailed Description	34
4.8.2	Function Documentation	34
4.8.2.1	<code>freeExamples(Examples *exps)</code>	34
4.8.2.2	<code>initExamples(Examples *exps)</code>	34
4.9	<code>types/interval.h</code> File Reference	35
4.9.1	Detailed Description	35
4.9.2	Function Documentation	35
4.9.2.1	<code>addToInterval(Interval *inter, int x)</code>	35
4.10	<code>types/model-attribute.h</code> File Reference	36
4.10.1	Detailed Description	36
4.10.2	Function Documentation	36

4.10.2.1	<code>freeModelAttribute(ModelAttribute *ma, int freeltself)</code>	36
4.11	<code>types/model-type.h</code> File Reference	37
4.11.1	Detailed Description	37
4.11.2	Function Documentation	37
4.11.2.1	<code>freeModelType(ModelType *mt, int freeltself)</code>	37
4.12	<code>types/model.h</code> File Reference	38
4.12.1	Detailed Description	38
4.12.2	Function Documentation	38
4.12.2.1	<code>freeModel(Model *mdl)</code>	38
4.12.2.2	<code>getEnumId(const char *str, Model *mdl, unsigned int index)</code>	39
4.12.2.3	<code>getEnumStr(int id, Model *mdl, unsigned int index)</code>	39
4.12.2.4	<code>getTreeId(const char *str, Model *mdl, unsigned int index)</code>	39
4.12.2.5	<code>getTreeStr(int id, Model *mdl, unsigned int index)</code>	40
4.12.2.6	<code>initModel(Model *mdl)</code>	40
4.13	<code>types/object.h</code> File Reference	40
4.13.1	Detailed Description	41
4.13.2	Function Documentation	41
4.13.2.1	<code>freeObject(Object *obj, int freeltself)</code>	41
4.13.2.2	<code>initObject(Object *obj)</code>	41
4.14	<code>types/out-attribute.h</code> File Reference	41
4.14.1	Detailed Description	42
4.14.2	Function Documentation	42
4.14.2.1	<code>freeOutAttribute(OutAttribute *oa, int freeltself)</code>	42
4.15	<code>types/out-enum.h</code> File Reference	42
4.15.1	Detailed Description	43
4.15.2	Function Documentation	43
4.15.2.1	<code>freeOutEnum(OutEnum *oenu, int freeltself)</code>	43
4.15.2.2	<code>initOutEnum(OutEnum *oenu)</code>	43
4.16	<code>types/out-object.h</code> File Reference	44
4.16.1	Detailed Description	44

4.16.2	Function Documentation	44
4.16.2.1	freeOutObject(OutObject *oo, int freeltself)	44
4.16.2.2	initOutObject(OutObject *oo)	45
4.17	types/solution.h File Reference	45
4.17.1	Detailed Description	45
4.17.2	Function Documentation	45
4.17.2.1	freeSolution(Solution *sol)	45
4.17.2.2	initSolution(Solution *sol)	46
4.18	types/string-type.h File Reference	46
4.18.1	Detailed Description	47
4.18.2	Function Documentation	47
4.18.2.1	strDuplicate(char *str)	47
4.18.2.2	strInit(char *str)	47
4.18.2.3	strLength(String *str)	47
4.18.2.4	strPush(String *str, char c)	48
4.18.2.5	strPushStr(String *str, char *str2)	48
4.19	types/tree.h File Reference	48
4.19.1	Detailed Description	49
4.19.2	Function Documentation	49
4.19.2.1	addChild(Tree *node, Tree *child)	49
4.19.2.2	createLeaf(int id, char *str)	49
4.19.2.3	createNode(int id, char *str, Tree *child)	50
4.19.2.4	depth(Tree *root, int id)	50
4.19.2.5	freeTree(Tree *t)	50
4.19.2.6	height(Tree *t)	51
4.19.2.7	isLeaf(Tree *t)	51
4.19.2.8	LCA(Tree *root, int id1, int id2)	51
4.20	types/vector.h File Reference	52
4.20.1	Detailed Description	52
4.20.2	Macro Definition Documentation	52
4.20.2.1	vectAt	52
4.20.2.2	vectFree	53
4.20.2.3	vectIndexOf	53
4.20.2.4	vectInit	53
4.20.2.5	Vector	53
4.20.2.6	vectPush	54
4.20.2.7	vectRemoveLast	54
4.20.2.8	vectSize	54

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Attribute	Represents an attribute and the value it holds	5
Enum	Structure that defines the enumeration type	5
EnumType	Structure that contains an item of the enumeration	6
Example	All the objects composing an example (or a counter-example)	6
Examples	Structure that contains the examples and counter-examples of the parsed example file	7
Interval	Structure that contains a signed integer interval	7
Model	Contains the attributes and relations definitions found in the model file	8
ModelAttribute	Contains the definition of an attribute and its type	8
ModelType	Structure that contains the definition of the type	9
Object	Contains all the attributes and relations that compose an object	10
ObjectIndex	Stores a list of indexes When combining multiple objects, the identifier of each of them can be stored in this structure	10
OutAttribute	Represents an attribute used by the solution and the value it holds	11
OutEnum	Contains multiple enumeration items When combining multiple Object , each enumeration item is to be stored, this structure does that	11
OutObject	Contains all the attributes and relations that compose an outObject	12
Solution	Contains all the possible solutions	12
String	Dynamic string handler	13
StringVector	Stores an array or C string	13
Tree	Defines the trees	14

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

app/core.h	File containing the core that generates the solutions	15
app/output.h	File containing the output functions and helpers	20
parser/parsers.h	File containing the example and model file parser	22
types/attribute-types.h	File containing the attribute's types definition	30
types/attribute.h	File containing the definition of the attributes	30
types/enum.h	File containing the definition of the enumerations that can be used as types for the attributes	31
types/example.h	File containing the definition of the examples	32
types/examples.h	File containing the definition of the examples (the all example file content)	33
types/interval.h	File containing the definition of the interval type	35
types/model-attribute.h	File containing the definition of the attributes (as defined by the model file)	36
types/model-type.h	File containing the definition of an attribute's type	37
types/model.h	File containing the definition of the model parsed in the model file	38
types/object.h	File containing the definition of the objects	40
types/out-attribute.h	File containing the definition of the out-attributes	41
types/out-enum.h	File containing the definition of the enumeration extracts	42
types/out-object.h	File containing the definition of the out-objects. Generated when combining multiple objects	44
types/solution.h	File containing the definition of the solution	45
types/string-type.h	File containing the definition of the String type and some tools to use it	46

types/ tree.h	
File containing the definition of the trees	48
types/ vector.h	
File containing the definition of the vectors (dynamic & generic arrays)	52

Chapter 3

Data Structure Documentation

3.1 Attribute Struct Reference

Represents an attribute and the value it holds.

```
#include <attribute.h>
```

Data Fields

- [attrType type](#)
The type of the attribute.
- `int` [value](#)
The value of the attribute.

3.1.1 Detailed Description

Represents an attribute and the value it holds.

This structure holds an attribute of an object. Attributes can hold values of different types (signed integer, item of an enumeration, node or leaf of a tree), each of these types can be represented by an integer (either the true value, or the ID of the real value).

The documentation for this struct was generated from the following file:

- `types/attribute.h`

3.2 Enum Struct Reference

Structure that defines the enumeration type.

```
#include <enum.h>
```

Public Member Functions

- [Vector \(EnumType\)](#) `enu`
Array of each items composing the enumeration.

3.2.1 Detailed Description

Structure that defines the enumeration type.

The documentation for this struct was generated from the following file:

- `types/enum.h`

3.3 EnumType Struct Reference

Structure that contains an item of the enumeration.

```
#include <enum.h>
```

Data Fields

- `int id`
The unique identifier of the value.
- `char * str`
The name of this item of the enumeration.

3.3.1 Detailed Description

Structure that contains an item of the enumeration.

Enumerations are arrays of [EnumType](#), each of which contain an item of the enumeration, characterized by a unique identifier and his name as a string

The documentation for this struct was generated from the following file:

- `types/enum.h`

3.4 Example Struct Reference

All the objects composing an example (or a counter-example)

```
#include <example.h>
```

Public Member Functions

- [Vector \(Object\)](#) objects

The array that contains the objects of which is composed the example.

3.4.1 Detailed Description

All the objects composing an example (or a counter-example)

An example contains all the objects linked as part of this example. It can store either examples or counter-examples (only the use we make of it differs between counter-examples and examples)

The documentation for this struct was generated from the following file:

- [types/example.h](#)

3.5 Examples Struct Reference

Structure that contains the examples and counter-examples of the parsed example file.

```
#include <examples.h>
```

Public Member Functions

- [Vector \(Example\)](#) examples
Contains all the examples of the file.
- [Vector \(Example\)](#) counterExamples
Contains all the counter-examples of the file.

3.5.1 Detailed Description

Structure that contains the examples and counter-examples of the parsed example file.

Each example and counter-example found in the example file is stored in this structure. Counter-examples are [Example](#) too, only the fields allow to differentiate them

The documentation for this struct was generated from the following file:

- [types/examples.h](#)

3.6 Interval Struct Reference

Structure that contains a signed integer interval.

```
#include <interval.h>
```

Data Fields

- int [min](#)
The lower bound.
- int [max](#)
The upper bound.

3.6.1 Detailed Description

Structure that contains a signed integer interval.

An [Interval](#) represents the interval between a lower and an upper bound

The documentation for this struct was generated from the following file:

- [types/interval.h](#)

3.7 Model Struct Reference

Contains the attributes and relations definitions found in the model file.

```
#include <model.h>
```

Public Member Functions

- [Vector](#) ([ModelAttribute](#)) ma
An array of attributes definition. Allow to know which attribute is ruled by which rules. The order in which they are stored matters.
- [Vector](#) (char *) rel
An array of the relations definitions. The order in which they are stored matters Only a string is stored (their name), the object they link is defined in the [Examples](#).

3.7.1 Detailed Description

Contains the attributes and relations definitions found in the model file.

The documentation for this struct was generated from the following file:

- [types/model.h](#)

3.8 ModelAttribute Struct Reference

Contains the definition of an attribute and its type.

```
#include <model-attribute.h>
```


Data Fields

- [ModelType mt](#)
The definition of the value this attribute can hold.
- char * [name](#)
The name of this attribute, as found in the model file.

3.8.1 Detailed Description

Contains the definition of an attribute and its type.

Each attribute can be of 4 types (signed integer, enumeration item, tree node or leaf, relation) This structure contains the link between an attribute, its name and the type it holds (and the boundaries of this type (bounds of the interval, possible values of the enumerations, etc...))

The documentation for this struct was generated from the following file:

- types/[model-attribute.h](#)

3.9 ModelType Struct Reference

Structure that contains the definition of the type.

```
#include <model-type.h>
```

Data Fields

- [attrType type](#)
The type contained.
 - union {
 - [Interval inter](#)
Contains the definition of the type if it is an interval.
 - [Enum enu](#)
Contains the definition of the type if it is an enumeration.
 - [Tree tree](#)
Contains the definition of the type if it is a tree.
- ```
};
```

### 3.9.1 Detailed Description

Structure that contains the definition of the type.

The documentation for this struct was generated from the following file:

- types/[model-type.h](#)

## 3.10 Object Struct Reference

Contains all the attributes and relations that compose an object.

```
#include <object.h>
```

### Public Member Functions

- [Vector \(Attribute\)](#) attributes  
*Array of the attributes of the object. Each attribute must be at the same index as its definition in the [Model](#).*
- [Vector \(Attribute\)](#) relations  
*Array of the relations of the object. Each relation must be at the same index as its definition in the [Model](#).*

### Data Fields

- char \* [name](#)  
*The name of the object.*
- unsigned int [id](#)  
*the unique identifier of the object*

#### 3.10.1 Detailed Description

Contains all the attributes and relations that compose an object.

The documentation for this struct was generated from the following file:

- [types/object.h](#)

## 3.11 ObjectIndice Struct Reference

Stores a list of indexes When combining multiple objects, the identifier of each of them can be stored in this structure.

```
#include <core.h>
```

### Public Member Functions

- [Vector](#) (int) indices  
*The indexes of the combined objects.*

#### 3.11.1 Detailed Description

Stores a list of indexes When combining multiple objects, the identifier of each of them can be stored in this structure.

The documentation for this struct was generated from the following file:

- [app/core.h](#)

## 3.12 OutAttribute Struct Reference

Represents an attribute used by the solution and the value it holds.

```
#include <out-attribute.h>
```

### Data Fields

- [attrType type](#)  
*The type contained by this attribute.*
- union {
  - [Interval inter](#)  
*Contains the definition of the type if it is an interval.*
  - [OutEnum oenu](#)  
*Contains the definition of the type if it is an enumeration extract.*
  - int [tree](#)  
*Contains the definition of the type if it is the node or the leaf of a tree.*

### 3.12.1 Detailed Description

Represents an attribute used by the solution and the value it holds.

This structure holds the attribute of the object generated by the solution. They contains the generalisation of the objects of which they are the composition OutAttributes can hold values of different types (signed integer interval, items of an enumeration, node or leaf of a tree).

The documentation for this struct was generated from the following file:

- types/[out-attribute.h](#)

## 3.13 OutEnum Struct Reference

Contains multiple enumeration items When combining multiple [Object](#), each enumeration item is to be stored, this structure does that.

```
#include <out-enum.h>
```

### Public Member Functions

- [Vector](#) (int) oenu  
*An array that contains the identifier of each enumeration item contained.*

### 3.13.1 Detailed Description

Contains multiple enumeration items When combining multiple [Object](#), each enumeration item is to be stored, this structure does that.

The documentation for this struct was generated from the following file:

- [types/out-enum.h](#)

## 3.14 OutObject Struct Reference

Contains all the attributes and relations that compose an outObject.

```
#include <out-object.h>
```

### Public Member Functions

- [Vector](#) ([OutAttribute](#)) attributes  
*Array of the attributes of the out object. Each attribute must be at the same index as its definition in the [Model](#).*
- [Vector](#) (struct [OutObject](#) \*) relations  
*Array of the relations of the out object. Each relation must be at the same index as its definition in the [Model](#).*

### Data Fields

- char \* [name](#)  
*Name of the out object.*
- unsigned char [specificity](#)  
*Level of specificity of the out object. Between 1 (none) and 100 (very), 0 if a duplicate.*

### 3.14.1 Detailed Description

Contains all the attributes and relations that compose an outObject.

The documentation for this struct was generated from the following file:

- [types/out-object.h](#)

## 3.15 Solution Struct Reference

Contains all the possible solutions.

```
#include <solution.h>
```

## Public Member Functions

- [Vector \(OutObject\)](#) outobjects  
*An array of out objects, each one representing a solution.*

### 3.15.1 Detailed Description

Contains all the possible solutions.

The documentation for this struct was generated from the following file:

- [types/solution.h](#)

## 3.16 String Struct Reference

Dynamic string handler.

```
#include <string-type.h>
```

## Data Fields

- `char * str`  
*The normal, nul terminated char array that represents the string.*
- `unsigned int length`  
*The current string length (number of characters in the string)*
- `unsigned int availableLength`  
*Size of the bloc allocated.*

### 3.16.1 Detailed Description

Dynamic string handler.

Allows to work on string and perform additions to the string without having to care about memory management and reallocations

The documentation for this struct was generated from the following file:

- [types/string-type.h](#)

## 3.17 StringVector Struct Reference

Stores an array or C string.

```
#include <parsers.h>
```

## Public Member Functions

- [Vector](#) (char \*) seen  
*An array of strings.*

### 3.17.1 Detailed Description

Stores an array or C string.

The documentation for this struct was generated from the following file:

- parser/[parsers.h](#)

## 3.18 Tree Struct Reference

Defines the trees.

```
#include <tree.h>
```

## Public Member Functions

- [Vector](#) (struct [Tree](#)) children  
*all the children of this node (or nothing if a leaf)*

## Data Fields

- int [id](#)  
*The unique identifier of this node or leaf.*
- char \* [str](#)  
*The name of this node or leaf.*

### 3.18.1 Detailed Description

Defines the trees.

The documentation for this struct was generated from the following file:

- types/[tree.h](#)

## Chapter 4

# File Documentation

### 4.1 app/core.h File Reference

File containing the core that generates the solutions.

```
#include <stdio.h>
#include <math.h>
#include "../types/model.h"
#include "../types/examples.h"
#include "../types/solution.h"
#include "output.h"
```

#### Data Structures

- struct [ObjectIndex](#)  
*Stores a list of indexes When combining multiple objects, the identifier of each of them can be stored in this structure.*

#### Typedefs

- typedef struct [ObjectIndex](#) **ObjectIndex**

#### Functions

- int [nbCombi](#) ([Examples](#) \*exp, int explndice)  
*Computes the number of combinations possible for our examples from an example.*
- [OutObject](#) \* [initOutObjectWithObject](#) ([Model](#) \*mdl, [Object](#) \*o)  
*Generate a filled [OutObject](#) based on an object values.*
- [Solution](#) \* [initAllCombi](#) ([Model](#) \*mdl, [Examples](#) \*exp)  
*Init all combinaisons with last example objects.*
- void [combiOutObjectObject](#) ([Model](#) \*mdl, [OutObject](#) \*oo, [Object](#) \*o)  
*Combine an [OutObject](#) and an [Object](#) into an [OutObject](#).*
- [Solution](#) \* [genAllCombi](#) ([Model](#) \*mdl, [Examples](#) \*exp)  
*Generate all the combinations for our examples.*
- void [genAllRelations](#) ([Solution](#) \*s, [Examples](#) \*e, [Model](#) \*m)

- Find all the common relation between the objects.*
- int `getIndex` (`Examples *exp`, `ObjectIndice *oi`)  
*Get the index of the combinaisons of object in the array.*
- void `genSpecificity` (`Model *mdl`, `OutObject *oo`)  
*Calculate the level of specifity of an `OutObject` based on the model.*
- int `compareOutObjects` (`OutObject *oo1`, `OutObject *oo2`)  
*Compare two `OutObjects`.*
- void `genGeneralisation` (`Solution *s`)  
*Generalisation of our solution(s)*

#### 4.1.1 Detailed Description

File containing the core that generates the solutions.

##### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

#### 4.1.2 Function Documentation

##### 4.1.2.1 void `combiOutObjectObject` ( `Model * mdl`, `OutObject * oo`, `Object * o` )

Combine an `OutObject` and an `Object` into an `OutObject`.

##### Parameters

|                  |                                                                           |
|------------------|---------------------------------------------------------------------------|
| <code>mdl</code> | Pointer to the <code>Model</code>                                         |
| <code>oo</code>  | Pointer to the <code>OutObject</code> to combine with <code>Object</code> |
| <code>o</code>   | Pointer to the <code>Object</code> to combine with <code>OutObject</code> |

##### 4.1.2.2 int `compareOutObjects` ( `OutObject * oo1`, `OutObject * oo2` )

Compare two `OutObjects`.

##### Parameters

|                  |                                                         |
|------------------|---------------------------------------------------------|
| <code>oo1</code> | Pointer to the first <code>OutObject</code> (reference) |
| <code>oo2</code> | Pointer to the second <code>OutObject</code>            |

##### Returns

An integer: -1 = `oo2` less specific than `oo1`, 0 = same, 1 = `oo2` more specific than `oo1` or different values



#### 4.1.2.3 Solution\* genAllCombi ( Model \* *mdl*, Examples \* *exp* )

Generate all the combinations for our examples.

## Parameters

|            |                                                    |
|------------|----------------------------------------------------|
| <i>mdl</i> | Pointer to the <a href="#">Model</a>               |
| <i>exp</i> | Pointer to our <a href="#">Examples</a> to combine |

## Returns

Pointer to [Solution](#) containing all our combinaisons

#### 4.1.2.4 void genAllRelations ( [Solution](#) \* *s*, [Examples](#) \* *e*, [Model](#) \* *m* )

Find all the common relation between the objects.

## Parameters

|          |                                                    |
|----------|----------------------------------------------------|
| <i>s</i> | The solution generated by the genAllCombi function |
| <i>e</i> | The examples to search the relations into          |
| <i>m</i> | The model to use the relations                     |

#### 4.1.2.5 void genGeneralisation ( [Solution](#) \* *s* )

Generalisation of our solution(s)

## Parameters

|          |                                         |
|----------|-----------------------------------------|
| <i>s</i> | Pointer to the <a href="#">Solution</a> |
|----------|-----------------------------------------|

#### 4.1.2.6 void genSpecificity ( [Model](#) \* *mdl*, [OutObject](#) \* *oo* )

Calculate the level of specificity of an [OutObject](#) based on the model.

## Parameters

|            |                                          |
|------------|------------------------------------------|
| <i>mdl</i> | Pointer to the model                     |
| <i>oo</i>  | Pointer to the <a href="#">OutObject</a> |

#### 4.1.2.7 int getIndex ( [Examples](#) \* *exp*, [ObjectIndice](#) \* *oi* )

Get the index of the combinaisons of object in the array.

## Parameters

|            |                                  |
|------------|----------------------------------|
| <i>exp</i> | Pointer to the examples          |
| <i>oi</i>  | Pointer to the objects's indices |

**Returns**

An integer

**4.1.2.8 Solution\* initAllCombi ( Model \* mdl, Examples \* exp )**

Init all combinaisons with last example objects.

**Parameters**

|            |                                         |
|------------|-----------------------------------------|
| <i>mdl</i> | Pointer to the <a href="#">Model</a>    |
| <i>exp</i> | Pointer to the <a href="#">Examples</a> |

**Returns**

Pointer to [Solution](#) initialized

**4.1.2.9 OutObject\* initOutObjectWithObject ( Model \* mdl, Object \* o )**

Generate a filled [OutObject](#) based on an object values.

**Parameters**

|            |                                       |
|------------|---------------------------------------|
| <i>mdl</i> | Pointer to the <a href="#">Model</a>  |
| <i>o</i>   | Pointer to the <a href="#">Object</a> |

**Returns**

Pointer to [OutObject](#)

**4.1.2.10 int nbCombi ( Examples \* exp, int step )**

Computes the number of combinations possible for our examples from an example.

**Parameters**

|                  |                                 |
|------------------|---------------------------------|
| <i>exp</i>       | Pointer to our array of exemple |
| <i>expIndice</i> | The example number (= indice)   |

**Returns**

The number of combinations starting at expIndice example

Computes the number of combinations possible for our examples from an example.

FOPPOLO Gaël PHILIP Bastien

## 4.2 app/output.h File Reference

File containing the output functions and helpers.

```
#include <stdio.h>
#include <stdarg.h>
#include "../types/model.h"
#include "../types/solution.h"
#include "../types/string-type.h"
```

### Macros

- `#define LERROR 8`  
*Flag for the output function. Represents an error.*

### TTY colors

*Color values in Unix and MacOS terminals*

- `#define SDEFAULT "\e[0m"`
- `#define SBDEFAULT "\e[1m"`
- `#define SBLACK "\e[0;30m"`
- `#define SRED "\e[0;31m"`
- `#define SGREEN "\e[0;32m"`
- `#define SYELLOW "\e[0;33m"`
- `#define SBLUE "\e[0;34m"`
- `#define SPURPLE "\e[0;35m"`
- `#define SCYAN "\e[0;36m"`
- `#define SWHITE "\e[0;37m"`
- `#define SBBLACK "\e[1;30m"`
- `#define SBRED "\e[1;31m"`
- `#define SBGREEN "\e[1;32m"`
- `#define SBYELLOW "\e[1;33m"`
- `#define SBBLUE "\e[1;34m"`
- `#define SBPURPLE "\e[1;35m"`
- `#define SBCYAN "\e[1;36m"`
- `#define SBWHITE "\e[1;37m"`
- `#define SUBLACK "\e[4;30m"`
- `#define SURED "\e[4;31m"`
- `#define SUGREEN "\e[4;32m"`
- `#define SUYELLOW "\e[4;33m"`
- `#define SUBLUE "\e[4;34m"`
- `#define SUPURPLE "\e[4;35m"`
- `#define SUCYAN "\e[4;36m"`
- `#define SUWHITE "\e[4;37m"`
- `#define SBUBLACK "\e[1;4;30m"`
- `#define SBURED "\e[1;4;31m"`
- `#define SBUGREEN "\e[1;4;32m"`
- `#define SBUYELLOW "\e[1;4;33m"`
- `#define SBUBLUE "\e[1;4;34m"`
- `#define SBUPURPLE "\e[1;4;35m"`
- `#define SBUCYAN "\e[1;4;36m"`
- `#define SBUWHITE "\e[1;4;37m"`

### output importance

*Flags for the output function. Represents levels of importance*

- `#define L0 0`
- `#define L1 1`
- `#define L2 2`
- `#define L3 3`
- `#define L4 4`
- `#define L5 5`
- `#define L6 6`
- `#define L7 7`

## Functions

- void [genOutput](#) ([Solution](#) \*sol, [Model](#) \*mdl)  
*Returns a string that contains the representation of the object in a readable presentation.*
- char \* [cPrint](#) (const char \*fmt,...)  
*Returns a string that contains the formatted output by concatenating all the arguments.*
- void [setOutputImportance](#) (unsigned int level)  
*Sets the max level of the messages to output (0 : only critical messages, the higher the value, the less importance the messages)*
- unsigned int [getOutputImportance](#) ()  
*Gets the max level of the messages to output (0 : only critical messages, the higher the value, the less importance the messages)*
- void [output](#) (unsigned int level, const char \*fmt,...)  
*Print the message in the standard output only if its importance is high enough to be printed.*
- unsigned int [extractVerbosityFromArg](#) (const char \*verbosity)  
*Extract the verbosity value from the "-v[v..]" formatted string.*

### 4.2.1 Detailed Description

File containing the output functions and helpers.

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

### 4.2.2 Function Documentation

#### 4.2.2.1 char\* cPrint ( const char \* fmt, ... )

Returns a string that contains the formatted output by concatenating all the arguments.

##### Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>fmt</i>  | The format string (same used by the printf family)    |
| <i>args</i> | The list of arguments to include in the output string |

##### Returns

A newly allocated string that contains the arguments given to the function formatted

#### 4.2.2.2 unsigned int extractVerbosityFromArg ( const char \* verbosity )

Extract the verbosity value from the "-v[v..]" formatted string.

##### Parameters

|                  |                                          |
|------------------|------------------------------------------|
| <i>verbosity</i> | The string to extract the verbosity from |
|------------------|------------------------------------------|

**Returns**

The level extracted

**4.2.2.3 void genOutput ( Solution \* *sol*, Model \* *mdl* )**

Returns a string that contains the representation of the object in a readable presentation.

**Parameters**

|            |                                                                                        |
|------------|----------------------------------------------------------------------------------------|
| <i>sol</i> | Pointer to the object grouping the common traits of the other objects int the examples |
| <i>mdl</i> | Pointer to the model object containing structure of the model                          |

**Returns**

A string representing the object in a readable way. Need to be freed by the user

**4.2.2.4 void output ( unsigned int *level*, const char \* *fmt*, ... )**

Print the message in the standard output only if its importance is high enough to be printed.

**Parameters**

|              |                                                                                                                                                                       |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>level</i> | The importance level of the message (flags, can use L[0-7] and add the flag LERROR if you want to write in the error stream LERROR alone is aquivalent to L0   LERROR |
| <i>fmt</i>   | The message to be printed                                                                                                                                             |
| <i>args</i>  | The arguments needed by the fmt argument                                                                                                                              |

**4.2.2.5 void setOutputImportance ( unsigned int *level* )**

Sets the max level of the messages to output (0 : only critical messages, the higher the value, the less importance the messages)

**Parameters**

|              |                  |
|--------------|------------------|
| <i>level</i> | The level to set |
|--------------|------------------|

**4.3 parser/parsers.h File Reference**

File containing the example and model file parser.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include "../types/examples.h"
#include "../types/model.h"
#include "../types/string-type.h"
#include "../app/output.h"
```

## Data Structures

- struct [StringVector](#)  
Stores an array or C string.

## Macros

- #define [PARSED\\_EXAMPLE](#) 1  
Value representing an example in the example file.
- #define [PARSED\\_COUNTEREXAMPLE](#) 2  
Value representing a counter-example in the example file.

## Functions

- char \* [getIncludeFile](#) (char const \*pathname, size\_t \*pos)  
Get the pathname to the config file included at the beginning of an example file.
- [Examples](#) \* [loadExampleFile](#) (char const \*pathname, [Model](#) \*model, size\_t startPos)  
Loads the example file given and generate the [Example](#) object that represents its content.
- unsigned int [getNextExample](#) (FILE \*f)  
Get the type of the next example (example or counter-example). Stop reading at the end of the example name, on the last character.
- int [parseExample](#) (FILE \*fp, char \*\*error, [Example](#) \*ex, [Model](#) \*m)  
Parse an example or a counterexample.
- int [parseExampleObject](#) (FILE \*fp, char \*\*error, [Object](#) \*o, [Model](#) \*m, struct [StringVector](#) \*seenObjects)  
Parse an object (only its properties. The name must already be known)
- int [getAttributePosition](#) (const char \*attr, [Model](#) \*m)  
Returns the position at which can be found an attribute (by name)
- int [getRelationPosition](#) (const char \*rel, [Model](#) \*m)  
Returns the position at which can be found a relation (by name)
- void [parseAttrValue](#) (FILE \*fp, char \*\*error, [Model](#) \*m, [attrType](#) type, [Attribute](#) \*attr, unsigned int position, struct [StringVector](#) \*seenObjects)  
Parse the attribute's value and populate the [Attribute](#) object accordingly.
- [Model](#) \* [loadConfigFile](#) (char const \*pathname)  
Loads the config file given and generate the [Model](#) object that represents its content.
- int [parseConfigLine](#) (FILE \*fp, char \*\*error, [Model](#) \*out)  
Tries to parse the line as a config file attribute definition. If the line is empty, continues to read until it finds a line.
- char \* [parseAttrName](#) (FILE \*fp, char \*\*error)  
Tries to parse the attribute name at the current position in the file (spaces & tabs are omitted)
- [ModelType](#) \* [parseAttrType](#) (FILE \*fp, char \*\*error)

*Tries to parse the attribute's value definition at the current position in the file (may read more than one line in case of trees)*

- **Interval** \* `parseAttrTypeInterval` (FILE \*fp, char \*\*error)

*Tries to parse an interval.*

- **Enum** \* `parseAttrTypeEnum` (FILE \*fp, char \*\*error)

*Tries to parse an enumeration.*

- **Tree** \* `parseAttrTypeTree` (FILE \*fp, char \*\*error, int \*index, int indent)

*Tries to parse a tree.*

- int `isValidAttrChar` (char c, unsigned int first)

*Check whether the character is allowed in an attribute name or not.*

- void `readFileSpaces` (FILE \*fp, char const \*set)

*Reads a file from the current position and reads while characters are in the set. Stops on the last one.*

- void `readTil` (FILE \*fp, char const \*set)

*Reads a file from the current position and reads until it finds a character in the set. Stops on the last character not in the set.*

- void `printIndent` (unsigned int flag, int indent)

*Display tabs(s)*

### 4.3.1 Detailed Description

File containing the example and model file parser.

#### Author

Bastien Philip (ebatsin)

Gaël Foppolo (gaelfoppolo)

### 4.3.2 Function Documentation

#### 4.3.2.1 int `getAttributePosition` ( const char \* *attr*, Model \* *m* )

Returns the position at which can be found an attribute (by name)

#### Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>attr</i> | the attribute to search for          |
| <i>m</i>    | The model in which to find the order |

#### Returns

the index of the attribute (or -1 in case the attributes is not in the model)

#### 4.3.2.2 char\* `getIncludeFile` ( char const \* *pathname*, size\_t \* *pos* )

Get the pathname to the config file included at the begening of an example file.



## Parameters

|                 |                                                                                                                     |
|-----------------|---------------------------------------------------------------------------------------------------------------------|
| <i>pathname</i> | The path to the example file                                                                                        |
| <i>pos</i>      | Will contain the position of the character after the last character of the include (basically, a " ", "\t" or "\n") |

## Returns

If the file to include is found, the file name. NULL otherwise

Get the pathname to the config file included at the begening of an example file.

FOPPOLO Gaël PHILIP Bastien

#### 4.3.2.3 unsigned int getNextExample ( FILE \* f )

Get the type of the next example (example or counter-example). Stop reading at the end of the example name, on the last character.

## Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>f</i>       | The file to be read                                                                                                  |
| <i>Returns</i> | 0 in case of error, PARSED_EXAMPLE if the line is an example, PARSED_COUNTEREXAMPLE if the line is a counter-example |

#### 4.3.2.4 int getRelationPosition ( const char \* rel, Model \* m )

Returns the position at which can be found a relation (by name)

## Parameters

|            |                                      |
|------------|--------------------------------------|
| <i>rel</i> | the relation to search for           |
| <i>m</i>   | The model in which to find the order |

## Returns

the index of the attribute (or -1 in case the attributes is not in the model)

#### 4.3.2.5 int isValidAttrChar ( char c, unsigned int first )

Check whether the character is allowed in an attribute name or not.

## Parameters

|              |                                                     |
|--------------|-----------------------------------------------------|
| <i>c</i>     | The character to Check                              |
| <i>first</i> | Wether the character is the first to be read or not |

**Returns**

Returns 1 if the character is valid. 0 otherwise

**4.3.2.6 Model\* loadConfigFile ( char const \* *pathname* )**

Loads the config file given and generate the [Model](#) object that represents its content.

**Parameters**

|                 |                                        |
|-----------------|----------------------------------------|
| <i>pathname</i> | The path to the config file to be read |
|-----------------|----------------------------------------|

**Returns**

A newly created [Model](#) object

**4.3.2.7 Examples\* loadExampleFile ( char const \* *pathname*, Model \* *model*, size\_t *startPos* )**

Loads the example file given and generate the [Example](#) object that represents its content.

**Parameters**

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| <i>pathname</i> | The path to the example file to be read                         |
| <i>model</i>    | The <a href="#">Model</a> object generated from the config file |
| <i>startPos</i> | The position at which to start parsing the file                 |

**Returns**

A newly created [Examples](#) object

**4.3.2.8 char\* parseAttrName ( FILE \* *fp*, char \*\* *error* )**

Tries to parse the attribute name at the current position in the file (spaces & tabs are omitted)

**Parameters**

|              |                                                                                                                                             |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <i>fp</i>    | The file in which to read                                                                                                                   |
| <i>error</i> | In case of error, contains a description of the error. NULL if no error occurred. Must be an uninitialized variable or data loss may happen |

**Returns**

Returns a new string that contains the attribute name or NULL in case of error

#### 4.3.2.9 `ModelType* parseAttrType ( FILE * fp, char ** error )`

Tries to parse the attribute's value definition at the current position in the file (may read more than one line in case of trees)

##### Parameters

|              |                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>fp</i>    | Ther file in which to read                                                                                                                |
| <i>error</i> | In case of error, contains a description of the error. NULL if no error append. Must be an uninitialized variable or data loss may happen |

##### Returns

Returns the [ModelType](#) built from the file definition or NULL in case of error

#### 4.3.2.10 `Enum* parseAttrTypeEnum ( FILE * fp, char ** error )`

Tries to parse an enumeration.

##### Parameters

|              |                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>fp</i>    | Ther file in which to read                                                                                                                |
| <i>error</i> | In case of error, contains a description of the error. NULL if no error append. Must be an uninitialized variable or data loss may happen |

##### Returns

The parsed value or NULL in case of error

#### 4.3.2.11 `Interval* parseAttrTypeInterval ( FILE * fp, char ** error )`

Tries to parse an interval.

##### Parameters

|              |                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>fp</i>    | Ther file in which to read                                                                                                                |
| <i>error</i> | In case of error, contains a description of the error. NULL if no error append. Must be an uninitialized variable or data loss may happen |

##### Returns

The parsed value or NULL in case of error

#### 4.3.2.12 `Tree* parseAttrTypeTree ( FILE * fp, char ** error, int * index, int indent )`

Tries to parse a tree.

## Parameters

|              |                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>fp</i>    | The file in which to read                                                                                                                 |
| <i>error</i> | In case of error, contains a description of the error. NULL if no error append. Must be an uninitialized variable or data loss may happen |

## Returns

The parsed value or NULL in case of error

**4.3.2.13** void parseAttrValue ( FILE \* *fp*, char \*\* *error*, Model \* *m*, attrType *type*, Attribute \* *attr*, unsigned int *position*, struct StringVector \* *seenObjects* )

Parse the attribute's value and populate the [Attribute](#) object accordingly.

## Parameters

|                    |                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <i>fp</i>          | The file in which to read                                                                                                                   |
| <i>error</i>       | In case of error, contains a description of the error. NULL if no error happened. Must be an uninitialized variable or data loss may occur. |
| <i>m</i>           | The model to use for the parsing                                                                                                            |
| <i>type</i>        | The expected type of the attribute                                                                                                          |
| <i>attr</i>        | A pointer to the attribute to populate                                                                                                      |
| <i>position</i>    | The position of the attribute in the model                                                                                                  |
| <i>seenObjects</i> | The names of the objects that have already been seen                                                                                        |

**4.3.2.14** int parseConfigLine ( FILE \* *fp*, char \*\* *error*, Model \* *out* )

Tries to parse the line as a config file attribute definition. If the line is empty, continues to read until it finds a line.

## Parameters

|              |                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>fp</i>    | The file in which to read                                                                                                                 |
| <i>error</i> | In case of error, contains a description of the error. NULL if no error append. Must be an uninitialized variable or data loss may happen |
| <i>out</i>   | A pointer to the <a href="#">Model</a> object to populate                                                                                 |

## Returns

A boolean. 1 for success. 0 for failure.

**4.3.2.15** int parseExample ( FILE \* *fp*, char \*\* *error*, Example \* *ex*, Model \* *m* )

Parse an example or a counterexample.

## Parameters

|              |                                                                                                                                             |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <i>fp</i>    | The file in which to read                                                                                                                   |
| <i>error</i> | In case of error, contains a description of the error. NULL if no error happened. Must be an uninitialized variable or data loss may occur. |
| <i>ex</i>    | A pointer to the example object to populate                                                                                                 |
| <i>m</i>     | The <a href="#">Model</a> object generated from the config file                                                                             |

## Returns

A boolean. 1 for success. 0 for failure.

**4.3.2.16** `int parseExampleObject ( FILE * fp, char ** error, Object * o, Model * m, struct StringVector * seenObjects )`

Parse an object (only its properties. The name must already be known)

## Parameters

|                    |                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <i>fp</i>          | The file in which to read                                                                                                                   |
| <i>error</i>       | In case of error, contains a description of the error. NULL if no error happened. Must be an uninitialized variable or data loss may occur. |
| <i>o</i>           | A pointer to the object <a href="#">Object</a> to populate                                                                                  |
| <i>m</i>           | The <a href="#">Model</a> object generated from the config file                                                                             |
| <i>seenObjects</i> | The names of the objects that have already been seen                                                                                        |

## Returns

A boolean. 1 for success. 0 for failure

**4.3.2.17** `void printIndent ( unsigned int flag, int indent )`

Display tabs(s)

## Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>flag</i>   | The output flag to use (L1 to L7 and/or LERROR) |
| <i>indent</i> | The number of tabs to display                   |

**4.3.2.18** `void readFileSpaces ( FILE * fp, char const * set )`

Reads a file from the current position and reads while characters are in the set. Stops on the last one.

## Parameters

|            |                                                                    |
|------------|--------------------------------------------------------------------|
| <i>f</i>   | The file to be read                                                |
| <i>set</i> | A nul terminated array of char that contains the set of characters |

#### 4.3.2.19 void readTil ( FILE \* *fp*, char const \* *set* )

Reads a file from the current position and reads until it finds a character in the set. Stops on the last character not in the set.

##### Parameters

|            |                                                                             |
|------------|-----------------------------------------------------------------------------|
| <i>f</i>   | The file to be read                                                         |
| <i>set</i> | A nul terminated array of char that contains the set of characters to reach |

## 4.4 types/attribute-types.h File Reference

File containing the attribute's types definition.

### Macros

- #define `TYPE_INT` 1  
*Indicate a type that stores signed integers.*
- #define `TYPE_ENUM` 2  
*Indicate a type that stores enumerations.*
- #define `TYPE_TREE` 3  
*Indicate a type that stores a tree.*
- #define `TYPE_RELATION` 4  
*Indicate a type that stores a relation.*
- #define `TYPE_NORELATION` 5  
*Denote the absence of a relation in this attribute.*

### Typedefs

- typedef unsigned char `attrType`  
*Stores the type of the attribute.*

#### 4.4.1 Detailed Description

File containing the attribute's types definition.

##### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

## 4.5 types/attribute.h File Reference

File containing the definition of the attributes.

```
#include "attribute-types.h"
```

## Data Structures

- struct [Attribute](#)  
*Represents an attribute and the value it holds.*

## Typedefs

- typedef struct [Attribute](#) **Attribute**

### 4.5.1 Detailed Description

File containing the definition of the attributes.

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

## 4.6 types/enum.h File Reference

File containing the definition of the enumerations that can be used as types for the attributes.

```
#include "vector.h"
```

## Data Structures

- struct [EnumType](#)  
*Structure that contains an item of the enumeration.*
- struct [Enum](#)  
*Structure that defines the enumeration type.*

## Typedefs

- typedef struct [EnumType](#) **EnumType**
- typedef struct [Enum](#) **Enum**

## Functions

- void [freeEnumType](#) ([EnumType](#) \*enuty, int freeltself)  
*Free the [EnumType](#).*
- void [freeEnum](#) ([Enum](#) \*enu, int freeltself)  
*Free the [Enum](#).*

### 4.6.1 Detailed Description

File containing the definition of the enumerations that can be used as types for the attributes.

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

### 4.6.2 Function Documentation

#### 4.6.2.1 void freeEnum ( Enum \* enu, int freeltself )

Free the [Enum](#).

##### Parameters

|                   |                                                                       |
|-------------------|-----------------------------------------------------------------------|
| <i>enu</i>        | A pointer to the <a href="#">Enum</a> to be freed                     |
| <i>freeltself</i> | Boolean to know wether the <a href="#">Enum</a> is to be freed or not |

#### 4.6.2.2 void freeEnumType ( EnumType \* enuty, int freeltself )

Free the [EnumType](#).

##### Parameters

|                   |                                                                           |
|-------------------|---------------------------------------------------------------------------|
| <i>enuty</i>      | A pointer to the <a href="#">EnumType</a> to be freed                     |
| <i>freeltself</i> | Boolean to know wether the <a href="#">EnumType</a> is to be freed or not |

Free the [EnumType](#).

FOPPOLO Gaël PHILIP Bastien

## 4.7 types/example.h File Reference

File containing the definition of the examples.

```
#include "vector.h"
#include "object.h"
```

### Data Structures

- struct [Example](#)

*All the objects composing an example (or a counter-example)*



## Typedefs

- typedef struct [Example](#) **Example**

## Functions

- void [initExample](#) ([Example](#) \*exp)  
*Init the example object.*
- void [freeExample](#) ([Example](#) \*exp, int freeltself)  
*Free the example object.*

### 4.7.1 Detailed Description

File containing the definition of the examples.

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

### 4.7.2 Function Documentation

#### 4.7.2.1 void [freeExample](#) ( [Example](#) \* *exp*, int *freeltself* )

Free the example object.

##### Parameters

|                   |                                                                          |
|-------------------|--------------------------------------------------------------------------|
| <i>exp</i>        | A pointer to the example to be freed                                     |
| <i>freeltself</i> | Boolean to know wether the <a href="#">Example</a> is to be freed or not |

#### 4.7.2.2 void [initExample](#) ( [Example](#) \* *exp* )

Init the example object.

##### Parameters

|            |                                            |
|------------|--------------------------------------------|
| <i>exp</i> | A pointer to the example to be initialized |
|------------|--------------------------------------------|

Init the example object.

FOPPOLO Gaël PHILIP Bastien

## 4.8 types/examples.h File Reference

File containing the definition of the examples (the all example file content)

```
#include "example.h"
```

## Data Structures

- struct [Examples](#)

*Structure that contains the examples and counter-examples of the parsed example file.*

## Typedefs

- typedef struct [Examples](#) **Examples**

## Functions

- void [initExamples](#) ([Examples](#) \*exps)  
*Init the [Examples](#) structure.*
- void [freeExamples](#) ([Examples](#) \*exps)  
*Free the [Examples](#) structure.*

### 4.8.1 Detailed Description

File containing the definition of the examples (the all example file content)

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

### 4.8.2 Function Documentation

#### 4.8.2.1 void [freeExamples](#) ( **Examples** \* *exps* )

Free the [Examples](#) structure.

##### Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>exps</i> | A pointer to the <a href="#">Examples</a> to be freed |
|-------------|-------------------------------------------------------|

#### 4.8.2.2 void [initExamples](#) ( **Examples** \* *exps* )

Init the [Examples](#) structure.

##### Parameters

|             |                                                                       |
|-------------|-----------------------------------------------------------------------|
| <i>exps</i> | A pointer to the <a href="#">Examples</a> structure to be initialized |
|-------------|-----------------------------------------------------------------------|

Init the [Examples](#) structure.

FOPPOLO Gaël PHILIP Bastien

## 4.9 types/interval.h File Reference

File containing the definition of the interval type.

### Data Structures

- struct [Interval](#)  
*Structure that contains a signed integer interval.*

### Typedefs

- typedef struct [Interval](#) **Interval**

### Functions

- void [addToInterval](#) ([Interval](#) \*inter, int x)  
*Change the interval (if needed) to contain a new value.*

#### 4.9.1 Detailed Description

File containing the definition of the interval type.

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

#### 4.9.2 Function Documentation

##### 4.9.2.1 void addToInterval ( [Interval](#) \* *inter*, int *x* )

Change the interval (if needed) to contain a new value.

#### Parameters

|              |                                    |
|--------------|------------------------------------|
| <i>inter</i> | The interval to change             |
| <i>x</i>     | The integer to add in the interval |

Change the interval (if needed) to contain a new value.

FOPPOLO Gaël PHILIP Bastien

## 4.10 types/model-attribute.h File Reference

File containing the definition of the attributes (as defined by the model file)

```
#include <string.h>
#include "model-type.h"
```

### Data Structures

- struct [ModelAttribute](#)  
*Contains the definition of an attribute and its type.*

### Typedefs

- typedef struct [ModelAttribute](#) **ModelAttribute**

### Functions

- void [freeModelAttribute](#) ([ModelAttribute](#) \*ma, int freeltself)  
*Free the [ModelAttribute](#).*

#### 4.10.1 Detailed Description

File containing the definition of the attributes (as defined by the model file)

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

#### 4.10.2 Function Documentation

##### 4.10.2.1 void freeModelAttribute ( [ModelAttribute](#) \* ma, int freeltself )

Free the [ModelAttribute](#).

#### Parameters

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| <i>ma</i>         | A pointer to the <a href="#">ModelAttribute</a> to be freed                     |
| <i>freeltself</i> | Boolean to know wether the <a href="#">ModelAttribute</a> is to be freed or not |

Free the [ModelAttribute](#).

FOPPOLO Gaël PHILIP Bastien

## 4.11 types/model-type.h File Reference

File containing the definition of an attribute's type.

```
#include "attribute-types.h"
#include "interval.h"
#include "enum.h"
#include "tree.h"
```

### Data Structures

- struct [ModelType](#)  
*Structure that contains the definition of the type.*

### Typedefs

- typedef struct [ModelType](#) **ModelType**

### Functions

- void [freeModelType](#) ([ModelType](#) \*mt, int freeltself)  
*Free the [ModelType](#).*

#### 4.11.1 Detailed Description

File containing the definition of an attribute's type.

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

#### 4.11.2 Function Documentation

##### 4.11.2.1 void [freeModelType](#) ( [ModelType](#) \* mt, int *freeltself* )

Free the [ModelType](#).

#### Parameters

|                   |                                                                            |
|-------------------|----------------------------------------------------------------------------|
| <i>mt</i>         | A pointer to the <a href="#">ModelType</a> to be freed                     |
| <i>freeltself</i> | Boolean to know wether the <a href="#">ModelType</a> is to be freed or not |

Free the [ModelType](#).

FOPPOLO Gaël PHILIP Bastien

## 4.12 types/model.h File Reference

File containing the definition of the model parsed in the model file.

```
#include <string.h>
#include "model-attribute.h"
#include "vector.h"
```

### Data Structures

- struct [Model](#)

*Contains the attributes and relations definitions found in the model file.*

### Typedefs

- typedef struct [Model](#) **Model**

### Functions

- void [initModel](#) ([Model](#) \*mdl)  
*Init the model.*
- void [freeModel](#) ([Model](#) \*mdl)  
*Free the [Model](#) struct created while parsing the config file.*
- int [getEnumId](#) (const char \*str, [Model](#) \*mdl, unsigned int index)  
*Returns the identifier of the enumeration item whose name is given as a parameter.*
- int [getTreeId](#) (const char \*str, [Model](#) \*mdl, unsigned int index)  
*Returns the identifier of the tree node or tree leaf whose name is given as a parameter.*
- char \* [getEnumStr](#) (int id, [Model](#) \*mdl, unsigned int index)  
*Returns the name of the enumeration item whose identifier is given as a parameter.*
- char \* [getTreeStr](#) (int id, [Model](#) \*mdl, unsigned int index)  
*Returns the name of the tree node or tree leaf whose identifier is given as a parameter.*

### 4.12.1 Detailed Description

File containing the definition of the model parsed in the model file.

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

### 4.12.2 Function Documentation

#### 4.12.2.1 void freeModel ( [Model](#) \* mdl )

Free the [Model](#) struct created while parsing the config file.

## Parameters

|            |                                          |
|------------|------------------------------------------|
| <i>mdl</i> | The <a href="#">Model</a> struct to free |
|------------|------------------------------------------|

4.12.2.2 int getEnumId ( const char \* *str*, **Model** \* *mdl*, unsigned int *index* )

Returns the identifier of the enumeration item whose name is given as a parameter.

## Parameters

|              |                                                               |
|--------------|---------------------------------------------------------------|
| <i>str</i>   | The enumeration item's name of which the identifier is needed |
| <i>mdl</i>   | The model                                                     |
| <i>index</i> | The index of the attribute in the model                       |

## Returns

Returns the identifier if found, -1 otherwise

4.12.2.3 char\* getEnumStr ( int *id*, **Model** \* *mdl*, unsigned int *index* )

Returns the name of the enumeration item whose identifier is given as a parameter.

## Parameters

|              |                                                               |
|--------------|---------------------------------------------------------------|
| <i>id</i>    | The enumeration item's identifier of which the name is needed |
| <i>mdl</i>   | The model                                                     |
| <i>index</i> | The index of the attribute in the model                       |

## Returns

Returns the name if found, NULL otherwise

4.12.2.4 int getTreeId ( const char \* *str*, **Model** \* *mdl*, unsigned int *index* )

Returns the identifier of the tree node or tree leaf whose name is given as a parameter.

## Parameters

|              |                                                         |
|--------------|---------------------------------------------------------|
| <i>str</i>   | The node or leaf name of which the identifier is needed |
| <i>mdl</i>   | The model                                               |
| <i>index</i> | The index of the attribute in the model                 |

## Returns

Returns the identifier if found, -1 otherwise

#### 4.12.2.5 `char* getTreeStr ( int id, Model * mdl, unsigned int index )`

Returns the name of the tree node or tree leaf whose identifier is given as a parameter.

##### Parameters

|              |                                                         |
|--------------|---------------------------------------------------------|
| <i>id</i>    | The node or leaf identifier of which the name is needed |
| <i>mdl</i>   | The model                                               |
| <i>index</i> | The index of the attribute in the model                 |

##### Returns

Returns the name if found, NULL otherwise

#### 4.12.2.6 `void initModel ( Model * mdl )`

Init the model.

##### Parameters

|            |                                |
|------------|--------------------------------|
| <i>mdl</i> | A pointer to the model to init |
|------------|--------------------------------|

Init the model.

FOPPOLO Gaël PHILIP Bastien

## 4.13 `types/object.h` File Reference

File containing the definition of the objects.

```
#include "vector.h"
#include "attribute.h"
```

### Data Structures

- struct [Object](#)  
*Contains all the attributes and relations that compose an object.*

### Typedefs

- typedef struct [Object](#) **Object**



## Functions

- void `initObject` (`Object` \*obj)  
*Init the object.*
- void `freeObject` (`Object` \*obj, int freeltself)  
*Free the object previously initialized by initObject.*

### 4.13.1 Detailed Description

File containing the definition of the objects.

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

### 4.13.2 Function Documentation

#### 4.13.2.1 void freeObject ( Object \* obj, int freeltself )

Free the object previously initialized by initObject.

##### Parameters

|                   |                                                                      |
|-------------------|----------------------------------------------------------------------|
| <i>obj</i>        | A pointer to the object to free                                      |
| <i>freeltself</i> | Boolean to know wether the <code>Object</code> is to be freed or not |

#### 4.13.2.2 void initObject ( Object \* obj )

Init the object.

##### Parameters

|            |                                 |
|------------|---------------------------------|
| <i>obj</i> | A pointer to the object to init |
|------------|---------------------------------|

Init the object.

FOPPOLO Gaël PHILIP Bastien

## 4.14 types/out-attribute.h File Reference

File containing the definition of the out-attributes.

```
#include "attribute-types.h"
#include "interval.h"
#include "out-enum.h"
```

## Data Structures

- struct [OutAttribute](#)

*Represents an attribute used by the solution and the value it holds.*

## Typedefs

- typedef struct [OutAttribute](#) **OutAttribute**

## Functions

- void [freeOutAttribute](#) ([OutAttribute](#) \*oa, int freeltself)

*Free the [OutAttribute](#).*

### 4.14.1 Detailed Description

File containing the definition of the out-attributes.

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

### 4.14.2 Function Documentation

#### 4.14.2.1 void [freeOutAttribute](#) ( [OutAttribute](#) \* oa, int *freeltself* )

Free the [OutAttribute](#).

#### Parameters

|                   |                                                                               |
|-------------------|-------------------------------------------------------------------------------|
| <i>oa</i>         | A pointer to the <a href="#">OutAttribute</a> to free                         |
| <i>freeltself</i> | Boolean to know wether the <a href="#">OutAttribute</a> is to be freed or not |

Free the [OutAttribute](#).

FOPPOLO Gaël PHILIP Bastien

## 4.15 types/out-enum.h File Reference

File containing the definition of the enumeration extracts.

```
#include "vector.h"
```

## Data Structures

- struct [OutEnum](#)

*Contains multiple enumeration items When combining multiple [Object](#), each enumeration item is to be stored, this structure does that.*

## Typedefs

- typedef struct [OutEnum](#) **OutEnum**

## Functions

- void [initOutEnum](#) ([OutEnum](#) \*oenu)  
*Init the output enum.*
- void [freeOutEnum](#) ([OutEnum](#) \*oenu, int freeltself)  
*Free the [OutEnum](#) previously initialized by initEnum.*

### 4.15.1 Detailed Description

File containing the definition of the enumeration extracts.

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

### 4.15.2 Function Documentation

#### 4.15.2.1 void [freeOutEnum](#) ( [OutEnum](#) \* *oenu*, int *freeltself* )

Free the [OutEnum](#) previously initialized by initEnum.

##### Parameters

|                   |                                                                          |
|-------------------|--------------------------------------------------------------------------|
| <i>oenu</i>       | A pointer to the enum to free                                            |
| <i>freeltself</i> | Boolean to know wether the <a href="#">OutEnum</a> is to be freed or not |

#### 4.15.2.2 void [initOutEnum](#) ( [OutEnum](#) \* *oenu* )

Init the output enum.

##### Parameters

|             |                               |
|-------------|-------------------------------|
| <i>oenu</i> | A pointer to the enum to init |
|-------------|-------------------------------|

Init the output enum.

FOPPOLO Gaël PHILIP Bastien

## 4.16 types/out-object.h File Reference

File containing the definition of the out-objects. Generated when combining multiple objects.

```
#include "out-attribute.h"
```

### Data Structures

- struct [OutObject](#)  
*Contains all the attributes and relations that compose an outObject.*

### Typedefs

- typedef struct [OutObject](#) **OutObject**

### Functions

- void [initOutObject](#) ([OutObject](#) \*oo)  
*Init the outobject.*
- void [freeOutObject](#) ([OutObject](#) \*oo, int freeltself)  
*Free the outobject previously initialized by initOutObject.*

#### 4.16.1 Detailed Description

File containing the definition of the out-objects. Generated when combining multiple objects.

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

#### 4.16.2 Function Documentation

##### 4.16.2.1 void [freeOutObject](#) ( [OutObject](#) \* oo, int *freeltself* )

Free the outobject previously initialized by [initOutObject](#).

#### Parameters

|                   |                                                                            |
|-------------------|----------------------------------------------------------------------------|
| <i>oo</i>         | A pointer to the outobject to free                                         |
| <i>freeltself</i> | Boolean to know wether the <a href="#">OutObject</a> is to be freed or not |

## 4.16.2.2 void initOutObject ( OutObject \* oo )

Init the outobject.

## Parameters

|           |                                    |
|-----------|------------------------------------|
| <i>oo</i> | A pointer to the outobject to init |
|-----------|------------------------------------|

Init the outobject.

FOPPOLO Gaël PHILIP Bastien

## 4.17 types/solution.h File Reference

File containing the definition of the solution.

```
#include "vector.h"
#include "out-object.h"
```

## Data Structures

- struct [Solution](#)  
*Contains all the possible solutions.*

## Typedefs

- typedef struct [Solution](#) **Solution**

## Functions

- void [initSolution](#) ([Solution](#) \*sol)  
*Init the solution.*
- void [freeSolution](#) ([Solution](#) \*sol)  
*Free the solution previously initialized by initSolution.*

## 4.17.1 Detailed Description

File containing the definition of the solution.

## Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

## 4.17.2 Function Documentation

## 4.17.2.1 void freeSolution ( Solution \* sol )

Free the solution previously initialized by initSolution.

## Parameters

|            |                                   |
|------------|-----------------------------------|
| <i>sol</i> | A pointer to the solution to free |
|------------|-----------------------------------|

4.17.2.2 void initSolution ( Solution \* *sol* )

Init the solution.

## Parameters

|            |                                   |
|------------|-----------------------------------|
| <i>sol</i> | A pointer to the solution to init |
|------------|-----------------------------------|

Init the solution.

FOPPOLO Gaël PHILIP Bastien

## 4.18 types/string-type.h File Reference

File containing the definition of the [String](#) type and some tools to use it.

```
#include <stdlib.h>
```

### Data Structures

- struct [String](#)  
*Dynamic string handler.*

### Typedefs

- typedef struct [String](#) **String**

### Functions

- [String strInit](#) (char \*str)  
*Transforms a char\* to a string. The char\* MUST be a nul terminated array allocated with malloc.*
- unsigned int [strLength](#) ([String](#) \*str)  
*Returns the current string length.*
- void [strPush](#) ([String](#) \*str, char c)  
*Adds a character at the end of the string.*
- void [strPushStr](#) ([String](#) \*str, char \*str2)  
*Add a string at the end of the current string.*
- char \* [strDuplicate](#) (char \*str)  
*Create a perfect copy of the string given. Used when a malloc created string is needed.*

### 4.18.1 Detailed Description

File containing the definition of the [String](#) type and some tools to use it.

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

### 4.18.2 Function Documentation

#### 4.18.2.1 `char* strDuplicate ( char * str )`

Create a perfect copy of the string given. Used when a malloc created string is needed.

##### Parameters

|            |                     |
|------------|---------------------|
| <i>str</i> | The original string |
|------------|---------------------|

##### Returns

A new string created with malloc

#### 4.18.2.2 `String strInit ( char * str )`

Transforms a char\* to a string. The char\* MUST be a nul terminated array allocated with malloc.

##### Parameters

|            |                                                                  |
|------------|------------------------------------------------------------------|
| <i>str</i> | The original string (the original is used, no copy is performed) |
|------------|------------------------------------------------------------------|

##### Returns

The newly created string structure

Transforms a char\* to a string. The char\* MUST be a nul terminated array allocated with malloc.

FOPPOLO Gaël PHILIP Bastien

#### 4.18.2.3 `unsigned int strLength ( String * str )`

Returns the current string length.

##### Parameters

|            |                                            |
|------------|--------------------------------------------|
| <i>str</i> | The string of which the length is returned |
|------------|--------------------------------------------|

### Returns

The length of the string

#### 4.18.2.4 void strPush ( String \* *str*, char *c* )

Adds a character at the end of the string.

### Parameters

|            |                                                  |
|------------|--------------------------------------------------|
| <i>str</i> | The string at the end of which the char is added |
| <i>c</i>   | The char to be added                             |

#### 4.18.2.5 void strPushStr ( String \* *str*, char \* *str2* )

Add a string at the end of the current string.

### Parameters

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>str</i>  | The string at the end of which the second string is added |
| <i>str2</i> | The string to be added (need to be nul terminated)        |

## 4.19 types/tree.h File Reference

File containing the definition of the trees.

```
#include <stdio.h>
#include "vector.h"
```

### Data Structures

- struct [Tree](#)  
*Defines the trees.*

### Macros

- #define [max](#)(a, b) ((a) > (b) ? (a) : (b))  
*Computes the maximum of a and b.*

### Typedefs

- typedef struct [Tree](#) **Tree**



## Functions

- **Tree** \* **createLeaf** (int id, char \*str)  
*Create a new leaf.*
- **Tree** \* **createNode** (int id, char \*str, **Tree** \*child)  
*Create a new node.*
- **Tree** \* **addChild** (**Tree** \*node, **Tree** \*child)  
*Add a child to a node.*
- int **isLeaf** (**Tree** \*t)  
*Check wether the tree is a leaf or not.*
- int **height** (**Tree** \*t)  
*Get the height of a tree.*
- int **depth** (**Tree** \*root, int id)  
*Get the depth of a node in the tree.*
- **Tree** \* **LCA** (**Tree** \*root, int id1, int id2)  
*Find the lowest common ancestor We traverse from root to leaf. When we find a node matching at least one value, we pass it to its parent. The parent tests wether a child contains the value or not. If yes, the parent is the LCA, otherwise, we pass its parent, up to root. What is passed is the lower node or NULL.*
- void **freeTree** (**Tree** \*t)  
*Free the tree.*

### 4.19.1 Detailed Description

File containing the definition of the trees.

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

### 4.19.2 Function Documentation

#### 4.19.2.1 **Tree**\* **addChild** ( **Tree** \* *node*, **Tree** \* *child* )

Add a child to a node.

##### Parameters

|              |                                    |
|--------------|------------------------------------|
| <i>node</i>  | The node to which to add the child |
| <i>child</i> | The child to add to our node       |

##### Returns

The modified node (same as given as parameter)

#### 4.19.2.2 **Tree**\* **createLeaf** ( int *id*, char \* *str* )

Create a new leaf.

**Parameters**

|            |                                                                        |
|------------|------------------------------------------------------------------------|
| <i>id</i>  | The value to store in the leaf                                         |
| <i>str</i> | <a href="#">String</a> that represents the real name of what is stored |

**Returns**

A new leaf

Create a new leaf.

FOPPOLO Gaël PHILIP Bastien

**4.19.2.3 Tree\* createNode ( int *id*, char \* *str*, Tree \* *child* )**

Create a new node.

**Parameters**

|              |                                                                        |
|--------------|------------------------------------------------------------------------|
| <i>id</i>    | The value to store in the node                                         |
| <i>str</i>   | <a href="#">String</a> that represents the real name of what is stored |
| <i>child</i> | The child to add to our new node                                       |

**Returns**

A new node

**4.19.2.4 int depth ( Tree \* *root*, int *id* )**

Get the depth of a node in the tree.

**Parameters**

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>root</i> | The root of the tree                            |
| <i>id</i>   | The id of the node of which the depth is needed |

**Returns**

The depth of the node in the tree

**4.19.2.5 void freeTree ( Tree \* *t* )**

Free the tree.

## Parameters

|          |                     |
|----------|---------------------|
| <i>t</i> | Pointer to the tree |
|----------|---------------------|

4.19.2.6 int height ( Tree \* *t* )

Get the height of a tree.

## Parameters

|          |                                        |
|----------|----------------------------------------|
| <i>t</i> | The tree of which the height is needed |
|----------|----------------------------------------|

## Returns

The height of the tree

4.19.2.7 int isLeaf ( Tree \* *t* )

Check wether the tree is a leaf or not.

## Parameters

|          |                   |
|----------|-------------------|
| <i>t</i> | The tree to check |
|----------|-------------------|

## Returns

Returns 1 if the parameter is a leaf, 0 otherwise

4.19.2.8 Tree\* LCA ( Tree \* *root*, int *id1*, int *id2* )

Find the lowest common ancestor We traverse from root to leaf. When we find a node matching at least one value, we pass it to its parent. The parent tests wether a child contains the value or not. If yes, the parent is the LCA, otherwise, we pass its parent, up to root. What is passed is the lower node or NULL.

## Parameters

|             |                      |
|-------------|----------------------|
| <i>root</i> | The root of the tree |
| <i>id1</i>  | The first value      |
| <i>id2</i>  | The second value     |

## Returns

The lowest common ancestor (node or leaf)

## 4.20 types/vector.h File Reference

File containing the definition of the vectors (dynamic & generic arrays)

```
#include <stdlib.h>
```

### Macros

- `#define Vector(t) struct {int size, capacity; t *data; }`  
*define a dynamic array*
- `#define vectInit(vect) ((vect).size = (vect).capacity = 0, (vect).data = 0)`  
*Init the vector.*
- `#define vectFree(vect) free((vect).data)`  
*Free the vector.*
- `#define vectAt(vect, index) ((vect).data[(index)])`  
*Returns the element at a certain index of the array.*
- `#define vectSize(vect) ((vect).size)`  
*Return the size of the array.*
- `#define vectPush(type, vect, value)`  
*Append an element at the end of the vector.*
- `#define vectIndexOf(vect, value, out)`  
*Search for an element in the vector. Returns its index if found.*
- `#define vectRemoveLast(vect)`  
*Remove the last element in the vector.*

### 4.20.1 Detailed Description

File containing the definition of the vectors (dynamic & generic arrays)

#### Author

Bastien Philip (ebatsin)  
Gaël Foppolo (gaelfoppolo)

### 4.20.2 Macro Definition Documentation

#### 4.20.2.1 `#define vectAt( vect, index ) ((vect).data[(index)])`

Returns the element at a certain index of the array.

#### Parameters

|              |                                                          |
|--------------|----------------------------------------------------------|
| <i>vect</i>  | The vector of which to access the element                |
| <i>index</i> | The index of the element (between 0 and vector size - 1) |

## Returns

The element (directly. You can use this as a left value)

#### 4.20.2.2 #define vectFree( vect ) free((vect).data)

Free the vector.

### Parameters

|             |                              |
|-------------|------------------------------|
| <i>vect</i> | The vector to be initialized |
|-------------|------------------------------|

#### 4.20.2.3 #define vectIndexOf( vect, value, out )

**Value:**

```
do {
 \out = -1;
 \for(int i = 0; i < vectSize(vect); ++i) {
 \if(vectAt(vect, i) == value) {
 \out = i;
 \break;
 }
 }
} while(0)
```

Search for an element in the vector. Returns its index if found.

## Parameters

|              |                                                                                                                   |
|--------------|-------------------------------------------------------------------------------------------------------------------|
| <i>vect</i>  | The vector to search in                                                                                           |
| <i>value</i> | The value to search for                                                                                           |
| <i>out</i>   | An integer that will hold the return value (either the index if found ou -1 if the element is not int the vector) |

#### 4.20.2.4 #define vectInit( vect ) ((vect).size = (vect).capacity = 0, (vect).data = 0)

Init the vector.

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>vect</i> | The vector to be initialized |
|-------------|------------------------------|

#### 4.20.2.5 #define Vector( t ) struct {int size, capacity; t \*data; }

define a dynamic array



**Parameters**

|             |                                    |
|-------------|------------------------------------|
| <i>vect</i> | The array of which to get the size |
|-------------|------------------------------------|

**Returns**

The size of the array





# Index

- addChild
  - tree.h, [49](#)
- addToInterval
  - interval.h, [35](#)
- app/core.h, [15](#)
- app/output.h, [20](#)
- Attribute, [5](#)
- cPrint
  - output.h, [21](#)
- combiOutObjectObject
  - core.h, [16](#)
- compareOutObjects
  - core.h, [16](#)
- core.h
  - combiOutObjectObject, [16](#)
  - compareOutObjects, [16](#)
  - genAllCombi, [16](#)
  - genAllRelations, [18](#)
  - genGeneralisation, [18](#)
  - genSpecificity, [18](#)
  - getIndex, [18](#)
  - initAllCombi, [19](#)
  - initOutObjectWithObject, [19](#)
  - nbCombi, [19](#)
- createLeaf
  - tree.h, [49](#)
- createNode
  - tree.h, [50](#)
- depth
  - tree.h, [50](#)
- Enum, [5](#)
- enum.h
  - freeEnum, [32](#)
  - freeEnumType, [32](#)
- EnumType, [6](#)
- Example, [6](#)
- example.h
  - freeExample, [33](#)
  - initExample, [33](#)
- Examples, [7](#)
- examples.h
  - freeExamples, [34](#)
  - initExamples, [34](#)
- extractVerbosityFromArg
  - output.h, [21](#)
- freeEnum
  - enum.h, [32](#)
- freeEnumType
  - enum.h, [32](#)
- freeExample
  - example.h, [33](#)
- freeExamples
  - examples.h, [34](#)
- freeModel
  - model.h, [38](#)
- freeModelAttribute
  - model-attribute.h, [36](#)
- freeModelType
  - model-type.h, [37](#)
- freeObject
  - object.h, [41](#)
- freeOutAttribute
  - out-attribute.h, [42](#)
- freeOutEnum
  - out-enum.h, [43](#)
- freeOutObject
  - out-object.h, [44](#)
- freeSolution
  - solution.h, [45](#)
- freeTree
  - tree.h, [50](#)
- genAllCombi
  - core.h, [16](#)
- genAllRelations
  - core.h, [18](#)
- genGeneralisation
  - core.h, [18](#)
- genOutput
  - output.h, [22](#)
- genSpecificity
  - core.h, [18](#)
- getAttributePosition
  - parsers.h, [24](#)
- getEnumId
  - model.h, [39](#)
- getEnumStr
  - model.h, [39](#)
- getIncludeFile
  - parsers.h, [24](#)
- getIndex
  - core.h, [18](#)
- getNextExample
  - parsers.h, [25](#)
- getRelationPosition
  - parsers.h, [25](#)

- getTreeld
  - model.h, [39](#)
- getTreeStr
  - model.h, [39](#)
- height
  - tree.h, [51](#)
- initAllCombi
  - core.h, [19](#)
- initExample
  - example.h, [33](#)
- initExamples
  - examples.h, [34](#)
- initModel
  - model.h, [40](#)
- initObject
  - object.h, [41](#)
- initOutEnum
  - out-enum.h, [43](#)
- initOutObject
  - out-object.h, [45](#)
- initOutObjectWithObject
  - core.h, [19](#)
- initSolution
  - solution.h, [46](#)
- Interval, [7](#)
- interval.h
  - addToInterval, [35](#)
- isLeaf
  - tree.h, [51](#)
- isValidAttrChar
  - parsers.h, [25](#)
- LCA
  - tree.h, [51](#)
- loadConfigFile
  - parsers.h, [26](#)
- loadExampleFile
  - parsers.h, [26](#)
- Model, [8](#)
- model-attribute.h
  - freeModelAttribute, [36](#)
- model-type.h
  - freeModelType, [37](#)
- model.h
  - freeModel, [38](#)
  - getEnumId, [39](#)
  - getEnumStr, [39](#)
  - getTreeld, [39](#)
  - getTreeStr, [39](#)
  - initModel, [40](#)
- ModelAttribute, [8](#)
- ModelType, [9](#)
- nbCombi
  - core.h, [19](#)
- Object, [10](#)
- object.h
  - freeObject, [41](#)
  - initObject, [41](#)
- ObjectIndice, [10](#)
- out-attribute.h
  - freeOutAttribute, [42](#)
- out-enum.h
  - freeOutEnum, [43](#)
  - initOutEnum, [43](#)
- out-object.h
  - freeOutObject, [44](#)
  - initOutObject, [45](#)
- OutAttribute, [11](#)
- OutEnum, [11](#)
- OutObject, [12](#)
- output
  - output.h, [22](#)
- output.h
  - cPrint, [21](#)
  - extractVerbosityFromArg, [21](#)
  - genOutput, [22](#)
  - output, [22](#)
  - setOutputImportance, [22](#)
- parseAttrName
  - parsers.h, [26](#)
- parseAttrType
  - parsers.h, [26](#)
- parseAttrTypeEnum
  - parsers.h, [27](#)
- parseAttrTypeInterval
  - parsers.h, [27](#)
- parseAttrTypeTree
  - parsers.h, [27](#)
- parseAttrValue
  - parsers.h, [28](#)
- parseConfigLine
  - parsers.h, [28](#)
- parseExample
  - parsers.h, [28](#)
- parseExampleObject
  - parsers.h, [29](#)
- parser/parsers.h, [22](#)
- parsers.h
  - getAttributePosition, [24](#)
  - getIncludeFile, [24](#)
  - getNextExample, [25](#)
  - getRelationPosition, [25](#)
  - isValidAttrChar, [25](#)
  - loadConfigFile, [26](#)
  - loadExampleFile, [26](#)
  - parseAttrName, [26](#)
  - parseAttrType, [26](#)
  - parseAttrTypeEnum, [27](#)
  - parseAttrTypeInterval, [27](#)
  - parseAttrTypeTree, [27](#)
  - parseAttrValue, [28](#)
  - parseConfigLine, [28](#)
  - parseExample, [28](#)

- parseExampleObject, [29](#)
- printIndent, [29](#)
- readFileSpaces, [29](#)
- readTil, [30](#)
- printIndent
  - parsers.h, [29](#)
- readFileSpaces
  - parsers.h, [29](#)
- readTil
  - parsers.h, [30](#)
- setOutputImportance
  - output.h, [22](#)
- Solution, [12](#)
- solution.h
  - freeSolution, [45](#)
  - initSolution, [46](#)
- strDuplicate
  - string-type.h, [47](#)
- strInit
  - string-type.h, [47](#)
- strLength
  - string-type.h, [47](#)
- strPush
  - string-type.h, [48](#)
- strPushStr
  - string-type.h, [48](#)
- String, [13](#)
- string-type.h
  - strDuplicate, [47](#)
  - strInit, [47](#)
  - strLength, [47](#)
  - strPush, [48](#)
  - strPushStr, [48](#)
- StringVector, [13](#)
- Tree, [14](#)
- tree.h
  - addChild, [49](#)
  - createLeaf, [49](#)
  - createNode, [50](#)
  - depth, [50](#)
  - freeTree, [50](#)
  - height, [51](#)
  - isLeaf, [51](#)
  - LCA, [51](#)
- types/attribute-types.h, [30](#)
- types/attribute.h, [30](#)
- types/enum.h, [31](#)
- types/example.h, [32](#)
- types/examples.h, [33](#)
- types/interval.h, [35](#)
- types/model-attribute.h, [36](#)
- types/model-type.h, [37](#)
- types/model.h, [38](#)
- types/object.h, [40](#)
- types/out-attribute.h, [41](#)
- types/out-enum.h, [42](#)
- types/out-object.h, [44](#)
- types/solution.h, [45](#)
- types/string-type.h, [46](#)
- types/tree.h, [48](#)
- types/vector.h, [52](#)
- vectAt
  - vector.h, [52](#)
- vectFree
  - vector.h, [53](#)
- vectIndexOf
  - vector.h, [53](#)
- vectInit
  - vector.h, [53](#)
- vectPush
  - vector.h, [54](#)
- vectRemoveLast
  - vector.h, [54](#)
- vectSize
  - vector.h, [54](#)
- Vector
  - vector.h, [53](#)
- vector.h
  - vectAt, [52](#)
  - vectFree, [53](#)
  - vectIndexOf, [53](#)
  - vectInit, [53](#)
  - vectPush, [54](#)
  - vectRemoveLast, [54](#)
  - vectSize, [54](#)
  - Vector, [53](#)