Package ‘rminizinc’
October 14, 2022

Type Package
Title R Interface to 'MiniZinc'
Version 0.0.8
Author Akshit Achara, Lars Kotthoff, Hans W. Borchers, Guido Tack
Maintainer Akshit Achara <acharaakshit@gmail.com>
URL https://github.com/acharaakshit/RMiniZinc
BugReports https://github.com/acharaakshit/RMiniZinc/issues

Description Constraint optimization, or constraint programming, is the name given to identifying feasible solutions out of a very large set of candidates, where the problem can be modeled in terms of arbitrary constraints. 'MiniZinc' is a free and open-source constraint modeling language. Constraint satisfaction and discrete optimization problems can be formulated in a high-level modeling language. Models are compiled into an intermediate representation that is understood by a wide range of solvers. 'MiniZinc' itself provides several solvers, for instance 'GeCode'. R users can use the package to solve constraint programming problems without using 'MiniZinc' directly, modify existing 'MiniZinc' models and also create their own models.

License Mozilla Public License Version 2.0
Encoding UTF-8
LazyData true
RoxygenNote 7.1.2
Depends R (>= 3.5.0), json
Imports R6, checkmate, Rcpp, rlang, rlist
LinkingTo Rcpp
Suggests knitr, rmarkdown, testthat, stringr
SystemRequirements pandoc (>=1.14, needed for the vignette)
VignetteBuilder knitr
Biarch true
NeedsCompilation yes
Repository CRAN
Date/Publication 2021-10-15 04:40:02 UTC
R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>rminizinc-package</td>
<td>3</td>
</tr>
<tr>
<td>Annotation</td>
<td>4</td>
</tr>
<tr>
<td>Array</td>
<td>5</td>
</tr>
<tr>
<td>ArrayAccess</td>
<td>8</td>
</tr>
<tr>
<td>ArrDomainDecl</td>
<td>10</td>
</tr>
<tr>
<td>AssignItem</td>
<td>10</td>
</tr>
<tr>
<td>assignment</td>
<td>12</td>
</tr>
<tr>
<td>assignment_2</td>
<td>13</td>
</tr>
<tr>
<td>BinOp</td>
<td>13</td>
</tr>
<tr>
<td>Bool</td>
<td>16</td>
</tr>
<tr>
<td>BoolArrDecl</td>
<td>17</td>
</tr>
<tr>
<td>BoolDecl</td>
<td>17</td>
</tr>
<tr>
<td>boolExpressions</td>
<td>18</td>
</tr>
<tr>
<td>BoolSetDecl</td>
<td>18</td>
</tr>
<tr>
<td>Call</td>
<td>19</td>
</tr>
<tr>
<td>Comprehension</td>
<td>21</td>
</tr>
<tr>
<td>ConstraintItem</td>
<td>24</td>
</tr>
<tr>
<td>Expression</td>
<td>25</td>
</tr>
<tr>
<td>expressionDelete</td>
<td>26</td>
</tr>
<tr>
<td>Float</td>
<td>26</td>
</tr>
<tr>
<td>FloatArrDecl</td>
<td>27</td>
</tr>
<tr>
<td>FloatDecl</td>
<td>28</td>
</tr>
<tr>
<td>floatExpressions</td>
<td>28</td>
</tr>
<tr>
<td>FloatSetDecl</td>
<td>29</td>
</tr>
<tr>
<td>FloatSetVal</td>
<td>29</td>
</tr>
<tr>
<td>FloatVal</td>
<td>31</td>
</tr>
<tr>
<td>FunctionItem</td>
<td>32</td>
</tr>
<tr>
<td>Generator</td>
<td>34</td>
</tr>
<tr>
<td>getRModel</td>
<td>37</td>
</tr>
<tr>
<td>getType</td>
<td>37</td>
</tr>
<tr>
<td>get_missing_pars</td>
<td>37</td>
</tr>
<tr>
<td>helperDeleteExpression</td>
<td>38</td>
</tr>
<tr>
<td>helperDeleteItem</td>
<td>38</td>
</tr>
<tr>
<td>Id</td>
<td>38</td>
</tr>
<tr>
<td>IncludeItem</td>
<td>40</td>
</tr>
<tr>
<td>initExpression</td>
<td>41</td>
</tr>
<tr>
<td>initItem</td>
<td>42</td>
</tr>
<tr>
<td>Int</td>
<td>42</td>
</tr>
<tr>
<td>IntArrDecl</td>
<td>43</td>
</tr>
<tr>
<td>IntDecl</td>
<td>44</td>
</tr>
<tr>
<td>intExpressions</td>
<td>44</td>
</tr>
<tr>
<td>IntSetDecl</td>
<td>45</td>
</tr>
<tr>
<td>IntSetVal</td>
<td>45</td>
</tr>
<tr>
<td>IntVal</td>
<td>47</td>
</tr>
<tr>
<td>Ite</td>
<td>48</td>
</tr>
<tr>
<td>Item</td>
<td>51</td>
</tr>
</tbody>
</table>
Description

Load the required libraries used by most of the functions and classes

See Also

Useful links:

- https://github.com/acharaakshit/RMiniZinc
- Report bugs at https://github.com/acharaakshit/RMiniZinc/issues
Description

Create Annotations in MiniZinc

Public fields

.expVec  list of expressions
.delete_flag  used to delete items

Active bindings

.expVec  list of expressions
.delete_flag  used to delete items

Methods

Public methods:

• Annotation$new()
• Annotation$getExps()
• Annotation$setExps()
• Annotation$c_str()
• Annotation$getDeleteFlag()
• Annotation$delete()
• Annotation$clone()

Method new(): constructor

Usage:
Annotation$new(expVec)

Arguments:
expVec  vector of MiniZinc expressions

Method getExps(): get the list of expressions

Usage:
Annotation$getExps()

Method setExps(): set the list of expressions

Usage:
Annotation$setExps(expVec)

Arguments:
expVec  list of expressions to be set
**Method** `c_str()`: get the MiniZinc expression

*Usage:*

`Annotation$c_str()`

**Method** `getDeleteFlag()`: delete flag for internal use

*Usage:*

`Annotation$getDeleteFlag()`

**Method** `delete()`: delete the assignment item

*Usage:*

`Annotation$delete()`

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`Annotation$clone(deep = FALSE)`

*Arguments:*

- `deep` Whether to make a deep clone.

---

**Array**  
*create an array*

---

**Description**

Create an array in MiniZinc

**Super class**

`rminizinc::Expression -> Array`

**Public fields**

- `.exprVec` vector of value expressions
- `.dims` vector of dimension expressions
- `.delete_flag` used to delete items

**Active bindings**

- `.exprVec` vector of value expressions
- `.dims` vector of dimension expressions
- `.delete_flag` used to delete items
Methods

Public methods:

- `Array$new()`
- `Array$ndims()`
- `Array$getMinIndex()`
- `Array$getMaxIndex()`
- `Array$getMinIndex()`
- `Array$setMinIndex()`
- `Array$setMaxIndex()`
- `Array$setVal()`
- `Array$c_str()`
- `Array$getDeleteFlag()`
- `Array$delete()`
- `Array$clone()`

Method `new()`: constructor for an int literal

Usage:

```r
Array$new(exprVec, dimranges = NULL)
```

Arguments:

- `exprVec` list of expressions in the array
- `dimranges` list of min and max index of each dimension

Method `ndims()`: get the number of dimensions

Usage:

```r
Array$ndims()
```

Method `getMinIndex()`: get the minimum index of dimension i

Usage:

```r
Array$getMinIndex(i)
```

Arguments:

- `i` ith dimension

Method `getMaxIndex()`: get the maximum index of dimension i

Usage:

```r
Array$getMaxIndex(i)
```

Arguments:

- `i` ith dimension

Method `setMinIndex()`: set the minimum index of dimension i

Usage:

```r
Array$setMinIndex(i, minIndex)
```

Arguments:
i  dimension number
minIndex  integer for min index

**Method** `setMaxIndex()`: set the maximum index of dimension i

*Usage:*
`Array$setMaxIndex(i, maxIndex)`

*Arguments:*
i  dimension number
maxIndex  integer for max index

**Method** `getVal()`: get the ith element from vector

*Usage:*
`Array$getVal(i)`

*Arguments:*
i  index

**Method** `setVal()`: set the ith element from vector

*Usage:*
`Array$setVal(i, val)`

*Arguments:*
i  index
val  value of expression to be set

**Method** `c_str()`: return the MiniZinc representation

*Usage:*
`Array$c_str()`

**Method** `getDeleteFlag()`: delete flag for internal use

*Usage:*
`Array$getDeleteFlag()`

**Method** `delete()`: delete the assignment item

*Usage:*
`Array$delete()`

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*
`Array$clone(deep = FALSE)`

*Arguments:*
deep  Whether to make a deep clone.

**Examples**

```
newArray = Array$new(exprVec = c(Int$new(1), Int$new(2)))
newArray$c_str()
```
ArrayAccess

Description
Create ArrayAccess elements in MiniZinc

Super class
\texttt{rminizinc::Expression} \rightarrow \texttt{ArrayAccess}

Public fields
\begin{itemize}
  \item \texttt{v} \hspace{1em} \text{the id/value of array}
  \item \texttt{args} \hspace{1em} \text{arguments of the array}
  \item \texttt{delete\_flag} \hspace{1em} \text{used to delete items}
\end{itemize}

Active bindings
\begin{itemize}
  \item \texttt{v} \hspace{1em} \text{the id/value of array}
  \item \texttt{args} \hspace{1em} \text{arguments of the array}
  \item \texttt{delete\_flag} \hspace{1em} \text{used to delete items}
\end{itemize}

Methods
Public methods:
\begin{itemize}
  \item \texttt{ArrayAccess\_new()}
  \item \texttt{ArrayAccess\_getV()}
  \item \texttt{ArrayAccess\_setV()}
  \item \texttt{ArrayAccess\_nargs()}
  \item \texttt{ArrayAccess\_getArgs()}
  \item \texttt{ArrayAccess\_setArgs()}
  \item \texttt{ArrayAccess\_c\_str()}
  \item \texttt{ArrayAccess\_getDeleteFlag()}
  \item \texttt{ArrayAccess\_delete()}
  \item \texttt{ArrayAccess\_clone()}
\end{itemize}

Method \texttt{new()}: constructor

Usage:
\texttt{ArrayAccess\_new(v, args)}

Arguments:
\begin{itemize}
  \item \texttt{v} \hspace{1em} \text{the value/identifier of variable decl}
  \item \texttt{args} \hspace{1em} \text{the array indices}
\end{itemize}
**Method** `getV()`: get the array access value  
*Usage:*  
`ArrayAccess$getV()`

**Method** `setV()`: set the array access value  
*Usage:*  
`ArrayAccess$setV(val)`  
*Arguments:*  
val new array access value

**Method** `nargs()`: get the number of arguments  
*Usage:*  
`ArrayAccess$nargs()`

**Method** `getArgs()`: get the arguments  
*Usage:*  
`ArrayAccess$getArgs()`

**Method** `setArgs()`: set the arguments  
*Usage:*  
`ArrayAccess$setArgs(val)`  
*Arguments:*  
val new arguments

**Method** `c_str()`: return the MiniZinc representation  
*Usage:*  
`ArrayAccess$c_str()`

**Method** `getDeleteFlag()`: delete flag for internal use  
*Usage:*  
`ArrayAccess$getDeleteFlag()`

**Method** `delete()`: delete the assignment item  
*Usage:*  
`ArrayAccess$delete()`

**Method** `clone()`: The objects of this class are cloneable with this method.  
*Usage:*  
`ArrayAccess$clone(deep = FALSE)`  
*Arguments:*  
deep Whether to make a deep clone.

**Examples**

```plaintext```
vDecl1 = IntSetDecl(name = "SET", kind = "par")  
vDecl2 = IntArrDecl(name = "profit", kind = "par", ndim = 1,  
ind = list(vDecl1$getId()))  
newArrayAccess = ArrayAccess$new(v = vDecl2$getId(),  
args = list(IntDecl(name = "i", kind = "par")))
```
**ArrDomainDecl**

Declare a n-dimensional array with domain

**Usage**

`ArrDomainDecl(name, kind, dom, ndim)`

**Arguments**

- **name**: variable name
- **kind**: variable or parameter
- **dom**: domain
- **ndim**: number of dimensions

---

**AssignItem**

Assign values to variables in MiniZinc by creating an assignment item.

**Super class**

`rminizinc::Item -> AssignItem`

**Public fields**

- `.decl` associated declaration
- `.e` value to be assigned
- `.delete_flag` used to delete items

**Active bindings**

- `.decl` associated declaration
- `.e` value to be assigned
- `.delete_flag` used to delete items
AssignItem

Methods

Public methods:

• AssignItem$new()  
• AssignItem$id()  
• AssignItem$getValue()  
• AssignItem$setValue()  
• AssignItem$getDecl()  
• AssignItem$setDecl()  
• AssignItem$c_str()  
• AssignItem$getDeleteFlag()  
• AssignItem$delete()  
• AssignItem$clone()

Method new(): constructor

Usage:
AssignItem$new(decl, value)

Arguments:
decl declaration associated with assignment.
value expression to be assigned.

Method id(): get the name of assigned variable

Usage:
AssignItem$id()

Method getValue(): get the value

Usage:
AssignItem$getValue()

Method setValue(): set the value

Usage:
AssignItem$setValue(val)

Arguments:
val value/expression to be set

Method getDecl(): get the associated declaration

Usage:
AssignItem$getDecl()

Method setDecl(): set the associated declaration

Usage:
AssignItem$setDecl(decl)

Arguments:
decl declaration to be set
Method `c_str()`: get the MiniZinc representation

Usage:
AssignItem$c_str()

Method `getDeleteFlag()`: delete flag for internal use

Usage:
AssignItem$getDeleteFlag()

Method `delete()`: delete the assignment item

Usage:
AssignItem$delete()

Method `clone()`: The objects of this class are cloneable with this method.

Usage:
AssignItem$clone(deep = FALSE)

Arguments:
- deep Whether to make a deep clone.

---

**Assignment**

**Assignment problem 2**

**Description**

Solve an assignment problem (Goal is to minimize the cost)

**Usage**

`assignment(n, m, cost)`

**Arguments**

- `n`: number of agents
- `m`: number of tasks
- `cost`: m x n 2D array where each row corresponds to the cost of each task for that agent. (to be provided as 1-D vector)
Description

Solve an assignment problem Winston "Operations Research", page 398, swimming team example
Model created by Hakan Kjellerstrand(hakank(at)bonetmail.com) See : http://www.hakank.org/minizinc/assignment2.mzn

Usage

assignment_2(rows, cols, cost)

Arguments

rows number of columns
cols number of tasks
cost cost matrix (to be provided as 1-D vector)

BinOp

Description


Super class

rminizinc::Expression -> BinOp

Public fields

.1hs_exp the left hand side expression
.rhs_exp the right hand side expression
.op the operator
.delete_flag used to delete items

Active bindings

.1hs_exp the left hand side expression
.rhs_exp the right hand side expression
.op the operator
.delete_flag used to delete items
Methods

Public methods:

- BinOp$new()
- BinOp$getLhs()
- BinOp$getRhs()
- BinOp$getOp()
- BinOp$setOp()
- BinOp$setLhs()
- BinOp$setRhs()
- BinOp$c_str()
- BinOp$getDeleteFlag()
- BinOp$delete()
- BinOp$clone()

Method new(): constructor

Usage:
BinOp$new(lhs, binop, rhs)

Arguments:
lhs the left hand side expression
binop the binary operator to be used
rhs the right hand side expression

Method getLhs(): get the lhs expression

Usage:
BinOp$getLhs()

Method getRhs(): get the rhs expression

Usage:
BinOp$getRhs()

Method getOp(): get the operator

Usage:
BinOp$getOp()

Method setOp(): set the operator

Usage:
BinOp$setOp(binop)

Arguments:
op binary operator to be set

Method setLhs(): set the lhs expression

Usage:
BinOp$setLhs(e)
Arguments:
    e  expression to set

Method setRhs(): set the rhs expression
    Usage:
    BinOp$setRhs(e)
    Arguments:
    e  expression to set

Method c_str(): return the MiniZinc representation
    Usage:
    BinOp$c_str()

Method getDeleteFlag(): delete flag for internal use
    Usage:
    BinOp$getDeleteFlag()

Method delete(): delete the assignment item
    Usage:
    BinOp$delete()

Method clone(): The objects of this class are cloneable with this method.
    Usage:
    BinOp$clone(deep = FALSE)
    Arguments:
    deep  Whether to make a deep clone.

Examples
    newBinOp = BinOp$new(lhs = Int$new(2), binop = "+", rhs = Int$new(5))
    newBinOp$c_str()
    newBinOp$setLhs(Int$new(5))
    newBinOp$setOp("-")
    newBinOp$setRhs(Int$new(2))
    newBinOp$c_str()
Description
Create a bool in MiniZinc

Super class
`rminizinc::Expression` -> `Bool`

Public fields
```
.value value
```

Active bindings
```
.value value
```

Methods

Public methods:
- `Bool$new()`
- `Bool$v()`
- `Bool$c_str()`
- `Bool$clone()`

Method `new()`: constructor

Usage:
```c
Bool$new(val)
```
Arguments:
```
val boolean input
```

Method `v()`: get boolean value

Usage:
```c
Bool$v()
```

Method `c_str()`: get the MiniZinc representation

Usage:
```c
Bool$c_str()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:
```c
Bool$clone(deep = FALSE)
```
Arguments:
```
deep Whether to make a deep clone.
```
Examples

```python
newBool = Bool$new(TRUE)
newBool$c_str()
```

---

**BoolArrDecl**

*n-D bool array declaration*

**Description**

Declare a new n-dimensional array of bools

**Usage**

```python
BoolArrDecl(name, kind, ind, value = NULL, ndim)
```

**Arguments**

- **name**: variable/parameter name
- **kind**: "var" or "par"
- **ind**: index of the array
- **value**: value (NULL by default)
- **ndim**: number of dimensions of the array

---

**BoolDecl**

*new bool declaration*

**Description**

Declare a new bool

**Usage**

```python
BoolDecl(name, kind, value = NULL)
```

**Arguments**

- **name**: variable/parameter name
- **kind**: "var" or "par"
- **value**: provide TRUE or FALSE (NULL by default)
boolExpressions  \text{get bools}

\textbf{Description}

Get a list of bool expressions

\textbf{Usage}

\texttt{boolExpressions(vals)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{vals} \quad \text{vector of bool values}
\end{itemize}

\textbf{BoolSetDecl} \quad \textit{set of bool declaration}

\textbf{Description}

Declare a new set of bool

\textbf{Usage}

\texttt{BoolSetDecl(name, kind, value = NULL)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{name} \quad \text{variable/parameter name}
  \item \texttt{kind} \quad "var" or "par"
  \item \texttt{value} \quad \text{provide a Set object (or NULL)}
\end{itemize}
Description
Create function calls in MiniZinc

Super class
rminizinc::Expression -> Call

Public fields
.id the function id
.lExp list of expressions
.delete_flag used to delete items

Active bindings
.id the function id
.lExp list of expressions
.delete_flag used to delete items

Methods
Public methods:
• Call$new()
• Call$getName()
• Call$setName()
• Call$nargs()
• Call$getArgs()
• Call$setArgs()
• Call$getArg()
• Call$setArg()
• Call$c_str()
• Call$getDeleteFlag()
• Call$delete()
• Call$clone()

Method new(): constructor
Usage:
Call$new(fnName, args)
Arguments:
fnName  function name
args  the list of expressions

Method getName(): get the function id/string
   Usage:
   Call$getName()

Method setName(): get the function id/string
   Usage:
   Call$setName(name)
   Arguments:
   name  new function name

Method nargs(): get the number of arguments
   Usage:
   Call$nargs()

Method getArgs(): get the expression list
   Usage:
   Call$getArgs()

Method setArgs(): set the expression list
   Usage:
   Call$setArgs(args)
   Arguments:
   args  list of expressions to be set

Method getArg(): get the expression based on index
   Usage:
   Call$getArg(i)
   Arguments:
   i  index

Method setArg(): set argument i
   Usage:
   Call$setArg(e, i)
   Arguments:
   e  expression
   i  index

Method c_str(): return the MiniZinc representation
   Usage:
   Call$c_str()
Method getDeleteFlag(): delete flag for internal use
   Usage:
   Call$getDeleteFlag()

Method delete(): delete the assignment item
   Usage:
   Call$delete()

Method clone(): The objects of this class are cloneable with this method.
   Usage:
   Call$clone(deep = FALSE)
   Arguments:
   deep Whether to make a deep clone.

Examples
   newCall = Call$new(fnName = "sum", args = list(Int$new(2), Int$new(5)))
   newCall$c_str()

Description
   Create a Comprehension in MiniZinc

Super class
   rminizinc::Expression -> Comprehension

Public fields
   .generators a vector of generators
   .expression the comprehension expression
   .set TRUE if comprehension is a set
   .delete_flag used to delete items

Active bindings
   .generators a vector of generators
   .expression the comprehension expression
   .set TRUE if comprehension is a set
   .delete_flag used to delete items
Methods

Public methods:

- Comprehension$new()
- Comprehension$ngens()
- Comprehension$getGens()
- Comprehension$setGens()
- Comprehension$getGen()
- Comprehension$setGen()
- Comprehension$getBody()
- Comprehension$setBody()
- Comprehension$isSet()
- Comprehension$c_str()
- Comprehension$getDeleteFlag()
- Comprehension$delete()
- Comprehension$clone()

Method new(): constructor

Usage:
Comprehension$new(generators, body, set)

Arguments:
generators generators of the expression
body body/expression of the comprehension
set bool to specify if comprehension is a set.

Method ngens(): get the number of generators

Usage:
Comprehension$ngens()

Method getGens(): get all the generator expressions

Usage:
Comprehension$getGens()

Method setGens(): set all the generator expressions

Usage:
Comprehension$setGens(generators)

Arguments:
generators list of generator expressions to be set

Method getGen(): get the ith generator expression

Usage:
Comprehension$getGen(i)

Arguments:
Comprehension

i index

**Method** `setGen()`: set the ith generator expression

*Usage:*
Comprehension$setGen(i, expGen)

*Arguments:*
i index
expGen generator expression to be set

**Method** `getBody()`: get the expression/body

*Usage:*
Comprehension$getBody()

**Method** `setBody()`: set the expression/body

*Usage:*
Comprehension$setBody(e)

*Arguments:*
e new expression value

**Method** `isSet()`: check if comprehension is a set

*Usage:*
Comprehension$isSet()

**Method** `c_str()`: get the MiniZinc representation

*Usage:*
Comprehension$c_str()

**Method** `getDeleteFlag()`: delete flag for internal use

*Usage:*
Comprehension$getDeleteFlag()

**Method** `delete()`: delete the assignment item

*Usage:*
Comprehension$delete()

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*
Comprehension$clone(deep = FALSE)

*Arguments:*
deep Whether to make a deep clone.
Description

Describe Minizinc constraints on decision variables.

Super class

\texttt{rminizinc::Item} -> ConstraintItem

Public fields

\begin{itemize}
\item \texttt{.e} the constraint expression
\item \texttt{.delete\_flag} used to delete items
\end{itemize}

Active bindings

\begin{itemize}
\item \texttt{.e} the constraint expression
\item \texttt{.delete\_flag} used to delete items
\end{itemize}

Methods

Public methods:

\begin{itemize}
\item \texttt{ConstraintItem$new()}
\item \texttt{ConstraintItem$getExp()}
\item \texttt{ConstraintItem$setExp()}
\item \texttt{ConstraintItem$c\_str()}
\item \texttt{ConstraintItem$getDeleteFlag()}
\item \texttt{ConstraintItem$delete()}
\item \texttt{ConstraintItem$clone()}
\end{itemize}

Method \texttt{new()}: Creates a new instance of Constraint class.

Usage:
\texttt{ConstraintItem$new(e = \text{NULL}, mzn\_str = \text{NULL})}

Arguments:
\begin{itemize}
\item \texttt{e} The expression for the constraint (used if \texttt{e} is \text{NULL})
\item \texttt{mzn\_str} string representation of Constraint item
\end{itemize}

Method \texttt{getExp()}: get the constraint expression

Usage:
\texttt{ConstraintItem$getExp()}

Method \texttt{setExp()}: set the constraint expression
**Expression**

*Usage:*
ConstraintItem$setExp(e)

*Arguments:*
e  expression

**Method c_str():** serialize to MiniZinc syntax

*Usage:*
ConstraintItem$c_str()

**Method getDeleteFlag():** delete flag for internal use

*Usage:*
ConstraintItem$getDeleteFlag()

**Method delete():** delete the constraint item

*Usage:*
ConstraintItem$delete()

**Method clone():** The objects of this class are cloneable with this method.

*Usage:*
ConstraintItem$clone(deep = FALSE)

*Arguments:*
depth  Whether to make a deep clone.

---

**Expression**  
*Expression (Abstract class – should not be initialized)*

**Description**

This class represents an expression in MiniZinc.

**Methods**

**Public methods:**

- **Expression$new()**
- **Expression$clone()**

**Method new():** constructor

*Usage:*
Expression$new()

**Method clone():** The objects of this class are cloneable with this method.

*Usage:*
Expression$clone(deep = FALSE)

*Arguments:*
depth  Whether to make a deep clone.
**expressionDelete**  
*delete an expression*

**Description**
Delete the object everywhere from the MiniZinc model

**Usage**
expressionDelete(classNm, model)

**Arguments**
classNm  
class of the object to delete
model  
model to delete the object from

---

**Float**  
*Float*

**Description**
Create a float in MiniZinc

**Super class**
*rmminizinc::Expression -> Float*

**Public fields**
.value  object of class expression

**Active bindings**
.value  object of class expression

**Methods**

**Public methods:**
- Float$new()  
- Float$getFloatVal()  
- Float$setFloatVal()  
- Float$c_str()  
- Float$clone()

**Method** new(): constructor
FloatArrDecl

Usage:
Float$new(val)

Arguments:
val  the float value

Method getFloatVal(): get the float value

Usage:
Float$getFloatVal()

Method setFloatVal(): set the float value

Usage:
Float$setFloatVal(val)

Arguments:
val  value to be set

Method c_str(): get the MiniZinc representation

Usage:
Float$c_str()

Method clone(): The objects of this class are cloneable with this method.

Usage:
Float$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

Examples
newFloat = Float$new(1.5)
newFloat$c_str()
newFloat$setFloatVal(newFloat$getFloatVal() + 2.5)
newFloat$c_str()

FloatArrDecl  n-D float array declaration

Description
Declare a new n-dimensional array of float

Usage
FloatArrDecl(name, kind, ind, value = NULL, ndim)
Arguments

name       variable/parameter name
kind       "var" or "par"
ind        index of the array
value      value (NULL by default)
ndim       number of dimensions of the array

---

**FloatDecl**  
<float declaration>

**Description**
Declare a new float

**Usage**

```
FloatDecl(name, kind, value = NULL, domain = NULL)
```

**Arguments**

name       variable/parameter name
kind       "var" or "par"
value      pass a numeric/double value in R (NULL by default)
domain     domain of the float variable (NULL by default)

---

**floatExpressions**  
<get floats>

**Description**
Get a list of floats expressions

**Usage**

```
floatExpressions(vals)
```

**Arguments**

vals       vector of floats values
**FloatSetDecl**

### Description
Declare a new set of float.

### Usage
```c
FloatSetDecl(name, kind, value = NULL)
```

### Arguments
- **name**: variable/parameter name
- **kind**: "var" or "par"
- **value**: provide an FloatSetVal object (or NULL)

**FloatSetVal**

### Description
Float set range in MiniZinc.

### Public fields
- `.min` minimum FloatVal
- `.max` maximum FloatVal

### Active bindings
- `.min` minimum FloatVal
- `.max` maximum FloatVal

### Methods
- **Public methods:**
  - `FloatSetVal$new()`
  - `FloatSetVal$getMin()`
  - `FloatSetVal$setMin()`
  - `FloatSetVal$getMax()`
  - `FloatSetVal$setMax()`
  - `FloatSetVal$clone()`
**Method** `new()`: constructor

*Usage:*

`FloatSetVal$new(fmin, fmax)`

*Arguments:*

- `fmin` the minimum `FloatVal`
- `fmax` the maximum `FloatVal`

**Method** `getMin()`: get the minimum float value

*Usage:*

`FloatSetVal$getMin()`

**Method** `setMin()`: set the minimum float value

*Usage:*

`FloatSetVal$setMin(val)`

*Arguments:*

- `val` float value to be set

**Method** `getMax()`: get the maximum float value

*Usage:*

`FloatSetVal$getMax()`

**Method** `setMax()`: set the maximum float value

*Usage:*

`FloatSetVal$setMax(val)`

*Arguments:*

- `val` float value to be set

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`FloatSetVal$clone(deep = FALSE)`

*Arguments:*

- `deep` Whether to make a deep clone.
FloatVal

FloatVal class (not exposed to user)

Description

create a Float Value in MiniZinc

Public fields

.val the integer value

Active bindings

.val the integer value

Methods

Public methods:

• FloatVal$new()
• FloatVal$v()
• FloatVal$clone()

Method new(): constructor

Usage:
FloatVal$new(val)

Arguments:
val float value to be assigned

Method v(): return the value

Usage:
FloatVal$v()

Method clone(): The objects of this class are cloneable with this method.

Usage:
FloatVal$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.
FunctionItem

**Description**

Create Independent functions (that are not part of any other items) in a MiniZinc model

**Super class**

    rminizinc::Item -> FunctionItem

**Public fields**

- `.id` name of the function
- `.e` expression in the function
- `.decls` parameter declarations
- `.ann` annotation
- `.ti` return type of the function
- `.delete_flag` used to delete items

**Active bindings**

- `.id` name of the function
- `.e` expression in the function
- `.decls` parameter declarations
- `.ann` annotation
- `.ti` return type of the function
- `.delete_flag` used to delete items

**Methods**

**Public methods:**

- `FunctionItem$new()`
- `FunctionItem$name()`
- `FunctionItem$getDecls()`
- `FunctionItem$getBody()`
- `FunctionItem$getAnn()`
- `FunctionItem$setDecls()`
- `FunctionItem$setBody()`
- `FunctionItem$setAnn()`
- `FunctionItem$rtype()`
- `FunctionItem$c_str()`
- `FunctionItem$getDeleteFlag()`
• `FunctionItem$delete()`
• `FunctionItem$clone()`

**Method** `new()`: constructor

*Usage:*
```
FunctionItem$new(
    name = NULL,
    decls = NULL,
    rt = NULL,
    ann = NULL,
    body = NULL,
    mzn_str = NULL
)
```

*Arguments:*
- `name` name of the function
- `decls` variable declarations
- `rt` the return type ("bool par", "bool var" or other)
- `ann` annotation
- `body` body of the function
- `mzn_str` string representation of Function Item

**Method** `name()`: get the name of the function

*Usage:*
```
FunctionItem$name()
```

**Method** `getDecls()`: get the list of declarations

*Usage:*
```
FunctionItem$getDecls()
```

**Method** `getBody()`: get the function body

*Usage:*
```
FunctionItem$getBody()
```

**Method** `getAnn()`: get the function annotation

*Usage:*
```
FunctionItem$getAnn()
```

**Method** `setDecls()`: set the list of declarations

*Usage:*
```
FunctionItem$setDecls(decls)
```

*Arguments:*
- `decls` list of declarations to be set

**Method** `setBody()`: set the function body

*Usage:*

FunctionItem$setBody()
Arguments:
body  function expression to set or NULL

**Method** setAnn(): set the function annotation
Usage:
FunctionItem$setAnn()
Arguments:
ann  annotation to be set or NULL

**Method** rtype(): get if the function is a test, predicate or a function call itself.
Usage:
FunctionItem$rtype()

**Method** c_str(): get the MiniZinc representation
Usage:
FunctionItem$c_str()

**Method** getDeleteFlag(): delete flag for internal use
Usage:
FunctionItem$getDeleteFlag()

**Method** delete(): delete the variable item
Usage:
FunctionItem$delete()

**Method** clone(): The objects of this class are cloneable with this method.
Usage:
FunctionItem$clone(deep = FALSE)
Arguments:
deep  Whether to make a deep clone.

<table>
<thead>
<tr>
<th>Generator</th>
<th>Generator</th>
</tr>
</thead>
</table>

**Description**
Create a generator in MiniZinc

**Super class**
$rminizinc::Expression$ -> Generator
Public fields

.decls variable declarations
.in in expression
.where where expression
.delete_flag used to delete items

Active bindings

.decls variable declarations
.in in expression
.where where expression
.delete_flag used to delete items

Methods

Public methods:

• Generator$new()
• Generator$getIn()
• Generator$setIn()
• Generator$getWhere()
• Generator$setWhere()
• Generator$getDecl()
• Generator$setDecl()
• Generator$c_str()
• Generator$getDeleteFlag()
• Generator$delete()
• Generator$clone()

Method new(): constructor

Usage:
Generator$new(decls, IN = NULL, where = NULL)

Arguments:
decls list of variable declarations
IN the in expression of generator
where the where expression of generator

Method getIn(): get the in expression

Usage:
Generator$getIn()

Method setIn(): set the in expression

Usage:
Generator$setIn(expIn)
Arguments:
expIn  expression to be set

Method getWhere(): get the where expression
Usage:
Generator$getWhere()

Method setWhere(): get the where expression
Usage:
Generator$setWhere(expWhere)
Arguments:
expWhere  where expression (or NULL)

Method getDecl(): get the ith declaration
Usage:
Generator$getDecl(i)
Arguments:
i  index

Method setDecl(): get the ith declaration
Usage:
Generator$setDecl(i, decl)
Arguments:
i  index
decl  declaration to be set

Method c_str(): get the MiniZinc representation
Usage:
Generator$c_str()

Method getDeleteFlag(): delete flag for internal use
Usage:
Generator$getDeleteFlag()

Method delete(): delete the assignment item
Usage:
Generator$delete()

Method clone(): The objects of this class are cloneable with this method.
Usage:
Generator$clone(deep = FALSE)
Arguments:
deep  Whether to make a deep clone.

Examples
newGen = Generator$new(IN = IntSetDecl(name = "SET", kind = "par"),
decs = list(IntDecl(name = "i", kind = "par")))
**getRModel**

init all classes

**Description**

Given the return value of `mzn_parse()`, it creates a model in R using the API mirror.

**Usage**

getRModel(mznParseList)

**Arguments**

- `mznParseList` list input

---

**getType**

initialized type (not exposed to user)

**Description**

Helper function to initialise the type.

**Usage**

getType(type_str, kind)

**Arguments**

- `type_str` type string returned by `parse_mzn()`.
- `kind` par or var

---

**get_missing_pars**

get missing parameters

**Description**

Get the values of the missing parameters

**Usage**

get_missing_pars(model)

**Arguments**

- `model` object of Model class
### helperDeleteExpression

**helper delete expression**

**Description**

Helper function to search the through a model for an expression and return the object if found.

**Usage**

`helperDeleteExpression(classNm)`

**Arguments**

- `classNm` name of the object class

---

### helperDeleteItem

**helper delete item**

**Description**

Helper function to search the through a model for an item and return the object if found.

**Usage**

`helperDeleteItem(classNm)`

**Arguments**

- `classNm` name of the object class

---

### Id

**Id class (not exposed to the user)**

**Description**

Create a new Id in MiniZinc.

**Super class**

```
rmminizinc::Expression -> Id
```
Public fields

.id the string identifier
.delete_flag used to delete items

Active bindings

.id the string identifier
.delete_flag used to delete items

Methods

Public methods:

• Id$new()
• Id$getName()
• Id$setName()
• Id$c_str()
• Id$getDeleteFlag()
• Id$delete()
• Id$clone()

Method new(): constructor

Usage:
Id$new(id)

Arguments:
id id to be created

Method getName(): get the string identifier

Usage:
Id$getName()

Method setName(): set the string identifier

Usage:
Id$setName(name)

Arguments:
name string name to set

Method c_str(): return the MiniZinc representation

Usage:
Id$c_str()

Method getDeleteFlag(): delete flag for internal use

Usage:
Id$getDeleteFlag()
Method delete(): delete the assignment item

Usage:
Id$delete()

Method clone(): The objects of this class are cloneable with this method.

Usage:
Id$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

---

IncludeItem Include Items

Description
Include external mzn files in your model.

Super class
rminizinc::Item -> IncludeItem

Public fields
.id name of mzn file
.delete_flag used to delete items

Active bindings
.id name of mzn file
.delete_flag used to delete items

Methods
Public methods:
- IncludeItem$new()
- IncludeItem$getmznName()
- IncludeItem$setmznName()
- IncludeItem$c_str()
- IncludeItem$getDeleteFlag()
- IncludeItem$delete()
- IncludeItem$clone()

Method new(): constructor

Usage:
IncludeItem$new(name = NULL, mzn_str = NULL)

Arguments:
name name of the file to include
mzn_str string representation of Include Item get file name set the file name

Method getmznName():
Usage:
IncludeItem$getmznName()

Method setmznName(name):
Usage:
IncludeItem$setmznName(name)
Arguments:
name name of file

Method c_str(): get the MiniZinc representation
Usage:
IncludeItem$c_str()

Method getDeleteFlag(): delete flag for internal use
Usage:
IncludeItem$getDeleteFlag()

Method delete(): delete the include item
Usage:
IncludeItem$delete()

Method clone(): The objects of this class are cloneable with this method.
Usage:
IncludeItem$clone(deep = FALSE)
Arguments:
deep Whether to make a deep clone.

Description
Recursive helper function for initilizing expression classes

Usage
initExpression(pList)

Arguments
pList list from mzn_parse to initialise objects
**initItem**

initialize R6 from parsed (not to be exposed)

---

**Description**

Initialize all the R6 objects using the list returned by `mzn_parse()` to create exactly the same structure in R.

**Usage**

```
initItem(parsedList)
```

**Arguments**

- `parsedList`: list returned by `mzn_parse()`

---

**Int**

**Description**

Create an integer in MiniZinc

**Super class**

```
 rminizinc::Expression -> Int
```

**Public fields**

- `.value`: object of class expression

**Active bindings**

- `.value`: object of class expression

**Methods**

**Public methods:**

- `Int$new()`
- `Int$getIntVal()`
- `Int$setIntVal()`
- `Int$c_str()`
- `Int$clone()`

**Method** `new()`: constructor
**Usage**

Int$new(val)

**Arguments**:

val the value of the integer

**Method** getIntVal(): get the IntVal value

**Usage**:

Int$getIntVal()

**Method** setIntVal(): set the IntVal value

**Usage**:

Int$setIntVal(val)

**Arguments**:

val value to be set

**Method** c_str(): get the MiniZinc representation

**Usage**:

Int$c_str()

**Method** clone(): The objects of this class are cloneable with this method.

**Usage**:

Int$clone(deep = FALSE)

**Arguments**:

deep Whether to make a deep clone.

**Examples**

```plaintext
newInt = Int$new(10)
newInt$c_str()
newInt$setIntVal(newInt$getIntVal() + 20)
newInt$c_str()
```

---

**IntArrDecl**

*n-D int array declaration*

**Description**

Declare a new n-dimensional array of int

**Usage**

IntArrDecl(name, kind, ind, value = NULL, ndim)
**Arguments**

- **name**: variable/parameter name
- **kind**: "var" or "par"
- **ind**: index of the array
- **value**: Array Object (NULL by default)
- **ndim**: number of dimensions of the array

---

**IntDecl**

*int declaration*

---

**Description**

Declare a new int

**Usage**

IntDecl(name, kind, value = NULL, domain = NULL)

**Arguments**

- **name**: variable/parameter name
- **kind**: "var" or "par"
- **value**: pass a numeric/integer value in R (NULL by default)
- **domain**: domain of the int variable (NULL by default)

---

**intExpressions**

*get ints*

---

**Description**

Get a list of integer expressions

**Usage**

intExpressions(vals)

**Arguments**

- **vals**: vector of integer values
**IntSetDecl**

**int set declaration**

**Description**

Declare a new set of int

**Usage**

\[
\text{IntSetDecl(name, kind, value = NULL)}
\]

**Arguments**

- **name**: variable/parameter name
- **kind**: "var" or "par"
- **value**: provide an IntSetVal object (NULL by default)

**IntSetVal**

**Integer set value**

**Description**

integer range set value in MiniZinc

**Public fields**

- `.min` minimum value of integer range
- `.max` maximum value of integer range

**Active bindings**

- `.min` minimum value of integer range
- `.max` maximum value of integer range

**Methods**

**Public methods:**

- `IntSetVal$new()`
- `IntSetVal$getMin()`
- `IntSetVal$setMin()`
- `IntSetVal$getMax()`
- `IntSetVal$setMax()`
- `IntSetVal$clone()`
Method new(): constructor

Usage:
IntSetVal$new(imin, imax)

Arguments:
imin minimum int value
imax maximum int value

Method getMin(): get the minimum IntVal

Usage:
IntSetVal$getMin()

Method setMin(): set the minimum IntVal

Usage:
IntSetVal$setMin(val)

Arguments:
val int value to be set

Method getMax(): get the maximum IntVal

Usage:
IntSetVal$getMax()

Method setMax(): set the maximum IntVal

Usage:
IntSetVal$setMax(val)

Arguments:
val int value to be set

Method clone(): The objects of this class are cloneable with this method.

Usage:
IntSetVal$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.
**IntVal**

*IntVal class (not exposed to user)*

---

**Description**

create an Integer Value in MiniZinc

**Public fields**

`.val` the integer value

**Active bindings**

`.val` the integer value

**Methods**

**Public methods:**

- `IntVal$new()`
- `IntVal$v()`
- `IntVal$clone()`

**Method** `new()`: constructor

*Usage:*

`IntVal$new(val)`

*Arguments:*

val int value to be assigned

**Method** `v()`: return the value

*Usage:*

`IntVal$v()`

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`IntVal$clone(deep = FALSE)`

*Arguments:*

deep Whether to make a deep clone.
Description
Create if-then-else expressions in MiniZinc

Super class
rminizinc::Expression -> Ite

Public fields
- .ifs list of if expressions
- .thens list of corresponding then expressions
- .else else expression
- .delete_flag used to delete items

Active bindings
- .ifs list of if expressions
- .thens list of corresponding then expressions
- .else else expression
- .delete_flag used to delete items

Methods
Public methods:
- Ite$new()
- Ite$getIfs()
- Ite$getThens()
- Ite$setIfsThens()
- Ite$getIf()
- Ite$setIf()
- Ite$getThen()
- Ite$setThen()
- Ite$getElse()
- Ite$setElse()
- Ite$c_str()
- Ite$getDeleteFlag()
- Ite.delete()
- Ite$clone()

Method new(): constructor
Usage:
Ite$new(ifs, thens, Else)

Arguments:
ifs  list of if expressions
thens list of corresponding then expressions
Else  else expression

Method getIfs(): get the if expression list
Usage:
Ite$getIfs()

Method getThens(): get the then expression list
Usage:
Ite$getThens()

Method setIfsThens(): set the if and then expression list
Usage:
Ite$setIfsThens(ifs, thens)

Arguments:
ifs  expression list to be set
thens expression list to be set

Method getIf(): get the ith if expression
Usage:
Ite$getIf(i)

Arguments:
i  index

Method setIf(): set the ith if expression
Usage:
Ite$setIf(i, expIf)

Arguments:
i  index
expIf if expression to be set

Method getThen(): get the ith then expression
Usage:
Ite$getThen(i)

Arguments:
i  index

Method setThen(): set the ith then expression
Usage:
ITE$setThen(i, expThen)

*Arguments:*
i index
expThen then expression to be set

**Method** `getElse()`: get the else expression

*Usage:*
ITE$getElse()

**Method** `setElse()`: get the else expression

*Usage:*
ITE$setElse(expElse)

*Arguments:*
expElse else expression to be set

**Method** `c_str()`: get the MiniZinc representation

*Usage:*
ITE$c_str()

**Method** `getDeleteFlag()`: delete flag for internal use

*Usage:*
ITE$getDeleteFlag()

**Method** `delete()`: delete the assignment item

*Usage:*
ITE$delete()

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*
ITE$clone(deep = FALSE)

*Arguments:*
depth Whether to make a deep clone.
**Item**

**Item class (Abstract)**

**Description**

Abstract class for all items in MiniZinc grammar

**Methods**

**Public methods:**

- `Item$new()`
- `Item$clone()`

**Method** `new()`: constructor

*Usage:*

`Item$new()`

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`Item$clone(deep = FALSE)`

*Arguments:*

`deep` Whether to make a deep clone.

**itemDelete**

**search item in model and delete**

**Description**

Find the object in the model and delete it.

**Usage**

`itemDelete(classNm, model)`

**Arguments**

- `classNm` object to be deleted
- `model` model to delete the object from
iterExpression  
iterate through expressions and delete (Under Development)

Description

Given an object to delete and expression object, delete all the embedded expression objects that are identical.

Usage

iterExpression(classNm, expObj)

Arguments

classNm class name of the object to delete
expObj expression object to iterate through

iterItem  
check all possible items (Under Development)

Description

Find the expressions in the items and delete them if matched.

Usage

iterItem(mod, classNm)

Arguments

mod model to be searched
classNm class name of the object to be deleted
Description

Solve a simple knapsack problem (Goal is to maximize the profit)

Usage

knapsack(n, capacity, profit, size)

Arguments

n                      number of items
capacity               total capacity of carrying weight
profit                 profit corresponding to each item
size                   weight/size of each item

Let

Description

Create let expression in MiniZinc

Super class

rminizinc::Expression -> Let

Public fields

.decl list of local declarations
.in  body of the let
.delete_flag used to delete items

Active bindings

.decl list of local declarations
.in  body of the let
.delete_flag used to delete items
Methods

Public methods:

• Let$new()
• Let$getLets()
• Let$setLets()
• Let$getLet()
• Let$setLet()
• Let$getBody()
• Let$setBody()
• Let$c_str()
• Let$getDeleteFlag()
• Let$delete()
• Let$clone()

Method new(): constructor

Usage:
Let$new(let, body)

Arguments:
let list of local declaration items and/or constraint items
body body of the let

Method getLets(): access list of declaration items and/or constraint items

Usage:
Let$getLets()

Method setLets(): set list of declaration items and/or constraint items

Usage:
Let$setLets(letList)

Arguments:
letList list of declaration items and/or constraint items to be set

Method getLet(): access declaration item and/or constraint item i

Usage:
Let$getLet(i)

Arguments:
i index of let declaration item and/or constraint item to be accessed

Method setLet(): set list of declaration item and/or constraint item i

Usage:
Let$setLet(let)

Arguments:
let declaration item and/or constraint item to be set
Method `getBody()`: get the body

Usage:
Let $getBody()$

Method `setBody()`: set the body

Usage:
Let $setBody(expBody)$

Arguments:
expBody expression to be set for body

Method `c_str()`: get the MiniZinc representation

Usage:
Let $c_str()$

Method `getDeleteFlag()`: delete flag for internal use

Usage:
Let $getDeleteFlag()$

Method `delete()`: delete the assignment item

Usage:
Let $delete()$

Method `clone()`: The objects of this class are cloneable with this method.

Usage:
Let $clone(deep = FALSE)$

Arguments:
deep Whether to make a deep clone.

---

**LIBMINIZINC_PATH**

*Absolute path of the libminizinc library*

**Description**

Absolute path of the libminizinc library

**Usage**

LIBMINIZINC_PATH

**Format**

A string containing linker flag
**Description**

Solve a magic series problem in MiniZinc Model created by Hakan Kjellerstrand (hakank(at)bonetmail.com)
See: http://www.hakank.org/minizinc/magic_series.mzn

**Usage**

```
magic_series(n)
```

**Arguments**

- `n` order of magic square

---

**Description**

Solve a magic squares problem in MiniZinc Model created by Hakan Kjellerstrand (hakank(at)bonetmail.com)
See: http://www.hakank.org/minizinc/magic_square.mzn

**Usage**

```
magic_square(n)
```

**Arguments**

- `n` order of magic square
**Model**

*MiniZinc Model class*

---

**Description**

This class will take all the objects required to create a MiniZinc model.

**Public fields**

- .items list of items in the model

**Active bindings**

- .items list of items in the model

**Methods**

**Public methods:**

- `Model$new()`  
- `Model$getItems()`  
- `Model$setItems()`  
- `Model$getItem()`  
- `Model$setItem()`  
- `Model$addItem()`  
- `Model$nitems()`  
- `Model$mzn_string()`  
- `Model$clone()`

**Method** `new()`: create a new instance of model class

*Usage:*

`Model$new(items)`

*Arguments:*

- `items` all items of the model

**Method** `getItems()`: get all the items

*Usage:*

`Model$getItems()`

**Method** `setItems()`: set all the items

*Usage:*

`Model$setItems(items)`

*Arguments:*

- `items` items to be set
Method `getItem()`: get the item using index

Usage:
Model$getItem(i)

Arguments:
i index

Method `setItem()`: set the item using index

Usage:
Model$setItem(i, item)

Arguments:
i index
item item to be set

Method `addItem()`: add item to the model

Usage:
Model$addItem(item)

Arguments:
item item to add

Method `nitems()`: get the number of items

Usage:
Model$nitems()

Method `mzn_string()`: get the string representation of the model

Usage:
Model$mzn_string()

Method `clone()`: The objects of this class are cloneable with this method.

Usage:
Model$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.
Description

evaluates the MiniZinc model

Usage

mzn_eval(
  lib_path = "", 
  r_model = NULL, 
  mzn_path = "", 
  model_string = "", 
  solver = "org.gecode.gecode", 
  dzn_path = "", 
  all_solutions = TRUE, 
  time_limit = 300000L, 
  other_cl_options = NULL
)

Arguments

lib_path the path of the library where the standard library files are present (the parent directory of the std directory).
r_model R6 Model object
mzn_path path of the mzn file to be solved
model_string model string to be solved.
solver the name of the solver to use.(default: Gecode)
dzn_path path of the datafile to be used.
all_solutions bool to specify if all solutions are specified.(default: true)
time_limit stop after <time_limit> milliseconds. (default: 300000ms – 5 mins)
other_cl_options other command line options/flags that you want to provide 1. Please provide as a character/string vector with each element as a flag 2. Incorrect flags or incorrect commands will throw errors. 3. Changing the default solution output options will result in parsing errors and the solutions will not be parsed correctly to R but the solution string will be returned.
production_planning

mzn_parse

MiniZinc syntax parser

Description

parses the MiniZinc syntax into R objects

Usage

mzn_parse(model_string = "", mzn_path = "", include_path = NULL)

Arguments

model_string string representation of the MiniZinc model.
mzn_path the path of model mzn.
include_path path of the included mzn in the model if it exists.

production_planning

production planning problem

Description

simple production planning problem taken from https://github.com/MiniZinc/minizinc-examples
Goal is to maximize the profit

Usage

production_planning(
  nproducts,
  profit,
  pnames,
  nresources,
  capacity,
  rnames,
  consumption
)

Arguments

nproducts number of different products
profit profit for each product (1-D vector)
pnames names of each product (1-D vector)
nresources number of resources
capacity amount of each resource available (1-D vector)
rnames: names of each resource (1-D vector)
consumption: units of each resource required to produce 1 unit of product (2-D vector to be provided as 1-D vector)

\[
\text{PROJECT\_DIRECTORY} \\
\text{Absolute path of project directory}
\]

**Description**

Absolute path of project directory

**Usage**

\[
\text{PROJECT\_DIRECTORY}
\]

**Format**

A string containing absolute path of the project directory

\[
\text{Set} \\
\text{Set}
\]

**Description**

Create a set in MiniZinc

**Super class**

\[
\text{rminizinc::Expression} \rightarrow \text{Set}
\]

**Public fields**

- .setVal: the value of the set
- .isv: the integer range set
- .fsv: the float range set
- .et: empty set
- .delete_flag: used to delete items

**Active bindings**

- .setVal: the value of the set
- .isv: the integer range set
- .fsv: the float range set
- .et: empty set
- .delete_flag: used to delete items
Methods

Public methods:

- Set$new()
- Set$getSetVec()
- Set$setSetVec()
- Set$isEmpty()
- Set$makeEmpty()
- Set$getIsv()
- Set$setIsv()
- Set$getFsv()
- Set$setFsv()
- Set$c_str()
- Set$getDeleteFlag()
- Set$delete()
- Set$clone()

Method new(): constructor

Usage:
Set$new(val = NULL, empty_set = FALSE)

Arguments:
val the set value
empty_set bool to specify is set is empty (FALSE by default)

Method getSetVec(): get the set expression

Usage:
Set$getSetVec()

Method setSetVec(): set the set expression

Usage:
Set$setSetVec(val)

Arguments:
val list of expressions

Method isEmpty(): is the set empty

Usage:
Set$isEmpty()

Method makeEmpty(): make the set empty

Usage:
Set$makeEmpty()

Method getIsv(): return the integer set range

Usage:
Set$getIsv()

**Method** setIsv(): set the integer set range

*Usage:*
Set$setIsv(val)

*Arguments:*

val integer set range

**Method** getFsv(): get the float set range

*Usage:*
Set$getFsv()

**Method** setFsv(): set the float set range

*Usage:*
Set$setFsv(val)

*Arguments:*

val float set range

**Method** c_str(): get the MiniZinc representation

*Usage:*
Set$c_str()

**Method** getDeleteFlag(): delete flag for internal use

*Usage:*
Set$getDeleteFlag()

**Method** delete(): delete the assignment item

*Usage:*
Set$delete()

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*
Set$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

**Examples**

newIntSet = Set$new(val = IntSetVal$new(1,5))
newIntSet$c_str()
newIntSet$setIsv(IntSetVal$new(2,6))
newIntSet$c_str()
newFloatSet = Set$new(val = FloatSetVal$new(1.1,5.1))
newFloatSet$c_str()
newFloatSet$setFsv(FloatSetVal$new(1.2,4.1))
**set_params**

*set missing parameters*

**Description**

Assign values to parameters which don’t have a value assigned yet.

**Usage**

```python
set_params(model, modData)
```

**Arguments**

- **model**
  - Model object
- **modData**
  - list of the value objects to be assigned

---

**SolveItem**

**Description**

specify whether the optimization problem is a satisfaction, minimization or maximization problem and/or expression to maximize/minimize and/or annotation

**Super class**

`rminizinc::Item` -> `SolveItem`

**Public fields**

- `.e` the expression to maximize or minimize
- `.st` the solve type
- `.ann` annotation of the solve type
- `.delete_flag` used to delete items

**Active bindings**

- `.e` the expression to maximize or minimize
- `.st` the solve type
- `.ann` annotation of the solve type
- `.delete_flag` used to delete items
Methods

Public methods:

- `SolveItem$new()`
- `SolveItem$getExp()`
- `SolveItem$getAnn()`
- `SolveItem$setExp()`
- `SolveItem$setAnn()`
- `SolveItem$getSt()`
- `SolveItem$setSt()`
- `SolveItem$c_str()`
- `SolveItem$getDeleteFlag()`
- `SolveItem$delete()`
- `SolveItem$clone()`

Method `new()`: create an instance of specify_problem class

Usage:

```r
SolveItem$new(solve_type = NULL, e = NULL, ann = NULL, mzn_str = NULL)
```

Arguments:

- `solve_type` satisfy, minimize or maximize
- `e` expression to minimize or maximize
- `ann` annotation
- `mzn_str` string representation of Solve Item

Method `getExp()`: get the expression (or NULL)

Usage:

```r
SolveItem$getExp()
```

Method `getAnn()`: get the annotation (or NULL)

Usage:

```r
SolveItem$getAnn()
```

Method `setExp()`: set the expression

Usage:

```r
SolveItem$setExp(e)
```

Arguments:

- `e` expression

Method `setAnn()`: set the annotation

Usage:

```r
SolveItem$setAnn(ann)
```

Arguments:

- `ann` annotation or Null
Method getSt(): get the solve type/objective

Usage:
SolveItem$getSt()

Method setSt(): set the solve type/objective

Usage:
SolveItem$setSt(objective)
Arguments:
objective solve type

Method c_str(): to string method

Usage:
SolveItem$c_str()

Method getDeleteFlag(): delete flag for internal use

Usage:
SolveItem$getDeleteFlag()

Method delete(): delete the variable item

Usage:
SolveItem$delete()

Method clone(): The objects of this class are cloneable with this method.

Usage:
SolveItem$clone(deep = FALSE)
Arguments:
depth Whether to make a deep clone.

---

**SOLVER_BIN**

Absolute path of the solver executable directory

Description
Absolute path of the solver executable directory

Usage
SOLVER_BIN

Format
A string containing path of solver executable directory
Description

can parse the JSON solution of a model to return a list output

Usage

sol_parse(solutionString)

Arguments

solutionString solution of the model as a string representation

Description

Create a string in MiniZinc

Super class

rminizinc::Expression -> String

Public fields

.value string value

Active bindings

.value string value

Methods

Public methods:

- String$new()
- String$getV()
- String$setV()
- String$c_str()
- String$clone()

Method new(): constructor

Usage:
String$new(val)

Method setV(): set value

Usage:
String$setV(val)

Arguments:
val string value

Method c_str(): get the MiniZinc representation

Usage:
String$c_str()

Method clone(): The objects of this class are cloneable with this method.

Usage:
String$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

Examples

newString = String$new("example")
newString$c_str()
newString$setV("new example")
newString$c_str()

---

StringArrDecl n-D String array declaration

Description

Declare a new n-dimensional array of strings

Usage

StringArrDecl(name, kind, ind, value = NULL, ndim)
**stringExpressions**

**Arguments**

- **name**: variable/parameter name
- **kind**: "var" or "par"
- **ind**: index of the array
- **value**: value (NULL by default)
- **ndim**: number of dimensions of the array

---

**Description**

Get a list of string expressions

**Usage**

`stringExpressions(vals)`

**Arguments**

- **vals**: vector of string values

---

**StringSetDecl**

**set of string declaration**

**Description**

Declare a new set of string

**Usage**

`StringSetDecl(name, kind, value = NULL)`

**Arguments**

- **name**: variable/parameter name
- **kind**: "var" or "par"
- **value**: provide a Set object (or NULL)
Type

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The information of different data types</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Public fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>.bt the base type</td>
</tr>
<tr>
<td>.kind parameter or decision</td>
</tr>
<tr>
<td>.dim the number of dimensions set or plain</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Active bindings</th>
</tr>
</thead>
<tbody>
<tr>
<td>.bt the base type</td>
</tr>
<tr>
<td>.kind parameter or decision</td>
</tr>
<tr>
<td>.dim the number of dimensions set or plain</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
</table>

**Public methods:**
- Type$new()
- Type$bt()
- Type$st()
- Type$kind()
- Type$ndim()
- Type$isInt()
- Type$isFloat()
- Type$isBool()
- Type$isString()
- Type$isSet()
- Type$isIntSet()
- Type$isFloatSet()
- Type$isBoolSet()
- Type$clone()

**Method** `new()`: constructor

**Usage:**
Type$new(base_type, kind, dim = 0, set_type = FALSE)

**Arguments:**
- base_type the base type
- kind parameter or decision
dim the number of dimensions
set_type set or plain

Method bt(): return the base type
Usage:
Type$bt()

Method st(): return if it's set type
Usage:
Type$st()

Method kind(): return the kind
Usage:
Type$kind()

Method ndim(): return the number of dimensions
Usage:
Type$ndim()

Method isInt(): check if it's an int
Usage:
Type$isInt()

Method isFloat(): check if it's a float
Usage:
Type$isFloat()

Method isBool(): check if it's a bool
Usage:
Type$isBool()

Method isString(): check if it's a string
Usage:
Type$isString()

Method isSet(): return if set in MiniZinc
Usage:
Type$isSet()

Method isIntSet(): check if it's a set of int
Usage:
Type$isIntSet()

Method isFloatSet(): check if it's a set of float
Usage:
Type$isFloatSet()
**Method** isBoolSet(): check if it's a set of bool

*Usage:*

Type$isBoolSet()

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

Type$clone(deep = FALSE)

*Arguments:*

- `deep` Whether to make a deep clone.

---

**Description**

Create type instantiation with indices, etc.

**Super class**

`rminizinc::Expression` -> `TypeInst`

**Public fields**

- `.indExpr` the index expression
- `.domain` the domain of possible values to be taken
- `.type` the type information

**Active bindings**

- `.indExpr` the index expression
- `.domain` the domain of possible values to be taken
- `.type` the type information

**Methods**

**Public methods:**

- `TypeInst$new()`
- `TypeInst$getDomain()`
- `TypeInst$setDomain()`
- `TypeInst$ranges()`
- `TypeInst$isArray()`
- `TypeInst$isBoolSet()`
- `TypeInst$clone()`
Method `new()`: constructor

Usage:
TypeInst$new(type, indexExprVec = NULL, domain = NULL)

Arguments:
type type of declaration
indexExprVec expression list of indices
domain the domain of decision variables

Method `getDomain()`: get the variable domain

Usage:
TypeInst$getDomain()

Method `setDomain()`: set the variable domain

Usage:
TypeInst$setDomain(dom)

Arguments:
dom domain expression to be set

Method `ranges()`: return the index expression vector

Usage:
TypeInst$ranges()

Method `isArray()`: check if it's an array

Usage:
TypeInst$isArray()

Method `type()`: return the type information

Usage:
TypeInst$type()

Method `clone()`: The objects of this class are cloneable with this method.

Usage:
TypeInst$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

Examples

TypeInst$new(type = Type$new(base_type = "int", kind = "par", dim = 1),
domain = Set$new(IntSetVal$new(2,5)))
Description
Unary operation expression in MiniZinc Possible unary operators are: "+", "-", "not"

Super class
\texttt{rminizinc::Expression} \rightarrow \texttt{UnOp}

Public fields
\begin{itemize}
  \item \texttt{.args} list of expression arguments
  \item \texttt{.op} operator to be used
  \item \texttt{.delete\_flag} used to delete items
\end{itemize}

Active bindings
\begin{itemize}
  \item \texttt{.args} list of expression arguments
  \item \texttt{.op} operator to be used
  \item \texttt{.delete\_flag} used to delete items
\end{itemize}

Methods
Public methods:
\begin{itemize}
  \item \texttt{UnOp$new()}
  \item \texttt{UnOp$nargs()}
  \item \texttt{UnOp$getArgs()}
  \item \texttt{UnOp$setArgs()}
  \item \texttt{UnOp$getArg()}
  \item \texttt{UnOp$setArg()}
  \item \texttt{UnOp$getOp()}
  \item \texttt{UnOp$setOp()}
  \item \texttt{UnOp$c\_str()}
  \item \texttt{UnOp$getDeleteFlag()}
  \item \texttt{UnOp$delete()}
  \item \texttt{UnOp$clone()}
\end{itemize}

Method \texttt{new()}: constructor

\textit{Usage:}
\texttt{UnOp$new(args, op)}

\textit{Arguments:}
UnOp

args  list of expressions
op    unary operator

Method nargs(): get the number of arguments
  Usage:
  UnOp$nargs()

Method getArgs(): get all expression arguments
  Usage:
  UnOp$getArgs()

Method setArgs(): set all expression arguments
  Usage:
  UnOp$setArgs()
  Arguments:
  args  argument list to be set

Method getArg(): get the ith expression argument
  Usage:
  UnOp$getArg(i)
  Arguments:
  i    index

Method setArg(): set the ith expression argument
  Usage:
  UnOp$setArg(i, val)
  Arguments:
  i    index
  val  value of expression to be set

Method getOp(): get the unary operator
  Usage:
  UnOp$getOp()

Method setOp(): set the unary operator
  Usage:
  UnOp$setOp(unop)
  Arguments:
  unop  unary operator to be set

Method c_str(): return the MiniZinc representation
  Usage:
  UnOp$c_str()
Method `getDeleteFlag()`: delete flag for internal use

Usage:
UnOp$getDeleteFlag()

Method `delete()`: delete the assignment item

Usage:
UnOp$delete()

Method `clone()`: The objects of this class are cloneable with this method.

Usage:
UnOp$clone(deep = FALSE)

Arguments:
  deep  Whether to make a deep clone.

Examples

    newUnOp = UnOp$new(args = list(Int$new(5)), op = "-")
    newUnOp$c_str()
    newUnOp$setArg(1, Int$new(6))
    newUnOp$setOp("+")
    newUnOp$c_str()

---

Description
Contains different fields to create a variable declaration

Super class

`rminizinc::Expression` -> VarDecl

Public fields

- .ti  type instantiation information
- id  name of the variable
- .expression  the initialization expression
- .delete_flag  used to delete items

Active bindings

- .ti  type instantiation information
- id  name of the variable
- .expression  the initialization expression
- .delete_flag  used to delete items
Methods

Public methods:

• `VarDecl$new()`
• `VarDecl$getId()`
• `VarDecl$setId()`
• `VarDecl$isPar()`
• `VarDecl$isVar()`
• `VarDecl$setDomain()`
• `VarDecl$getDomain()`
• `VarDecl$getValue()`
• `VarDecl$setValue()`
• `VarDecl$ti()`
• `VarDecl$c_str()`
• `VarDecl$getValueFlag()`
• `VarDecl$delete()`
• `VarDecl$clone()`

Method `new()`: constructor

Usage:
VarDecl$new(name, type_inst, value = NULL)

Arguments:
name  the identifier/name
type_inst  type instantiation of the variable
value  value of variable, NULL by default

Method `getId()`: get the identifier object

Usage:
VarDecl$getId()

Method `setId()`: set the identifier object name

Usage:
VarDecl$setId(name)

Arguments:
name  name to be set

Method `isPar()`: check if it’s a parameter

Usage:
VarDecl$isPar()

Method `isVar()`: check if it’s a decision variable

Usage:
VarDecl$isVar()
**Method** `setDomain()`: overwrite the existing domain  
  **Usage:**  
  ```cpp
  VarDecl$setDomain(dom)
  ```  
  **Arguments:**  
  `dom` domain expression to be set

**Method** `getDomain()`: get the variable domain  
  **Usage:**  
  ```cpp
  VarDecl$getDomain()
  ```

**Method** `getValue()`: get the value  
  **Usage:**  
  ```cpp
  VarDecl$getValue()
  ```

**Method** `setValue()`: set the value  
  **Usage:**  
  ```cpp
  VarDecl$setValue(val)
  ```  
  **Arguments:**  
  `val` expression to be set (NULL to remove value)

**Method** `ti()`: get the type-inst of the variable declaration  
  **Usage:**  
  ```cpp
  VarDecl$ti()
  ```

**Method** `c_str()`: get the domain of the variable  
  return string representation of MiniZinc  
  **Usage:**  
  ```cpp
  VarDecl$c_str()
  ```

**Method** `getDeleteFlag()`: delete flag for internal use  
  **Usage:**  
  ```cpp
  VarDecl$getDeleteFlag()
  ```

**Method** `delete()`: delete the assignment item  
  **Usage:**  
  ```cpp
  VarDecl$delete()
  ```

**Method** `clone()`: The objects of this class are cloneable with this method.  
  **Usage:**  
  ```cpp
  VarDecl$clone(deep = FALSE)
  ```  
  **Arguments:**  
  `deep` Whether to make a deep clone.

**Examples**

```cpp
newVarDecl = VarDecl$new(name = "n",
           type_inst = TypeInst$new(Type$new(base_type = "int", kind = "par")))
newVarDecl$c_str()
```
VarDeclItem

The variable declaration item

Description
Declaration items in the model

Super class
rminizinc::Item -> VarDeclItem

Public fields
- .decl the declaration expression
- .delete_flag used to delete items

Active bindings
- .decl the declaration expression
- .delete_flag used to delete items

Methods
Public methods:
- VarDeclItem$new()
- VarDeclItem$getDecl()
- VarDeclItem$setDecl()
- VarDeclItem$getId()
- VarDeclItem$c_str()
- VarDeclItem$getDeleteFlag()
- VarDeclItem$delete()
- VarDeclItem$clone()

Method new(): constructor
Usage:
VarDeclItem$new(decl = NULL, mzn_str = NULL)
Arguments:
decl the declaration expression object
mzn_str string representation of variable declaration item

Method getDecl(): get the variable declaration
Usage:
VarDeclItem$getDecl()
**Method** setDecl(): set the variable declaration

_Usage:_

VarDeclItem$setDecl(e)

_Aroutines:_

e var decl expression

**Method** getId(): get the identifier object for the variable

_Usage:_

VarDeclItem$getId()

**Method** c_str(): set the variable declaration convert the declaration to String

_Usage:_

VarDeclItem$c_str()

**Method** getDeleteFlag(): delete flag for internal use

_Usage:_

VarDeclItem$getDeleteFlag()

**Method** delete(): delete the variable item

_Usage:_

VarDeclItem$delete()

**Method** clone(): The objects of this class are cloneable with this method.

_Usage:_

VarDeclItem$clone(deep = FALSE)

_Arguments:_

deep Whether to make a deep clone.

---

**VarDomainDecl**

_declare 0-D variable with domain_

**Description**

Declare a 0 dimensional (int, float, bool or string) variable with domain

**Usage**

VarDomainDecl(name, dom)

**Arguments**

<table>
<thead>
<tr>
<th>name</th>
<th>variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>dom</td>
<td>domain</td>
</tr>
</tbody>
</table>
Index

* datasets
  LIBMINIZINC_PATH, 55
  PROJECT_DIRECTORY, 61
  SOLVER_BIN, 66

Annotation, 4
Array, 5
ArrayAccess, 8
ArrDomainDecl, 10
AssignItem, 10
assignment, 12
assignment_2, 13

BinOp, 13
Bool, 16
BoolArrDecl, 17
BoolDecl, 17
boolExpressions, 18
BoolSetDecl, 18

Call, 19
Comprehension, 21
ConstraintItem, 24

Expression, 25
expressionDelete, 26

Float, 26
FloatArrDecl, 27
FloatDecl, 28
floatExpressions, 28
FloatSetDecl, 29
FloatSetVal, 29
FloatVal, 31
FunctionItem, 32

Generator, 34
get_missing_pars, 37
getRModel, 37
getType, 37

helperDeleteExpression, 38
helperDeleteItem, 38

Id, 38
IncludeItem, 40
initExpression, 41
initItem, 42
Int, 42
IntArrDecl, 43
IntDecl, 44
intExpressions, 44
IntSetDecl, 45
IntSetVal, 45
IntVal, 47
Ite, 48
Item, 51
itemDelete, 51
iterExpression, 52
iterItem, 52

knapsack, 53

Let, 53
LIBMINIZINC_PATH, 55

magic_series, 56
magic_square, 56
Model, 57
mzn_eval, 59
mzn_parse, 60

production_planning, 60
PROJECT_DIRECTORY, 61

rminizinc (rminizinc-package), 3
rminizinc-package, 3
rminizinc::Expression, 5, 8, 13, 16, 19, 21, 26, 34, 38, 42, 48, 53, 61, 67, 72, 74, 76
rminizinc::Item, 10, 24, 32, 40, 64, 79
Set, 61
set_params, 64
sol_parse, 67
SolveItem, 64
SOLVER_BIN, 66
String, 67
StringArrDecl, 68
stringExpressions, 69
StringSetDecl, 69

Type, 70
TypeInst, 72

UnOp, 74

VarDecl, 76
VarDeclItem, 79
VarDomainDecl, 80