# Package: coinclp (via r-universe)

October 16, 2026

**Type** Package

**Title** R Interface to the 'COIN-OR' 'Clp' Linear Programming Solver

**Version** 0.1.2

**Description** Solves linear programs with 'Clp', the simplex and interior point code of the 'COIN-OR' project <https://github.com/coin-or/Clp>. Provides a one-call solver interface for dense and sparse constraint matrices, and complete low level bindings to the 'Clp' callable library covering problem construction, warm starts, presolve options, basis access and 'MPS' files. A compatibility layer reproduces the interface of the archived 'clpAPI' package so that existing code keeps working. 'Clp' itself is not bundled and must be installed on the system; the 'Rtools' toolchain supplies it on 'Windows', from 'Rtools' 4.3 on.

**License** MIT + file LICENSE

**URL** https://github.com/SamLovick/coinclp

**BugReports** https://github.com/SamLovick/coinclp/issues

**Encoding** UTF-8

**Depends** R (>= 4.0)

**Imports** methods, stats, utils

**Suggests** Matrix, slam, knitr, rmarkdown

**VignetteBuilder** knitr

**SystemRequirements** COIN-OR Clp (>= 1.16) with development headers, and a C++17 compiler. Debian/Ubuntu: coinor-libclp-dev, Fedora: coin-or-Clp-devel, macOS: brew install clp, Windows: supplied by Rtools (4.3 or later). CLP_CFLAGS and CLP_LIBS point the build at an installation elsewhere.

**NeedsCompilation** yes

**Config/roxygen2/version** 8.1.0

**Author** Sam Lovick [aut, cre]

**Maintainer** Sam Lovick <sam@lovickconsulting.com>

**Config/pak/sysreqs** coinor-libclp-dev

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

**Date/Publication** 2026-10-16 14:50:13 UTC

**RemoteUrl** https://github.com/cran/coinclp

**RemoteRef** HEAD

**RemoteSha** adbfe02710f043da7f3c693ff3ce4a34837bb065

# Contents

addRowsCLP . . . 3
clp_add_rows . . . 4
clp_col_solution . . . 5
clp_control . . . 6
clp_free . . . 7
clp_inf . . . 8
clp_initial_solve . . . 8
clp_is_open . . . 9
clp_is_proven_optimal . . . 10
clp_load_problem . . . 11
clp_load_quadratic_objective . . . 12
clp_log_level . . . 13
clp_matrix . . . 14
clp_model . . . 15
clp_modify_coefficient . . . 15
clp_num_rows . . . 16
clp_objective . . . 17
clp_objective_value . . . 18
clp_optimization_direction . . . 18
clp_options . . . 19
clp_options_do_dual . . . 20
clp_options_free . . . 21
clp_options_set_solve_type . . . 22
clp_primal_tolerance . . . 23
clp_print_model . . . 24
clp_problem_name . . . 25
clp_read_mps . . . 26
clp_resize . . . 26
clp_row_names . . . 27
clp_save_model . . . 28
clp_set_integer . . . 29
clp_solve . . . 30
clp_status . . . 32
clp_status_exists . . . 33
clp_sum_primal_infeasibilities . . . 34
clp_unbounded_ray . . . 34

clp_version . . . . . 35
clp_write_mps . . . . . 36
clpPtr-accessors . . . . . 37
clpPtr-class . . . . . 38
copyNamesCLP . . . . . 38
getNumRowsCLP . . . . . 40
getSolStatusCLP . . . . . 41
initProbCLP . . . . . 42
loadProblemCLP . . . . . 43
resizeCLP . . . . . 44
setObjDirCLP . . . . . 45
solveInitialCLP . . . . . 46
versionCLP . . . . . 47

**Index** **48**

---

`addRowsCLP` *Add and delete rows and columns, clpAPI style*

---

## Description

Add and delete rows and columns, clpAPI style

## Usage

```
addRowsCLP(lp, nrows, lb, ub, rowst, cols, val)

addColsCLP(lp, ncols, lb, ub, obj, colst, rows, val)

delRowsCLP(lp, num, i)

delColsCLP(lp, num, j)
```

## Arguments

| | |
|---|---|
| `lp` | A `clpPtr` object. |
| `nrows`, `ncols` | Number of rows or columns to add. |
| `lb`, `ub` | Bounds for the new rows or columns. |
| `rowst`, `colst` | Integer vectors of starts, length `n + 1`. |
| `cols`, `rows` | 0-based indices of the new entries. |
| `val` | Numeric vector of the new entries. |
| `obj` | Objective coefficients for the new columns. |
| `num` | Number of rows or columns to delete. |
| `i`, `j` | 0-based positions to delete. |

## Value

NULL, invisibly.

## Examples

```
lp <- initProbCLP()
loadMatrixCLP(lp, 2, 0, c(0, 0, 0), integer(0), numeric(0))
addRowsCLP(lp, 1, -1e30, 4, c(0, 2), c(0, 1), c(1, 1))
getNumRowsCLP(lp)
delProbCLP(lp)
```

---

# clp_add_rows *Add or delete rows and columns*

---

## Description

clp_add_rows() and clp_add_columns() extend a model with entries given in compressed sparse form; clp_delete_rows() and clp_delete_columns() remove them. All indices are 0-based.

## Usage

```
clp_add_rows(
  model,
  number,
  rowlb = NULL,
  rowub = NULL,
  rowstarts,
  columns,
  elements
)

clp_add_columns(
  model,
  number,
  collb = NULL,
  colub = NULL,
  obj = NULL,
  colstarts,
  rows,
  elements
)

clp_delete_rows(model, which)

clp_delete_columns(model, which)
```

### Arguments

| | |
|---|---|
| model | A "clp_model" object. |
| number | Number of rows or columns being added. |
| rowlb, rowub | Numeric vectors of bounds for the new rows, or NULL. |
| rowstarts | Integer vector of length number + 1, where each new row's entries start. |
| columns | Integer vector of 0-based column indices of the new entries. |
| elements | Numeric vector of matrix entries. |
| collb, colub | Numeric vectors of bounds for the new columns, or NULL. |
| obj | Objective coefficients for the new columns, or NULL. |
| colstarts | Integer vector of length number + 1, where each new column's entries start. |
| rows | Integer vector of 0-based row indices of the new entries. |
| which | Integer vector of 0-based positions to delete. |

### Value

NULL, invisibly.

### Examples

```
model <- clp_model()
clp_load_problem(model, 2, 0, start = c(0L, 0L, 0L),
                 index = integer(0), value = numeric(0),
                 obj = c(1, 1))
clp_add_rows(model, 1, rowlb = -clp_inf(), rowub = 5,
             rowstarts = c(0L, 2L), columns = c(0L, 1L), elements = c(1, 1))
clp_num_rows(model)
clp_free(model)
```

---

## clp_col_solution *Solution vectors*

---

### Description

After a solve, these return the primal and dual solution. clp_col_solution() is the primal solution, clp_reduced_costs() the dual values on the columns, clp_row_activity() the row activities and clp_row_price() the dual values (shadow prices) on the rows.

### Usage

```
clp_col_solution(model)

clp_reduced_costs(model)

clp_row_activity(model)
```

```
clp_row_price(model)

clp_set_col_solution(model, value)
```

## Arguments

| | |
|---|---|
| model | A "clp_model" object. |
| value | Numeric starting solution, one entry per column. |

## Value

A numeric vector, or NULL invisibly for the setter.

## Examples

```
model <- clp_model()
clp_load_problem(model, 2, 1, c(0L, 1L, 2L), c(0L, 0L), c(1, 1),
                 obj = c(-1, -1), rowub = 1)
clp_initial_solve(model)
clp_col_solution(model)
clp_row_price(model)
clp_free(model)
```

---

# clp_control *Control parameters for clp_solve()*

---

## Description

Control parameters for clp_solve()

## Usage

```
clp_control(
  log_level = 0L,
  algorithm = c("auto", "primal", "dual", "barrier", "barrier_nocross"),
  presolve = TRUE,
  max_iterations = NULL,
  max_seconds = NULL,
  primal_tolerance = NULL,
  dual_tolerance = NULL,
  scaling = NULL
)
```

### Arguments

| | |
|---|---|
| log_level | How much Clp prints: 0 silent (the default), up to 4. |
| algorithm | One of "auto", "primal", "dual", "barrier" or "barrier_nocross". |
| presolve | Run Clp's presolve. Ignored when algorithm is "auto" and presolve is TRUE, which is Clp's own default path. |
| max_iterations | Iteration limit, or NULL for Clp's default. |
| max_seconds | Time limit in seconds, or NULL for no limit. |
| primal_tolerance, dual_tolerance | Simplex tolerances, or NULL for Clp's defaults. |
| scaling | Scaling mode: 0 off, 1 equilibrium, 2 geometric, 3 auto, 4 dynamic, or NULL for Clp's default. |

### Value

A list of control settings for clp_solve.

### Examples

```
clp_control(algorithm = "dual", max_seconds = 10)
```

---

## clp_free *Release a Clp model*

---

### Description

Frees the solver memory behind a model. This happens automatically when the model is garbage collected; call it explicitly when working with many or large models. Calling it twice is harmless, but using the model afterwards is an error.

### Usage

```
clp_free(model)
```

### Arguments

| | |
|---|---|
| model | A "clp_model" object. |

### Value

NULL, invisibly.

### Examples

```
model <- clp_model()
clp_free(model)
clp_is_open(model)
```

## clp_inf *The value Clp treats as infinite*

### Description

Clp represents an infinite bound by 1e30 rather than by `Inf`. `clp_inf()` returns that value, and the high level interface translates `Inf` and `-Inf` to it automatically.

### Usage

```
clp_inf()
```

### Value

A single number, 1e30.

### Examples

```
clp_inf()
```

## clp_initial_solve *Solve a loaded model*

### Description

The entry points of the Clp callable library. `clp_initial_solve()` lets Clp choose the algorithm and applies presolve, which is the right default for most problems; the others force a particular method. `clp_primal_simplex()` and `clp_dual_simplex()` run the simplex without presolve, which is what you want when re-solving a modified model from a warm start.

### Usage

```
clp_initial_solve(model)

clp_initial_dual_solve(model)

clp_initial_primal_solve(model)

clp_initial_barrier_solve(model)

clp_initial_barrier_no_cross_solve(model)

clp_initial_solve_with_options(model, options)

clp_primal_simplex(model, if_values_pass = 0L)
```

```
clp_dual_simplex(model, if_values_pass = 0L)

clp_idiot(model, try_hard = 0L)

clp_crash(model, gap = 0, pivot = 0L)
```

## Arguments

| | |
|---|---|
| model | A "clp_model" object. |
| options | A "clp_options" object from clp_options. |
| if_values_pass | Pass 1 to start from the values in clp_set_col_solution, 0 otherwise. |
| try_hard | Effort level for the idiot crash. |
| gap | Bound gap below which variables may be flipped by the crash. |
| pivot | Crash pivoting rule: 0 none, 1 simple, 2 mini iterations. |

## Value

An integer return code from Clp: 0 if it solved the problem, 1 if the problem is primal infeasible, 2 if dual infeasible, and other values if it stopped early. clp_idiot() returns NULL invisibly. Use clp_status for the status of the model itself.

## Examples

```
model <- clp_model()
clp_load_problem(model, 2, 1, c(0L, 1L, 2L), c(0L, 0L), c(1, 1),
                 obj = c(-1, -2), rowub = 3)
clp_set_log_level(model, 0L)
clp_initial_solve(model)
clp_objective_value(model)
clp_free(model)
```

---

# clp_is_open *Is a model still usable?*

---

## Description

Is a model still usable?

## Usage

```
clp_is_open(model)
```

## Arguments

| | |
|---|---|
| model | A "clp_model" object. |

## Value

`TRUE` if the model still holds a live Clp problem.

## Examples

```
model <- clp_model()
clp_is_open(model)
```

---

`clp_is_proven_optimal` *Feasibility and optimality flags*

---

## Description

Convenience predicates over the last solve, mirroring the OSI-style queries in the Clp callable library.

## Usage

```
clp_is_proven_optimal(model)

clp_is_proven_primal_infeasible(model)

clp_is_proven_dual_infeasible(model)

clp_is_abandoned(model)

clp_is_primal_objective_limit_reached(model)

clp_is_dual_objective_limit_reached(model)

clp_is_iteration_limit_reached(model)

clp_primal_feasible(model)

clp_dual_feasible(model)
```

## Arguments

| | |
|---|---|
| `model` | A `"clp_model"` object. |

## Value

A single logical value.

## Examples

```
model <- clp_model()
clp_load_problem(model, 1, 1, c(0L, 1L), 0L, 1, obj = 1, rowlb = 1)
clp_initial_solve(model)
clp_is_proven_optimal(model)
clp_free(model)
```

---

`clp_load_problem` *Load a problem into a Clp model*

---

## Description

Loads a complete linear program given by a column-major (compressed sparse column) constraint matrix. This is the callable library's `Clp_loadProblem` and takes 0-based indices.

## Usage

```
clp_load_problem(
  model,
  ncols,
  nrows,
  start,
  index,
  value,
  collb = NULL,
  colub = NULL,
  obj = NULL,
  rowlb = NULL,
  rowub = NULL
)
```

## Arguments

| | |
|---|---|
| `model` | A `"clp_model"` object. |
| `ncols` | Number of columns (variables). |
| `nrows` | Number of rows (constraints). |
| `start` | Integer vector of length `ncols + 1` giving the 0-based position in `index`/`value` at which each column starts. |
| `index` | Integer vector of 0-based row indices, one per matrix entry. |
| `value` | Numeric vector of matrix entries. |
| `collb`, `colub` | Numeric vectors of column bounds, or `NULL`. |
| `obj` | Numeric vector of objective coefficients, or `NULL`. |
| `rowlb`, `rowub` | Numeric vectors of row bounds, or `NULL`. |

## Details

Any of the bound and objective arguments may be `NULL`, in which case Clp applies its defaults: columns get `[0, Inf)` bounds and a zero objective, rows get `(-Inf, Inf)`.

## Value

`NULL`, invisibly. The model is modified in place.

## See Also

`clp_solve`, which builds this representation from an ordinary R matrix.

## Examples

```
# maximise 2x + 3y subject to x + y <= 4, x + 3y <= 6
model <- clp_model()
clp_load_problem(model, ncols = 2, nrows = 2,
                 start = c(0L, 2L, 4L),
                 index = c(0L, 1L, 0L, 1L),
                 value = c(1, 1, 1, 3),
                 collb = c(0, 0), colub = c(clp_inf(), clp_inf()),
                 obj = c(-2, -3),
                 rowlb = c(-clp_inf(), -clp_inf()), rowub = c(4, 6))
clp_initial_solve(model)
clp_col_solution(model)
clp_free(model)
```

---

# clp_load_quadratic_objective

*Load a quadratic objective*

---

## Description

Attaches a quadratic term to the objective, in column-major form. Clp itself only solves such models with its quadratic simplex; most users want the linear objective set by `clp_set_objective`.

## Usage

```
clp_load_quadratic_objective(model, ncols, start, column, element)
```

## Arguments

| | |
|---|---|
| `model` | A `"clp_model"` object. |
| `ncols` | Number of columns in the quadratic term. |
| `start` | Integer vector of 0-based column starts, length `ncols + 1`. |
| `column` | Integer vector of 0-based column indices. |
| `element` | Numeric vector of coefficients. |

## Value

NULL, invisibly.

## Examples

```
model <- clp_model()
clp_load_problem(model, 1, 0, start = c(0L, 0L), index = integer(0),
                 value = numeric(0))
clp_load_quadratic_objective(model, 1, c(0L, 1L), 0L, 2)
clp_free(model)
```

---

# clp_log_level *Algorithm settings*

---

## Description

clp_log_level() controls how much Clp prints: 0 silent, 1 just the final line, 2 factorizations, 3 more, 4 verbose. clp_scaling() selects the scaling mode (0 off, 1 equilibrium, 2 geometric, 3 auto, 4 dynamic). clp_perturbation() and clp_algorithm() expose the corresponding simplex settings.

## Usage

```
clp_log_level(model)

clp_set_log_level(model, value)

clp_scaling(model)

clp_set_scaling(model, value)

clp_perturbation(model)

clp_set_perturbation(model, value)

clp_algorithm(model)

clp_set_algorithm(model, value)

clp_iterations(model)

clp_set_iterations(model, value)
```

## Arguments

| | |
|---|---|
| model | A "clp_model" object. |
| value | The new value. |

## Value

The getters return an integer; the setters NULL invisibly.

## Examples

```
model <- clp_model()
clp_set_log_level(model, 0L)
clp_log_level(model)
clp_free(model)
```

---

# clp_matrix *The constraint matrix*

---

## Description

clp_matrix() returns the constraint matrix as triplets with 1-based positions, which is convenient in R. The other functions expose Clp's own column-major arrays unchanged, with 0-based indices; note that the stored matrix may contain gaps, so clp_vector_lengths() rather than the differences of clp_vector_starts() gives the entries per column.

## Usage

```
clp_matrix(model)

clp_vector_starts(model)

clp_vector_lengths(model)

clp_indices(model)

clp_elements(model)
```

## Arguments

| | |
|---|---|
| model | A "clp_model" object. |

## Value

clp_matrix() returns a list with components i, j, v, nrow and ncol. The others return the corresponding Clp array.

## Examples

```
model <- clp_model()
clp_load_problem(model, 2, 1, c(0L, 1L, 2L), c(0L, 0L), c(1, 2), rowub = 3)
clp_matrix(model)
clp_free(model)
```

## clp_model *Create a Clp model*

### Description

Creates an empty problem in the COIN-OR Clp callable library. The result is an external pointer with class `"clp_model"`; the underlying solver object is released when the model is garbage collected, or immediately by `clp_free`.

### Usage

```
clp_model()
```

### Details

New models start silent, unlike Clp's own default: raise the message level with `clp_set_log_level` to see the solver's reporting.

### Value

An object of class `"clp_model"`.

### See Also

`clp_solve` for a one-call interface that builds and solves a model for you, `clp_load_problem` for filling a model in yourself.

### Examples

```
model <- clp_model()
clp_num_cols(model)
clp_free(model)
```

## clp_modify_coefficient *Change one matrix coefficient*

### Description

Needs a Clp build that provides `Clp_modifyCoefficient` (1.18 or later); check with `clp_features()`.

### Usage

```
clp_modify_coefficient(model, row, col, value, keep_zero = TRUE)
```

### Arguments

| | |
|---|---|
| model | A "clp_model" object. |
| row, col | 0-based row and column position. |
| value | New coefficient. |
| keep_zero | Keep the entry in the sparse structure when value is zero. |

### Value

NULL, invisibly.

### Examples

```
if (isTRUE(clp_features()[["modify_coefficient"]])) {
  model <- clp_model()
  clp_load_problem(model, 1, 1, c(0L, 1L), 0L, 1, rowub = 1)
  clp_modify_coefficient(model, 0, 0, 2)
  clp_free(model)
}
```

---

## clp_num_rows *Size of a model*

---

### Description

Size of a model

### Usage

```
clp_num_rows(model)

clp_num_cols(model)

clp_num_elements(model)
```

### Arguments

| | |
|---|---|
| model | A "clp_model" object. |

### Value

The number of rows, columns or stored matrix entries. clp_num_elements() returns a double, since Clp counts entries in a type that may exceed the range of an R integer.

### Examples

```
model <- clp_model()
clp_resize(model, 3, 4)
c(clp_num_rows(model), clp_num_cols(model))
clp_free(model)
```

clp_objective *Bounds and objective coefficients*

## Description

Read or replace the objective coefficients and the row and column bounds of a model. The getters return Clp's own values, in which an infinite bound is `clp_inf()` rather than `Inf`.

## Usage

```
clp_objective(model)

clp_col_lower(model)

clp_col_upper(model)

clp_row_lower(model)

clp_row_upper(model)

clp_set_objective(model, value)

clp_set_col_lower(model, value)

clp_set_col_upper(model, value)

clp_set_row_lower(model, value)

clp_set_row_upper(model, value)
```

## Arguments

| | |
|---|---|
| `model` | A `"clp_model"` object. |
| `value` | A numeric vector with one entry per column (objective and column bounds) or per row (row bounds). |

## Value

The getters return a numeric vector; the setters return `NULL` invisibly.

## Examples

```
model <- clp_model()
clp_load_problem(model, 2, 0, start = c(0L, 0L, 0L),
                 index = integer(0), value = numeric(0), obj = c(1, 2))
clp_objective(model)
clp_set_objective(model, c(3, 4))
clp_objective(model)
```

```
clp_free(model)
```

---

`clp_objective_value` *Objective value and offset*

---

## Description

Objective value and offset

## Usage

```
clp_objective_value(model)

clp_objective_offset(model)

clp_set_objective_offset(model, value)
```

## Arguments

| | |
|---|---|
| `model` | A `"clp_model"` object. |
| `value` | A constant added to the objective. |

## Value

A number, or `NULL` invisibly for the setter.

## Examples

```
model <- clp_model()
clp_load_problem(model, 1, 1, c(0L, 1L), 0L, 1, obj = -1, rowub = 2)
clp_initial_solve(model)
clp_objective_value(model)
clp_free(model)
```

---

`clp_optimization_direction`
*Optimization direction*

---

## Description

Clp stores the direction as a number: 1 to minimise, -1 to maximise and 0 to ignore the objective. `clp_obj_sense()` is Clp's OSI-style spelling of the same setting.

### Usage

```
clp_optimization_direction(model)

clp_set_optimization_direction(model, value)

clp_obj_sense(model)

clp_set_obj_sense(model, value)
```

### Arguments

| | |
|---|---|
| model | A "clp_model" object. |
| value | 1 (minimise), -1 (maximise) or 0 (ignore the objective). |

### Value

The getters return a number; the setters NULL invisibly.

### Examples

```
model <- clp_model()
clp_set_optimization_direction(model, -1)
clp_optimization_direction(model)
clp_free(model)
```

---

## clp_options *Options for Clp's presolve and algorithm choice*

---

### Description

Creates a ClpSolve options object, the structure Clp uses to steer clp_initial_solve_with_options. Options are set with the clp_options_* functions.

### Usage

```
clp_options()
```

### Value

An external pointer with class "clp_options".

### Examples

```
opts <- clp_options()
clp_options_set_solve_type(opts, 1L)   # primal simplex
clp_options_get_solve_type(opts)
```

clp_options_do_dual *Individual presolve transformations*

## Description

Switch single presolve transformations on or off in a clp_options object. Each getter returns Clp's current setting for that transformation.

## Usage

```
clp_options_do_dual(options)

clp_options_set_do_dual(options, value)

clp_options_do_singleton(options)

clp_options_set_do_singleton(options, value)

clp_options_do_doubleton(options)

clp_options_set_do_doubleton(options, value)

clp_options_do_tripleton(options)

clp_options_set_do_tripleton(options, value)

clp_options_do_tighten(options)

clp_options_set_do_tighten(options, value)

clp_options_do_forcing(options)

clp_options_set_do_forcing(options, value)

clp_options_do_implied_free(options)

clp_options_set_do_implied_free(options, value)

clp_options_do_dupcol(options)

clp_options_set_do_dupcol(options, value)

clp_options_do_duprow(options)

clp_options_set_do_duprow(options, value)

clp_options_do_singleton_column(options)
```

```
clp_options_set_do_singleton_column(options, value)

clp_options_presolve_actions(options)

clp_options_set_presolve_actions(options, value)

clp_options_substitution(options)

clp_options_set_substitution(options, value)

clp_options_infeasible_return(options)

clp_options_set_infeasible_return(options, value)
```

## Arguments

| | |
|---|---|
| options | A "clp_options" object. |
| value | The new value; 0 or 1 for the switches. |

## Value

The getters return an integer; the setters NULL invisibly.

## Examples

```
opts <- clp_options()
clp_options_set_do_dual(opts, 1L)
clp_options_do_dual(opts)
clp_options_free(opts)
```

---

# clp_options_free *Release a Clp options object*

---

## Description

Release a Clp options object

## Usage

```
clp_options_free(options)
```

## Arguments

| | |
|---|---|
| options | A "clp_options" object. |

## Value

NULL, invisibly.

## Examples

```
opts <- clp_options()
clp_options_free(opts)
```

---

`clp_options_set_solve_type` *Presolve and algorithm options*

---

## Description

Settings on a `clp_options` object, passed to `clp_initial_solve_with_options`.

## Usage

```
clp_options_set_solve_type(options, method, extra = -1L)

clp_options_get_solve_type(options)

clp_options_set_presolve_type(options, amount, extra = -1L)

clp_options_get_presolve_type(options)

clp_options_get_presolve_passes(options)

clp_options_set_special_option(options, which, value, extra = -1L)

clp_options_get_special_option(options, which)

clp_options_get_extra_info(options, which)
```

## Arguments

| | |
|---|---|
| `options` | A `"clp_options"` object. |
| `method` | Algorithm code, see Details. |
| `extra` | Extra information for the setting; -1 selects Clp's default. |
| `amount` | Presolve code, see Details. |
| `which` | Index of the special option or extra information slot. |
| `value` | The new value. |

## Details

The solve type selects the algorithm: 0 dual simplex, 1 primal simplex, 2 primal or sprint, 3 barrier, 4 barrier without crossover, 5 automatic. The presolve type is 0 presolve on, 1 presolve off, 2 a fixed number of passes, 3 number and cost.

## Value

The getters return an integer; the setters NULL invisibly.

## Examples

```
opts <- clp_options()
clp_options_set_solve_type(opts, 0L)      # dual simplex
clp_options_set_presolve_type(opts, 1L)   # presolve off
clp_options_get_solve_type(opts)
clp_options_free(opts)
```

---

# clp_primal_tolerance *Numerical tolerances and limits*

---

## Description

Getters and setters for the simplex tolerances and the stopping criteria.

## Usage

```
clp_primal_tolerance(model)

clp_set_primal_tolerance(model, value)

clp_dual_tolerance(model)

clp_set_dual_tolerance(model, value)

clp_dual_objective_limit(model)

clp_set_dual_objective_limit(model, value)

clp_dual_bound(model)

clp_set_dual_bound(model, value)

clp_infeasibility_cost(model)

clp_set_infeasibility_cost(model, value)

clp_max_seconds(model)

clp_set_max_seconds(model, value)

clp_max_iterations(model)

clp_set_max_iterations(model, value)
```

```
clp_hit_max_iterations(model)

clp_small_element_value(model)

clp_set_small_element_value(model, value)
```

## Arguments

| | |
|---|---|
| model | A "clp_model" object. |
| value | The new value. |

## Details

`clp_set_max_seconds()` takes a number of seconds from now, and a negative value removes the limit. Clp turns that into an absolute deadline by adding the processor time already used, so `clp_max_seconds()` reads back the deadline rather than the number that was set.

## Value

The getters return a number; the setters `NULL` invisibly.

## Examples

```
model <- clp_model()
clp_set_primal_tolerance(model, 1e-8)
clp_primal_tolerance(model)
clp_set_max_iterations(model, 1000L)
clp_max_iterations(model)
clp_free(model)
```

---

# clp_print_model *Print a model through Clp*

---

## Description

Asks Clp to dump the model to the console. Useful for small problems when debugging a model build.

## Usage

```
clp_print_model(model, prefix = "clp")
```

## Arguments

| | |
|---|---|
| model | A "clp_model" object. |
| prefix | A string Clp puts in front of each line. |

### Value

NULL, invisibly.

### Examples

```
model <- clp_model()
clp_load_problem(model, 1, 1, c(0L, 1L), 0L, 1, rowub = 1)
clp_print_model(model)
clp_free(model)
```

---

## clp_problem_name *The name of the problem*

---

### Description

The name of the problem

### Usage

```
clp_problem_name(model)

clp_set_problem_name(model, name)
```

### Arguments

| | |
|---|---|
| model | A "clp_model" object. |
| name | A single character string. |

### Value

clp_problem_name() returns a string; the setter returns Clp's integer return code, invisibly.

### Examples

```
model <- clp_model()
clp_set_problem_name(model, "transport")
clp_problem_name(model)
clp_free(model)
```

## clp_read_mps *Read an MPS file*

### Description

Reads a problem in MPS format into a model, using Clp's own reader. Gzipped files are handled by Clp when it was built with zlib.

### Usage

```
clp_read_mps(model, file, keep_names = TRUE, ignore_errors = FALSE)
```

### Arguments

| | |
|---|---|
| model | A "clp_model" object. |
| file | Path to the MPS file. |
| keep_names | Keep the row and column names from the file. |
| ignore_errors | Carry on after errors in the file. |

### Value

Clp's integer return code, invisibly: 0 on success.

### Examples

```
model <- clp_model()
path <- system.file("extdata", "productmix.mps", package = "coinclp")
if (nzchar(path)) {
  clp_read_mps(model, path)
  clp_num_rows(model)
}
clp_free(model)
```

## clp_resize *Change the size of a model*

### Description

Change the size of a model

### Usage

```
clp_resize(model, nrows, ncols)
```

### Arguments

| | |
|---|---|
| model | A "clp_model" object. |
| nrows, ncols | New numbers of rows and columns. |

### Value

NULL, invisibly.

### Examples

```
model <- clp_model()
clp_resize(model, 2, 3)
clp_num_cols(model)
clp_free(model)
```

---

## clp_row_names *Row and column names*

---

### Description

clp_row_names() and clp_col_names() return all names as a character vector; clp_row_name() and clp_col_name() fetch a single one by 0-based position, as in the C API. A model that carries no names of its own reports the generated defaults "R0000000" and "C0000000"; clp_length_names() is 0 in that case.

### Usage

```
clp_row_names(model)

clp_col_names(model)

clp_row_name(model, index)

clp_col_name(model, index)

clp_set_names(model, row_names, col_names)

clp_set_row_name(model, index, name)

clp_set_col_name(model, index, name)

clp_drop_names(model)

clp_length_names(model)
```

### Arguments

| | |
|---|---|
| `model` | A `"clp_model"` object. |
| `index` | 0-based row or column position. |
| `row_names`, `col_names` | Character vectors with one entry per row or column. |
| `name` | A single name. |

### Details

`clp_set_names()` replaces every name at once and works with any Clp version. `clp_set_row_name()` and `clp_set_col_name()` change a single name but need Clp 1.18 or later; see `clp_features`.

### Value

The getters return character vectors or a single string; `clp_length_names()` an integer; the setters `NULL` invisibly.

### Examples

```
model <- clp_model()
clp_load_problem(model, 2, 1, c(0L, 1L, 2L), c(0L, 0L), c(1, 1), rowub = 1)
clp_set_names(model, "supply", c("x", "y"))
clp_col_names(model)
clp_free(model)
```

---

## `clp_save_model` *Save and restore a model*

---

### Description

Writes or reads Clp's own binary snapshot of a model. The format is Clp's internal one: it is not portable between Clp versions or machines, but it is a fast way to hand a model back to the same solver later. `clp_restore_model()` replaces whatever the model held.

### Usage

```
clp_save_model(model, file)

clp_restore_model(model, file)
```

### Arguments

| | |
|---|---|
| `model` | A `"clp_model"` object. |
| `file` | Path of the snapshot file. |

## Details

A note for anyone running this under valgrind: Clp's `saveModel()` writes a C struct to the file whole, and the struct's trailing padding bytes are never assigned, so valgrind reports uninitialised bytes passed to `write()` (Clp 1.17, 'ClpSimplex.cpp', Clp_scalars). It is harmless, since padding is never read back, but it comes from inside the Clp library and cannot be silenced from R. For a portable file that other software can read, use `clp_write_mps` instead.

## Value

Clp's integer return code, invisibly: 0 on success.

## Examples

```
## Not run:
# Not run in checks: Clp's saveModel() trips valgrind, see Details.
model <- clp_model()
clp_load_problem(model, 1, 1, c(0L, 1L), 0L, 1, obj = 1, rowub = 2)
path <- tempfile()
clp_save_model(model, path)
clp_restore_model(model, path)
clp_free(model)
unlink(path)

## End(Not run)
```

---

# clp_set_integer *Mark columns as integer*

---

## Description

Clp is a linear programming solver and always solves the continuous relaxation; these markers exist so that a model can be handed on to a branch and bound code such as Cbc.

## Usage

```
clp_set_integer(model, is_integer)

clp_delete_integer(model)

clp_integer_information(model)
```

## Arguments

| | |
|---|---|
| `model` | A `"clp_model"` object. |
| `is_integer` | Logical vector with one entry per column. |

## Value

`clp_integer_information()` returns a logical vector (empty when no markers are set); the others return `NULL` invisibly.

## Examples

```
model <- clp_model()
clp_load_problem(model, 2, 0, c(0L, 0L, 0L), integer(0), numeric(0))
clp_set_integer(model, c(TRUE, FALSE))
clp_integer_information(model)
clp_free(model)
```

---

# clp_solve *Solve a linear program*

---

## Description

Builds a Clp model from ordinary R objects, solves it and returns the solution. The model is freed before the function returns.

## Usage

```
clp_solve(
  objective,
  constraints = NULL,
  dir = "<=",
  rhs = NULL,
  row_lower = NULL,
  row_upper = NULL,
  lower = 0,
  upper = Inf,
  max = FALSE,
  control = clp_control(),
  col_names = NULL,
  row_names = NULL
)
```

## Arguments

| | |
|---|---|
| `objective` | Numeric vector of objective coefficients, one per variable. |
| `constraints` | Constraint matrix, or `NULL` for a problem with only variable bounds. |
| `dir` | Character vector of constraint directions, recycled over the rows: `"<="`, `">="` or `"=="`. |
| `rhs` | Numeric right hand side, one entry per row. |
| `row_lower, row_upper` | Row bounds, as an alternative to `dir`/`rhs`. |

`lower`, `upper` Variable bounds, scalars or one entry per variable. The default lower bound is 0, as in most LP formulations.

`max` Maximise instead of minimise.

`control` A list from `clp_control`.

`col_names`, `row_names` Optional names, used for the returned vectors.

## Details

Constraints are given either by `dir` and `rhs`, the usual one-sided form, or by `row_lower` and `row_upper`, which also covers ranged constraints (`2 <= x + y <= 5`). Infinite bounds are written as `Inf` and `-Inf`.

`constraints` may be a dense matrix, a sparse matrix from the **Matrix** package, a `simple_triplet_matrix` from **slam**, or a list of triplets with components `i`, `j`, `v` plus `nrow` and `ncol`.

## Value

An object of class `"clp_solution"`: a list with

`objval` the objective value at the solution

`solution` the primal solution

`status` Clp's status code, see `clp_status`

`status_message` that code as text

`optimal` `TRUE` when Clp proved optimality

`duals` dual values (shadow prices) for the rows

`reduced_costs` reduced costs for the columns

`row_activity` the value of each row at the solution

`iterations` simplex iterations used

## See Also

`clp_model` and `clp_load_problem` for building a model that you keep, modify and re-solve.

## Examples

```
# maximise 143x + 60y subject to
#   120x +  210y <= 15000
#   110x +   30y <=  4000
#      x +     y <=    75
A <- rbind(c(120, 210), c(110, 30), c(1, 1))
res <- clp_solve(c(143, 60), A, "<=", c(15000, 4000, 75), max = TRUE)
res$objval
res$solution
```

clp_status *Solution status*

## Description

clp_status() returns Clp's problem status: 0 optimal, 1 primal infeasible, 2 dual infeasible (unbounded), 3 stopped on a limit, 4 stopped because of errors, and -1 when the problem has not been solved. clp_status_message() turns such a code into a short description.

## Usage

```
clp_status(model)

clp_set_status(model, value)

clp_secondary_status(model)

clp_set_secondary_status(model, value)

clp_status_message(status)
```

## Arguments

| | |
|---|---|
| model | A "clp_model" object. |
| value | A new status code. |
| status | An integer status code. |

## Value

clp_status() and clp_secondary_status() return an integer, clp_status_message() a character string, the setters NULL invisibly.

## Examples

```
model <- clp_model()
clp_load_problem(model, 1, 1, c(0L, 1L), 0L, 1, obj = 1, rowlb = 1)
clp_initial_solve(model)
clp_status_message(clp_status(model))
clp_free(model)
```

clp_status_exists *Basis status for warm starts*

## Description

`clp_status_array()` returns Clp's packed basis, rows first, which can be handed back to a model of the same size with `clp_copyin_status()` to warm start it. The per-variable accessors use Clp's codes: 0 free, 1 basic, 2 at upper bound, 3 at lower bound, 4 superbasic, 5 fixed.

## Usage

```
clp_status_exists(model)

clp_status_array(model)

clp_copyin_status(model, status)

clp_row_status(model, index)

clp_col_status(model, index)

clp_set_row_status(model, index, value)

clp_set_col_status(model, index, value)
```

## Arguments

| | |
|---|---|
| `model` | A `"clp_model"` object. |
| `status` | A raw vector previously returned by `clp_status_array()`. |
| `index` | 0-based row or column position. |
| `value` | New status code. |

## Value

`clp_status_exists()` a logical, `clp_status_array()` a raw vector, the accessors an integer status code, the setters `NULL` invisibly.

## Examples

```
model <- clp_model()
clp_load_problem(model, 2, 1, c(0L, 1L, 2L), c(0L, 0L), c(1, 1),
                 obj = c(-1, -1), rowub = 1)
clp_initial_solve(model)
basis <- clp_status_array(model)
clp_copyin_status(model, basis)
clp_free(model)
```

## clp_sum_primal_infeasibilities *Infeasibility measures*

### Description

Infeasibility measures

### Usage

```
clp_sum_primal_infeasibilities(model)

clp_sum_dual_infeasibilities(model)

clp_num_primal_infeasibilities(model)

clp_num_dual_infeasibilities(model)

clp_check_solution(model)
```

### Arguments

| | |
|---|---|
| model | A "clp_model" object. |

### Value

A number of violations, or the sum of their sizes.

### Examples

```
model <- clp_model()
clp_load_problem(model, 1, 1, c(0L, 1L), 0L, 1, obj = 1, rowlb = 1)
clp_initial_solve(model)
clp_sum_primal_infeasibilities(model)
clp_free(model)
```

## clp_unbounded_ray *Rays certifying unboundedness or infeasibility*

### Description

Rays certifying unboundedness or infeasibility

### Usage

```
clp_unbounded_ray(model)

clp_infeasibility_ray(model)
```

### Arguments

| | |
|---|---|
| model | A "clp_model" object. |

### Value

A numeric vector, empty when Clp holds no such ray.

### Examples

```
model <- clp_model()
clp_load_problem(model, 1, 1, c(0L, 1L), 0L, 1, obj = -1, rowub = clp_inf())
clp_initial_solve(model)
clp_unbounded_ray(model)
clp_free(model)
```

---

## clp_version *Version of the Clp library in use*

---

### Description

Version of the Clp library in use

### Usage

```
clp_version()

clp_features()
```

### Value

`clp_version()` returns a list with the version string and its major, minor and release components. `clp_features()` returns a named logical vector saying which optional entry points this build of Clp provides.

### Examples

```
clp_version()
clp_features()
```

`clp_write_mps` *Write an MPS file*

## Description

Writes the model in MPS format. Clp gained `Clp_writeMps` in its C API after the 1.17 series, so where that entry point is missing (see `clp_features`) this falls back to an MPS writer implemented in R, which writes the same problem in fixed-column MPS format.

## Usage

```
clp_write_mps(
  model,
  file,
  format_type = 0L,
  number_across = 2L,
  obj_sense = 1,
  force_r = FALSE
)
```

## Arguments

| | |
|---|---|
| `model` | A `"clp_model"` object. |
| `file` | Path of the file to write. |
| `format_type` | Passed to Clp: 0 normal, 1 extra accuracy, 2 IEEE hex. Ignored by the R fallback, which always writes full precision. |
| `number_across` | Passed to Clp: 1 or 2 pairs of entries per line. |
| `obj_sense` | 1 to write the objective as it stands, -1 to negate it so that a reader minimising the file maximises the original objective. |
| `force_r` | Use the R writer even when Clp provides its own. |

## Details

A row that is unbounded on both sides is written as a second N row. The format has no other way to say so, and readers, Clp's included, keep only the first N row as the objective and drop the rest.

## Value

The path written, invisibly.

## Examples

```
model <- clp_model()
clp_load_problem(model, 2, 1, c(0L, 1L, 2L), c(0L, 0L), c(1, 1),
                 obj = c(1, 2), rowub = 4)
path <- tempfile(fileext = ".mps")
```

```
clp_write_mps(model, path)
readLines(path)[1:3]
clp_free(model)
unlink(path)
```

---

clpPtr-accessors *Accessors for clpPtr objects*

---

## Description

Accessors for clpPtr objects

## Usage

```
clpPointer(object)

clpPtrType(object)

clpPtrType(object) <- value

isNULLpointerCLP(object)

isCLPpointer(object)
```

## Arguments

| | |
|---|---|
| object | A clpPtr object. |
| value | A replacement value. |

## Value

clpPointer() the external pointer, clpPtrType() a string, isNULLpointerCLP() and isCLPpointer() a logical.

## Examples

```
lp <- initProbCLP()
clpPtrType(lp)
delProbCLP(lp)
```

## clpPtr-class *A pointer to a Clp problem, clpAPI style*

### Description

An S4 class with the same shape as the `clpPtr` class of the archived **clpAPI** package, so that code written against that package keeps working. `initProbCLP` creates one.

### Slots

`clpPtrType` A string describing the pointer, `"clp_prob"`.

`clpPointer` The external pointer to the Clp model.

### Examples

```
lp <- initProbCLP()
isCLPpointer(lp)
delProbCLP(lp)
```

## copyNamesCLP *Names and files, clpAPI style*

### Description

Names and files, clpAPI style

### Usage

```
copyNamesCLP(lp, cnames, rnames)

dropNamesCLP(lp)

lengthNamesCLP(lp)

probNameCLP(lp, pname)

setRowNameCLP(lp, i, rname)

setColNameCLP(lp, j, cname)

printModelCLP(lp, prefix = "CLPmodel")

readMPSCLP(lp, fname, keepNames = TRUE, ignoreErrors = FALSE)

writeMPSCLP(lp, fname, formatType = 0, numberAcross = 1, objSense = 1)
```

```
saveModelCLP(lp, fname)

restoreModelCLP(lp, fname)

modifyCoefficientCLP(lp, i, j, el, keepZero = TRUE)

isAvailableFuncCLP(funcname)
```

## Arguments

| | |
|---|---|
| lp | A clpPtr object. |
| cnames, rnames | Character vectors of column and row names. |
| pname | A problem name. |
| i, j | 0-based row and column positions. |
| rname, cname | A single name. |
| prefix | A prefix for printModelCLP(). |
| fname | A file name. |
| keepNames | Keep the names found in an MPS file. |
| ignoreErrors | Carry on after errors in an MPS file. |
| formatType, numberAcross, objSense | Passed on to the MPS writer. |
| el | A new matrix coefficient. |
| keepZero | Keep zero entries in the sparse structure. |
| funcname | Name of an optional Clp entry point. |

## Value

Character or integer values as in **clpAPI**; the setters return NULL invisibly.

## Examples

```
lp <- initProbCLP()
loadMatrixCLP(lp, 2, 1, c(0, 1, 2), c(0, 0), c(1, 1))
copyNamesCLP(lp, c("x", "y"), "budget")
lengthNamesCLP(lp)
delProbCLP(lp)
```

getNumRowsCLP *Problem data, clpAPI style*

## Description

Problem data, clpAPI style

## Usage

```
getNumRowsCLP(lp)

getNumColsCLP(lp)

getNumNnzCLP(lp)

getObjCoefsCLP(lp)

chgObjCoefsCLP(lp, objCoef)

getColLowerCLP(lp)

chgColLowerCLP(lp, lb)

getColUpperCLP(lp)

chgColUpperCLP(lp, ub)

getRowLowerCLP(lp)

chgRowLowerCLP(lp, rlb)

getRowUpperCLP(lp)

chgRowUpperCLP(lp, rub)

getVecStartCLP(lp)

getVecLenCLP(lp)

getIndCLP(lp)

getNnzCLP(lp)
```

## Arguments

| | |
|---|---|
| lp | A clpPtr object. |
| objCoef | Objective coefficients. |

| | |
|---|---|
| lb, ub | Column bounds. |
| rlb, rub | Row bounds. |

## Value

The getters return numeric or integer vectors; the setters NULL invisibly.

## Examples

```
lp <- initProbCLP()
loadMatrixCLP(lp, 2, 0, c(0, 0, 0), integer(0), numeric(0))
chgObjCoefsCLP(lp, c(1, 2))
getObjCoefsCLP(lp)
delProbCLP(lp)
```

---

# getSolStatusCLP *Solutions and status, clpAPI style*

---

## Description

Solutions and status, clpAPI style

## Usage

```
getSolStatusCLP(lp)

getObjValCLP(lp)

getColPrimCLP(lp)

getColDualCLP(lp)

getRowPrimCLP(lp)

getRowDualCLP(lp)

status_codeCLP(code)

return_codeCLP(code)
```

## Arguments

| | |
|---|---|
| lp | A clpPtr object. |
| code | A status or return code. |

## Value

Numeric vectors for the solution accessors, an integer for getSolStatusCLP(), a string for the code descriptions.

## Examples

```
lp <- initProbCLP()
loadProblemCLP(lp, 1, 1, c(0, 1), 0, 1, obj_coef = -1, rlb = -1e30, rub = 2)
setLogLevelCLP(lp, 0)
solveInitialCLP(lp)
status_codeCLP(getSolStatusCLP(lp))
delProbCLP(lp)
```

---

`initProbCLP` *Create and delete a problem, clpAPI style*

---

## Description

As in `clp_model`, a new problem starts with Clp's message level at 0; `setLogLevelCLP()` turns the solver's reporting on.

## Usage

```
initProbCLP(ptrtype = "clp_prob")

delProbCLP(lp)
```

## Arguments

| | |
|---|---|
| `ptrtype` | A string stored in the returned object. |
| `lp` | A `clpPtr` object. |

## Value

`initProbCLP()` returns a `clpPtr`; `delProbCLP()` returns `NULL` invisibly.

## Examples

```
lp <- initProbCLP()
setLogLevelCLP(lp, 0)
delProbCLP(lp)
```

`loadProblemCLP` *Build a problem, clpAPI style*

## Description

Column-major, 0-based arguments, exactly as in **clpAPI**: `ia` holds the column starts, `ja` the row indices and `ra` the matrix entries.

## Usage

```
loadProblemCLP(
  lp,
  ncols,
  nrows,
  ia,
  ja,
  ra,
  lb = NULL,
  ub = NULL,
  obj_coef = NULL,
  rlb = NULL,
  rub = NULL
)

loadMatrixCLP(lp, ncols, nrows, ia, ja, ra)
```

## Arguments

| | |
|---|---|
| `lp` | A `clpPtr` object. |
| `ncols`, `nrows` | Problem dimensions. |
| `ia` | Integer vector of column starts, length `ncols + 1`. |
| `ja` | Integer vector of 0-based row indices. |
| `ra` | Numeric vector of matrix entries. |
| `lb`, `ub` | Column bounds, or `NULL`. |
| `obj_coef` | Objective coefficients, or `NULL`. |
| `rlb`, `rub` | Row bounds, or `NULL`. |

## Value

`NULL`, invisibly.

## Examples

```
lp <- initProbCLP()
loadProblemCLP(lp, 2, 1, c(0, 1, 2), c(0, 0), c(1, 1),
               lb = c(0, 0), ub = c(1e30, 1e30), obj_coef = c(-1, -2),
               rlb = -1e30, rub = 3)
solveInitialCLP(lp)
getObjValCLP(lp)
delProbCLP(lp)
```

---

`resizeCLP` *Change the size of a problem, clpAPI style*

---

## Description

Change the size of a problem, clpAPI style

## Usage

```
resizeCLP(lp, nrows, ncols)
```

## Arguments

| | |
|---|---|
| `lp` | A `clpPtr` object. |
| `nrows`, `ncols` | New dimensions. |

## Value

`NULL`, invisibly.

## Examples

```
lp <- initProbCLP()
resizeCLP(lp, 2, 2)
getNumRowsCLP(lp)
delProbCLP(lp)
```

setObjDirCLP *Solver settings, clpAPI style*

## Description

Solver settings, clpAPI style

## Usage

```
setObjDirCLP(lp, lpdir)

getObjDirCLP(lp)

setLogLevelCLP(lp, amount)

getLogLevelCLP(lp)

scaleModelCLP(lp, mode)

getScaleFlagCLP(lp)

setNumberIterationsCLP(lp, iterations)

setMaximumIterationsCLP(lp, iterations)

getMaximumIterationsCLP(lp)

getHitMaximumIterationsCLP(lp)

setMaximumSecondsCLP(lp, seconds)

getMaximumSecondsCLP(lp)
```

## Arguments

| | |
|---|---|
| lp | A clpPtr object. |
| lpdir | 1 to minimise, -1 to maximise. |
| amount | Log level, 0 to 4. |
| mode | Scaling mode. |
| iterations | Iteration count or limit. |
| seconds | Time limit in seconds. |

## Value

The getters return a number; the setters NULL invisibly.

## Examples

```
lp <- initProbCLP()
setObjDirCLP(lp, -1)
getObjDirCLP(lp)
delProbCLP(lp)
```

---

`solveInitialCLP` *Solve a problem, clpAPI style*

---

## Description

Solve a problem, clpAPI style

## Usage

```
solveInitialCLP(lp)

solveInitialDualCLP(lp)

solveInitialPrimalCLP(lp)

solveInitialBarrierCLP(lp)

solveInitialBarrierNoCrossCLP(lp)

primalCLP(lp, ifValP = 0)

dualCLP(lp, ifValP = 0)

idiotCLP(lp, thd = 0)
```

## Arguments

| | |
|---|---|
| `lp` | A `clpPtr` object. |
| `ifValP` | Pass 1 for a values pass, 0 otherwise. |
| `thd` | Effort level for the idiot crash. |

## Value

An integer return code from Clp; `idiotCLP()` returns `NULL` invisibly.

## Examples

```
lp <- initProbCLP()
loadProblemCLP(lp, 1, 1, c(0, 1), 0, 1, obj_coef = -1, rlb = -1e30, rub = 2)
setLogLevelCLP(lp, 0)
solveInitialCLP(lp)
getColPrimCLP(lp)
delProbCLP(lp)
```

---

# versionCLP *The Clp version, clpAPI style*

---

## Description

The Clp version, clpAPI style

## Usage

```
versionCLP()
```

## Value

The version of the Clp library as a string.

## Examples

```
versionCLP()
```

# Index


addColsCLP *(addRowsCLP)*, 3
addRowsCLP, 3

chgColLowerCLP *(getNumRowsCLP)*, 40
chgColUpperCLP *(getNumRowsCLP)*, 40
chgObjCoefsCLP *(getNumRowsCLP)*, 40
chgRowLowerCLP *(getNumRowsCLP)*, 40
chgRowUpperCLP *(getNumRowsCLP)*, 40
clp_add_columns *(clp_add_rows)*, 4
clp_add_rows, 4
clp_algorithm *(clp_log_level)*, 13
clp_check_solution *(clp_sum_primal_infeasibilities)*, 34
clp_col_lower *(clp_objective)*, 17
clp_col_name *(clp_row_names)*, 27
clp_col_names *(clp_row_names)*, 27
clp_col_solution, 5
clp_col_status *(clp_status_exists)*, 33
clp_col_upper *(clp_objective)*, 17
clp_control, 6, *31*
clp_copyin_status *(clp_status_exists)*, 33
clp_crash *(clp_initial_solve)*, 8
clp_delete_columns *(clp_add_rows)*, 4
clp_delete_integer *(clp_set_integer)*, 29
clp_delete_rows *(clp_add_rows)*, 4
clp_drop_names *(clp_row_names)*, 27
clp_dual_bound *(clp_primal_tolerance)*, 23
clp_dual_feasible *(clp_is_proven_optimal)*, 10
clp_dual_objective_limit *(clp_primal_tolerance)*, 23
clp_dual_simplex *(clp_initial_solve)*, 8
clp_dual_tolerance *(clp_primal_tolerance)*, 23
clp_elements *(clp_matrix)*, 14
clp_features, *28*, *36*
clp_features *(clp_version)*, 35
clp_free, 7, *15*
clp_hit_max_iterations *(clp_primal_tolerance)*, 23
clp_idiot *(clp_initial_solve)*, 8
clp_indices *(clp_matrix)*, 14
clp_inf, 8, *17*
clp_infeasibility_cost *(clp_primal_tolerance)*, 23
clp_infeasibility_ray *(clp_unbounded_ray)*, 34
clp_initial_barrier_no_cross_solve *(clp_initial_solve)*, 8
clp_initial_barrier_solve *(clp_initial_solve)*, 8
clp_initial_dual_solve *(clp_initial_solve)*, 8
clp_initial_primal_solve *(clp_initial_solve)*, 8
clp_initial_solve, 8
clp_initial_solve_with_options, *19*, *22*
clp_initial_solve_with_options *(clp_initial_solve)*, 8
clp_integer_information *(clp_set_integer)*, 29
clp_is_abandoned *(clp_is_proven_optimal)*, 10
clp_is_dual_objective_limit_reached *(clp_is_proven_optimal)*, 10
clp_is_iteration_limit_reached *(clp_is_proven_optimal)*, 10
clp_is_open, 9
clp_is_primal_objective_limit_reached *(clp_is_proven_optimal)*, 10
clp_is_proven_dual_infeasible *(clp_is_proven_optimal)*, 10
clp_is_proven_optimal, 10
clp_is_proven_primal_infeasible *(clp_is_proven_optimal)*, 10
clp_iterations *(clp_log_level)*, 13

clp_length_names (clp_row_names), 27
clp_load_problem, 11, *15*, *31*
clp_load_quadratic_objective, 12
clp_log_level, 13
clp_matrix, 14
clp_max_iterations (clp_primal_tolerance), 23
clp_max_seconds (clp_primal_tolerance), 23
clp_model, 15, *31*, *42*
clp_modify_coefficient, 15
clp_num_cols (clp_num_rows), 16
clp_num_dual_infeasibilities (clp_sum_primal_infeasibilities), 34
clp_num_elements (clp_num_rows), 16
clp_num_primal_infeasibilities (clp_sum_primal_infeasibilities), 34
clp_num_rows, 16
clp_obj_sense (clp_optimization_direction), 18
clp_objective, 17
clp_objective_offset (clp_objective_value), 18
clp_objective_value, 18
clp_optimization_direction, 18
clp_options, *9*, 19, *20*, *22*
clp_options_do_doubleton (clp_options_do_dual), 20
clp_options_do_dual, 20
clp_options_do_dupcol (clp_options_do_dual), 20
clp_options_do_duprow (clp_options_do_dual), 20
clp_options_do_forcing (clp_options_do_dual), 20
clp_options_do_implied_free (clp_options_do_dual), 20
clp_options_do_singleton (clp_options_do_dual), 20
clp_options_do_singleton_column (clp_options_do_dual), 20
clp_options_do_tighten (clp_options_do_dual), 20
clp_options_do_tripleton (clp_options_do_dual), 20
clp_options_free, 21
clp_options_get_extra_info (clp_options_set_solve_type), 22
clp_options_get_presolve_passes (clp_options_set_solve_type), 22
clp_options_get_presolve_type (clp_options_set_solve_type), 22
clp_options_get_solve_type (clp_options_set_solve_type), 22
clp_options_get_special_option (clp_options_set_solve_type), 22
clp_options_infeasible_return (clp_options_do_dual), 20
clp_options_presolve_actions (clp_options_do_dual), 20
clp_options_set_do_doubleton (clp_options_do_dual), 20
clp_options_set_do_dual (clp_options_do_dual), 20
clp_options_set_do_dupcol (clp_options_do_dual), 20
clp_options_set_do_duprow (clp_options_do_dual), 20
clp_options_set_do_forcing (clp_options_do_dual), 20
clp_options_set_do_implied_free (clp_options_do_dual), 20
clp_options_set_do_singleton (clp_options_do_dual), 20
clp_options_set_do_singleton_column (clp_options_do_dual), 20
clp_options_set_do_tighten (clp_options_do_dual), 20
clp_options_set_do_tripleton (clp_options_do_dual), 20
clp_options_set_infeasible_return (clp_options_do_dual), 20
clp_options_set_presolve_actions (clp_options_do_dual), 20
clp_options_set_presolve_type (clp_options_set_solve_type), 22
clp_options_set_solve_type, 22

clp_options_set_special_option
(clp_options_set_solve_type), 22
clp_options_set_substitution
(clp_options_do_dual), 20
clp_options_substitution
(clp_options_do_dual), 20
clp_perturbation (clp_log_level), 13
clp_primal_feasible
(clp_is_proven_optimal), 10
clp_primal_simplex (clp_initial_solve), 8
clp_primal_tolerance, 23
clp_print_model, 24
clp_problem_name, 25
clp_read_mps, 26
clp_reduced_costs (clp_col_solution), 5
clp_resize, 26
clp_restore_model (clp_save_model), 28
clp_row_activity (clp_col_solution), 5
clp_row_lower (clp_objective), 17
clp_row_name (clp_row_names), 27
clp_row_names, 27
clp_row_price (clp_col_solution), 5
clp_row_status (clp_status_exists), 33
clp_row_upper (clp_objective), 17
clp_save_model, 28
clp_scaling (clp_log_level), 13
clp_secondary_status (clp_status), 32
clp_set_algorithm (clp_log_level), 13
clp_set_col_lower (clp_objective), 17
clp_set_col_name (clp_row_names), 27
clp_set_col_solution, *9*
clp_set_col_solution
(clp_col_solution), 5
clp_set_col_status (clp_status_exists), 33
clp_set_col_upper (clp_objective), 17
clp_set_dual_bound
(clp_primal_tolerance), 23
clp_set_dual_objective_limit
(clp_primal_tolerance), 23
clp_set_dual_tolerance
(clp_primal_tolerance), 23
clp_set_infeasibility_cost
(clp_primal_tolerance), 23
clp_set_integer, 29
clp_set_iterations (clp_log_level), 13
clp_set_log_level, *15*
clp_set_log_level (clp_log_level), 13
clp_set_max_iterations
(clp_primal_tolerance), 23
clp_set_max_seconds
(clp_primal_tolerance), 23
clp_set_names (clp_row_names), 27
clp_set_obj_sense
(clp_optimization_direction), 18
clp_set_objective, *12*
clp_set_objective (clp_objective), 17
clp_set_objective_offset
(clp_objective_value), 18
clp_set_optimization_direction
(clp_optimization_direction), 18
clp_set_perturbation (clp_log_level), 13
clp_set_primal_tolerance
(clp_primal_tolerance), 23
clp_set_problem_name
(clp_problem_name), 25
clp_set_row_lower (clp_objective), 17
clp_set_row_name (clp_row_names), 27
clp_set_row_status (clp_status_exists), 33
clp_set_row_upper (clp_objective), 17
clp_set_scaling (clp_log_level), 13
clp_set_secondary_status (clp_status), 32
clp_set_small_element_value
(clp_primal_tolerance), 23
clp_set_status (clp_status), 32
clp_small_element_value
(clp_primal_tolerance), 23
clp_solve, *7*, *12*, *15*, 30
clp_status, *9*, *31*, 32
clp_status_array (clp_status_exists), 33
clp_status_exists, 33
clp_status_message (clp_status), 32
clp_sum_dual_infeasibilities
(clp_sum_primal_infeasibilities), 34
clp_sum_primal_infeasibilities, 34
clp_unbounded_ray, 34
clp_vector_lengths (clp_matrix), 14
clp_vector_starts (clp_matrix), 14
clp_version, 35

clp_write_mps, *29*, 36
clpPointer *(*clpPtr-accessors*)*, 37
clpPointer,clpPtr-method
*(*clpPtr-accessors*)*, 37
clpPtr, *3*, *37*, *39–46*
clpPtr-accessors, 37
clpPtr-class, 38
clpPtrType *(*clpPtr-accessors*)*, 37
clpPtrType,clpPtr-method
*(*clpPtr-accessors*)*, 37
clpPtrType<- *(*clpPtr-accessors*)*, 37
clpPtrType<-,clpPtr,character-method
*(*clpPtr-accessors*)*, 37
copyNamesCLP, 38

delColsCLP *(*addRowsCLP*)*, 3
delProbCLP *(*initProbCLP*)*, 42
delRowsCLP *(*addRowsCLP*)*, 3
dropNamesCLP *(*copyNamesCLP*)*, 38
dualCLP *(*solveInitialCLP*)*, 46

getColDualCLP *(*getSolStatusCLP*)*, 41
getColLowerCLP *(*getNumRowsCLP*)*, 40
getColPrimCLP *(*getSolStatusCLP*)*, 41
getColUpperCLP *(*getNumRowsCLP*)*, 40
getHitMaximumIterationsCLP
*(*setObjDirCLP*)*, 45
getIndCLP *(*getNumRowsCLP*)*, 40
getLogLevelCLP *(*setObjDirCLP*)*, 45
getMaximumIterationsCLP *(*setObjDirCLP*)*,
45
getMaximumSecondsCLP *(*setObjDirCLP*)*, 45
getNnzCLP *(*getNumRowsCLP*)*, 40
getNumColsCLP *(*getNumRowsCLP*)*, 40
getNumNnzCLP *(*getNumRowsCLP*)*, 40
getNumRowsCLP, 40
getObjCoefsCLP *(*getNumRowsCLP*)*, 40
getObjDirCLP *(*setObjDirCLP*)*, 45
getObjValCLP *(*getSolStatusCLP*)*, 41
getRowDualCLP *(*getSolStatusCLP*)*, 41
getRowLowerCLP *(*getNumRowsCLP*)*, 40
getRowPrimCLP *(*getSolStatusCLP*)*, 41
getRowUpperCLP *(*getNumRowsCLP*)*, 40
getScaleFlagCLP *(*setObjDirCLP*)*, 45
getSolStatusCLP, 41
getVecLenCLP *(*getNumRowsCLP*)*, 40
getVecStartCLP *(*getNumRowsCLP*)*, 40

idiotCLP *(*solveInitialCLP*)*, 46
initProbCLP, *38*, 42
isAvailableFuncCLP *(*copyNamesCLP*)*, 38
isCLPpointer *(*clpPtr-accessors*)*, 37
isCLPpointer,clpPtr-method
*(*clpPtr-accessors*)*, 37
isNULLpointerCLP *(*clpPtr-accessors*)*, 37
isNULLpointerCLP,clpPtr-method
*(*clpPtr-accessors*)*, 37

lengthNamesCLP *(*copyNamesCLP*)*, 38
loadMatrixCLP *(*loadProblemCLP*)*, 43
loadProblemCLP, 43

modifyCoefficientCLP *(*copyNamesCLP*)*, 38

primalCLP *(*solveInitialCLP*)*, 46
printModelCLP *(*copyNamesCLP*)*, 38
probNameCLP *(*copyNamesCLP*)*, 38

readMPSCLP *(*copyNamesCLP*)*, 38
resizeCLP, 44
restoreModelCLP *(*copyNamesCLP*)*, 38
return_codeCLP *(*getSolStatusCLP*)*, 41

saveModelCLP *(*copyNamesCLP*)*, 38
scaleModelCLP *(*setObjDirCLP*)*, 45
setColNameCLP *(*copyNamesCLP*)*, 38
setLogLevelCLP *(*setObjDirCLP*)*, 45
setMaximumIterationsCLP *(*setObjDirCLP*)*,
45
setMaximumSecondsCLP *(*setObjDirCLP*)*, 45
setNumberIterationsCLP *(*setObjDirCLP*)*,
45
setObjDirCLP, 45
setRowNameCLP *(*copyNamesCLP*)*, 38
show,clpPtr-method *(*clpPtr-accessors*)*,
37
solveInitialBarrierCLP
*(*solveInitialCLP*)*, 46
solveInitialBarrierNoCrossCLP
*(*solveInitialCLP*)*, 46
solveInitialCLP, 46
solveInitialDualCLP *(*solveInitialCLP*)*,
46
solveInitialPrimalCLP
*(*solveInitialCLP*)*, 46
status_codeCLP *(*getSolStatusCLP*)*, 41

versionCLP, 47

writeMPSCLP *(*copyNamesCLP*)*, 38