Package 'highs'

June 8, 2026

Type Package

Title 'HiGHS' Optimization Solver

Version 1.14.0-2

Description R interface to 'HiGHS', an optimization solver for solving mixed integer optimization problems with quadratic or linear objective and linear constraints.

License GPL (>= 2)

Imports Rcpp (>= 1.0.7), checkmate

Depends R (>= 4.0.0)

SystemRequirements Bash, PkgConfig, CMAKE (>=3.16), C++17

URL https://gitlab.com/roigrp/solver/highs

BugReports https://gitlab.com/roigrp/solver/highs/-/work_items/issues

Suggests tinytest, knitr, rmarkdown, Matrix

VignetteBuilder knitr

Biarch FALSE

LinkingTo Rcpp

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation yes

Author Florian Schwendinger [aut, cre],
Balasubramanian Narasimhan [aut],
Dirk Schumacher [aut],
Julian Hall [cph],
Ivet Galabova [cph],
Leona Gottwald [cph],
Michael Feldmeier [cph]

Maintainer Florian Schwendinger <FlorianSchwendinger@gmx.at>

Repository CRAN

Date/Publication 2026-06-08 07:20:02 UTC

Contents

`example_model` *Generate Example Optimization Models*

Description

Creates example optimization models for different problem types: - Linear Programming (LP) - Mixed Integer Linear Programming (MILP) - Quadratic Programming (QP)

Usage

```
example_model(op_type = c("LP", "MILP", "QP"))
```

Arguments

op_type	Character string specifying the type of optimization model. Must be one of "LP", "MILP", or "QP".

Value

A HiGHS model object configured according to the specified type: - LP: Maximization problem with 3 variables and 3 constraints - MILP: Maximization problem with mixed integer and continuous variables - QP: Problem with quadratic objective function

Examples

```
model <- example_model("LP")
model <- example_model("MILP")
model <- example_model("QP")
```

---

example_solver *Create a HiGHS Solver Object*

Description

Creates and solves an example optimization model using the HiGHS solver. This is a convenience wrapper that combines model creation and solving in a single function call.

Usage

```
example_solver(op_type = c("LP", "MILP", "QP"))
```

Arguments

op_type	Character string specifying the type of optimization model. Must be one of "LP", "MILP", or "QP".

Value

An object of class "highs_solver".

Examples

```
solver <- example_solver("LP")
solver <- example_solver("MILP")
solver <- example_solver("QP")
```

highs_available_solver_options *Available Solver Options*

Description

Reference for the available solver options.

Usage

```
highs_available_solver_options()
```

Value

A `data.frame` containing the available solver options.

Examples

```
highs_available_solver_options()
```

highs_control *Highs Control*

Description

Highs Control

Usage

```
highs_control(threads = 1L, time_limit = Inf, log_to_console = FALSE, ...)
```

Arguments

`threads`	an integer giving the number of threads to be used.
`time_limit`	a double giving the time limit.
`log_to_console`	a logical giving if the output should be shown in the console.
`...`	other arguments supported by the `HiGHS` solver.

Examples

```
control <- highs_control()
```

highs_model *Create a Highs Model*

Description

Solve linear and quadratic mixed integer optimization problems.

Usage

```
highs_model(
  Q = NULL,
  L,
  lower,
  upper,
  A = NULL,
  lhs = NULL,
  rhs = NULL,
  types = rep.int(1L, length(L)),
  maximum = FALSE,
  offset = 0
)
```

Arguments

Q	a numeric symmetric matrix giving the quadratic part of the objective.
L	a numeric vector giving the linear part of the objective function.
lower	a numeric vector giving the lower bounds of the variables.
upper	a numeric vector giving the upper bounds of the variables.
A	a numeric matrix giving the linear part of the constraints. Rows are constraints, and columns are decision variables.
lhs	a numeric vector giving the left hand-side of the linear constraints.
rhs	a numeric vector giving the right hand-side of the linear constraints.
types	a integer vector or character vector giving the variable types. 'C' or '1' for continuous, 'I' or '2' for integer, 'SC' or '3' for semi continuous, 'SI' or '4' for semi integer and 'II' or '5' for implicit integer.
maximum	a logical if TRUE the solver searches for a maximum, if FALSE the solver searches for a minimum.
offset	a numeric value giving the offset (default is 0).

Value

A an object of class highs_model.

Examples

```
library("highs")
# Minimize:
#  x_0 +  x_1 + 3
# Subject to:
#               x_1 <=  7
#  5 <=  x_0 + 2x_1 <= 15
#  6 <= 3x_0 + 2x_1
#  0 <= x_0 <= 4
#  1 <= x_1
A <- rbind(c(0, 1), c(1, 2), c(3, 2))
m <- highs_model(L = c(1.0, 1), lower = c(0, 1), upper = c(4, Inf),
                 A = A, lhs = c(-Inf, 5, 6), rhs = c(7, 15, Inf),
                 offset = 3)
m

# Minimize:
#  -x_2 - 3x_3 + (1/2) * (2 x_1^2 - 2 x_1x_3 + 0.2 x_2^2 + 2 x_3^2)
# Subject to:
#  x_1 + x_3 <= 2
#  0 <= x
L <- c(0, -1, -3)
Q <- rbind(c(2, 0.0, -1), c(0, 0.2, 0), c(-1, 0.0, 2))
A <- cbind(1, 0, 1)
m <- highs_model(Q = Q, L = L, lower = 0, A = A, rhs = 2)
m
```

`highs_solve` *Solve an Optimization Problems*

Description

Solve linear and quadratic mixed integer optimization problems.

Usage

```
highs_solve(
  Q = NULL,
  L,
  lower,
  upper,
  A = NULL,
  lhs = NULL,
  rhs = NULL,
  types = rep.int(1L, length(L)),
  maximum = FALSE,
  offset = 0,
  start = NULL,
  control = highs_control()
)
```

Arguments

Q	a numeric symmetric matrix giving the quadratic part of the objective.
L	a numeric vector giving the linear part of the objective function.
lower	a numeric vector giving the lower bounds of the variables.
upper	a numeric vector giving the upper bounds of the variables.
A	a numeric matrix giving the linear part of the constraints. Rows are constraints, and columns are decision variables.
lhs	a numeric vector giving the left hand-side of the linear constraints.
rhs	a numeric vector giving the right hand-side of the linear constraints.
types	a integer vector or character vector giving the variable types. 'C' or '1' for continuous, 'I' or '2' for integer, 'SC' or '3' for semi continuous, 'SI' or '4' for semi integer and 'II' or '5' for implicit integer.
maximum	a logical if TRUE the solver searches for a maximum, if FALSE the solver searches for a minimum.
offset	a numeric value giving the offset (default is 0).
start	a numeric vector giving the start solution. Missing values can be used to specify that particular variables should have their starting values calculated automatically.
control	a list giving additional options for the solver, see highs_available_solver_options or the README file for a list of all available options.

Value

A list containing the result provided by the solver, containing the following named objects:

primal_solution	a numeric vector giving the primal solution.
objective_value	a numeric giving the objective value.
status	an integer giving the status code
status_message	a character string giving the status message (explanation of the status_code).
solver_msg	a list giving the original (not canonicalized) solver message.
info	a list giving additional information provided by the solver.

Additional information on can be found in the README file.

Examples

```
library("highs")
# Minimize:
#  x_0 +  x_1 + 3
# Subject to:
#               x_1 <=  7
#  5 <=  x_0 + 2x_1 <= 15
#  6 <= 3x_0 + 2x_1
```

```
#  0 <= x_0 <= 4
#  1 <= x_1
A <- rbind(c(0, 1), c(1, 2), c(3, 2))
s <- highs_solve(L = c(1.0, 1), lower = c(0, 1), upper = c(4, Inf),
                 A = A, lhs = c(-Inf, 5, 6), rhs = c(7, 15, Inf),
                 offset = 3)
s[["objective_value"]]
s[["primal_solution"]]

# Minimize:
#  -x_2 - 3x_3 + (1/2) * (2 x_1^2 - 2 x_1x_3 + 0.2 x_2^2 + 2 x_3^2)
# Subject to:
#  x_1 + x_3 <= 2
#  0 <= x
L <- c(0, -1, -3)
Q <- rbind(c(2, 0.0, -1), c(0, 0.2, 0), c(-1, 0.0, 2))
A <- cbind(1, 0, 1)
s <- highs_solve(Q = Q, L = L, lower = 0, A = A, rhs = 2)
s[["objective_value"]]
s[["primal_solution"]]
```

`highs_solver` *Highs Solver*

Description

Create a wrapper around the `HiGHS` solver. Manly usefull if one wants a low level wrapper around highs with hot-start capabilities.

Usage

```
highs_solver(model, control = highs_control())
```

Arguments

`model`	an object of class `"highs_model"` created with `highs_model()`.
`control`	an object of class `"highs_control"` created with `highs_control()`.

Details

Methods

The following methods are provided by the `"highs_solver"` class.

- `solve(...)` method to be called to solve the optimization problem. Returns an integer giving the status code returned by **HiGHS**.
- `status()` method to obtain the status from the solver.
- `status_message()` method to obtain the status message from the solver.

- `solution()` method to obtain the solution from the solver.
- `info()` info to obtain addtional information from the solver.
- `L(i, v)` method to get and set the linear part of the objective.
- `A(i, j, v)` method to get and set the constraint matrix coefficients.
- `cbounds(i, lhs, rhs)` method to get and set the constraint bounds (left hand-side and right hand-side).
- `types(i, v)` method to get and set the variable types.
- `vbounds(i, lower, upper)` method to get and set the variable bounds.
- `maximum(maximize)` method to get and set the sense of the problem.

Method arguments

- `...` optional control arguments, which can be used to alter the options set via the `control` argument when initializing the solver.
- `i` a vector of integers giving the index (vector index or row index) of the coeficcients to be altered.
- `j` a vector of integers giving the index (column index) of the coeficcients to be altered.
- `v` a vector of doubles giving the values of the coeficcients to be altered.
- `lhs` a vector of doubles giving left hand-side.
- `rhs` a vector of doubles giving right hand-side.
- `lower` a vector of doubles giving the lower bounds to be altered.
- `upper` a vector of doubles giving the upper bounds to be altered.

Value

an object of class `"highs_solver"`.

Examples

```
A <- rbind(c(0, 1), c(1, 2), c(3, 2))
m <- highs_model(L = c(1.0, 1), lower = c(0, 1), upper = c(4, Inf),
                 A = A, lhs = c(-Inf, 5, 6), rhs = c(7, 15, Inf),
                 offset = 3)
solver <- highs_solver(m)
```

highs_write_model *Write a Highs Model to a File*

Description

Write an highs model to file.

Usage

```
highs_write_model(model, file)
```

Arguments

model	an object of class highs_model.
file	a character string giving the filename.

Examples

```
model <- example_model()
model_file <- tempfile(fileext = ".mps")
highs_write_model(model, model_file)
```

hi_model_get_ncons *Get Number of Constraints in a Model*

Description

This function retrieves the number of constraints in a given ‘highs_model‘ object.

Usage

```
hi_model_get_ncons(model)
```

Arguments

model	A ‘highs_model‘ object. The model from which to get the number of variables.

Value

An integer representing the number of constraints in the model.

Examples

```
model <- hi_new_model()
hi_model_get_ncons(model)
```

hi_model_get_nvars *Get Number of Variables in a Highs Model*

Description

This function retrieves the number of variables in a given Highs model.

Usage

```
hi_model_get_nvars(model)
```

Arguments

model	A 'highs_model' object. The model from which to get the number of variables.

Value

An integer representing the number of variables in the model.

Examples

```
model <- hi_new_model()
hi_model_get_nvars(model)
```

hi_model_set_constraint_matrix *Set Constraint Matrix for Highs Model*

Description

This function sets the constraint matrix for a given Highs model.

Usage

```
hi_model_set_constraint_matrix(model, matrix)
```

Arguments

model	an object of class "highs_model".
matrix	a matrix giving the Hessian matrix. Allowed matrix classes are "matrix", "dgCMatrix", "matrix.csc", and "simple_triplet_matrix".

Value

NULL

Examples

```
model <- hi_new_model()
matrix <- matrix(c(1, 0, 0, 1), nrow = 2)
hi_model_set_constraint_matrix(model, matrix)
```

---

hi_model_set_hessian *Set Hessian Matrix for Highs Model*

Description

This function sets the Hessian matrix for a given Highs model.

Usage

```
hi_model_set_hessian(model, matrix)
```

Arguments

model	an object of class "highs_model".
matrix	a matrix giving the Hessian matrix. Allowed matrix classes are "matrix", "dgCMatrix", "matrix.csc", and "simple_triplet_matrix".

Value

NULL

Examples

```
model <- hi_new_model()
hessian_matrix <- matrix(c(1, 0, 0, 1), nrow = 2)
hi_model_set_hessian(model, hessian_matrix)
```

---

hi_model_set_lhs *Set Left Hand Side for a Highs Model*

Description

This function sets the left hand side for a given Highs model.

Usage

```
hi_model_set_lhs(model, lhs)
```

Arguments

model	an object of class "highs_model".
lhs	a numeric vector giving the left hand side values.

Value

NULL

Examples

```
model <- hi_new_model()
model <- hi_model_set_lhs(model, c(0, 1, 2))
```

---

hi_model_set_lower *Set Lower Bounds for Highs Model*

Description

This function sets the lower bounds for a given Highs model.

Usage

```
hi_model_set_lower(model, lower)
```

Arguments

model	an object of class "highs_model".
lower	a numeric vector giving the lower bounds.

Value

NULL

Examples

```
model <- hi_new_model()
lower_bounds <- c(0, 1, 2)
hi_model_set_lower(model, lower_bounds)
```

hi_model_set_ncol *Sets the number of columns in the model*

Description

This function sets the number of columns in the given model.

Usage

```
hi_model_set_ncol(model, ncol)
```

Arguments

model	an object of class "highs_model".
ncol	an integer giving the number of columns (variables) to set in the model

Value

NULL

Examples

```
model <- hi_new_model()
hi_model_set_ncol(model, 10L) # Sets the model to have 10 columns
```

hi_model_set_nrow *Set the number of rows in the model*

Description

This function sets the number of rows in the given model.

Usage

```
hi_model_set_nrow(model, nrow)
```

Arguments

model	an object of class "highs_model".
nrow	an integer giving the number of rows (variables) to set in the model

Value

NULL

Examples

```
model <- hi_new_model()
hi_model_set_nrow(model, 5L) # Sets the model to have 5 rows
```

`hi_model_set_objective` *Set Objective for Highs Model*

Description

This function sets the objective for a given Highs model.

Usage

```
hi_model_set_objective(model, objective)
```

Arguments

`model`	an object of class `"highs_model"`.
`objective`	a numeric vector giving the objective values to be set for the model.

Value

`NULL`

Examples

```
model <- hi_new_model()
objective <- c(1, 2, 3)
hi_model_set_objective(model, objective)
```

`hi_model_set_offset` *Set Offset for Highs Model*

Description

This function sets the offset for a given Highs model.

Usage

```
hi_model_set_offset(model, offset)
```

Arguments

`model`	an object of class `"highs_model"`.
`offset`	a numeric value of length 1. The offset value to be set for the model.

Value

NULL

Examples

```
model <- hi_new_model()
hi_model_set_offset(model, 10)
```

`hi_model_set_rhs` *Set Right Hand Side for a Highs Model*

Description

This function sets the left hand side for a given Highs model.

Usage

```
hi_model_set_rhs(model, rhs)
```

Arguments

`model`	an object of class "`highs_model`".
`rhs`	a numeric vector giving the left hand side values.

Value

NULL

Examples

```
model <- hi_new_model()
model <- hi_model_set_rhs(model, c(0, 1, 2))
```

hi_model_set_sense *Set the sense of the optimization model*

Description

This function sets the sense of the optimization model to either maximization or minimization.

Usage

```
hi_model_set_sense(model, maximum)
```

Arguments

model	an object of class "highs_model".
maximum	a boolean value indicating whether the model should be set to maximization ('TRUE') or minimization ('FALSE').

Value

NULL

Examples

```
model <- hi_new_model()
hi_model_set_sense(model, TRUE) # Set the model to maximization
hi_model_set_sense(model, FALSE) # Set the model to minimization
```

hi_model_set_upper *Set Upper Bounds for a Highs Model*

Description

This function sets the upper bounds for a given Highs model.

Usage

```
hi_model_set_upper(model, upper)
```

Arguments

model	an object of class "highs_model".
upper	a numeric vector giving the upper bounds.

Value

NULL

Examples

```
model <- hi_new_model()
upper_bounds <- c(10, 20, 30)
hi_model_set_upper(model, upper_bounds)
```

---

hi_model_set_vartype *Set Variable Types in a Highs Model*

Description

This function sets the variable types in a given Highs model.

Usage

```
hi_model_set_vartype(model, types)
```

Arguments

model	an object of class "highs_model".
types	an integer vector specifying the types of the variables.

Value

The function does not return a value. It modifies the ‘model‘ object in place.

Examples

```
model <- hi_new_model()
types <- c(1, 2, 1, 0)
hi_model_set_vartype(model, types)
```

---

hi_new_model *Create new Highs Model*

Description

Create a new highs model object.

Usage

```
hi_new_model()
```

Value

an object of class "highs_model".

Examples

```
model <- hi_new_model()
```

`hi_new_solver` *Create a new solver instance.*

Description

This function initializes a new Highs solver instance using the provided model pointer.

Usage

```
hi_new_solver(model)
```

Arguments

`model`	an object of class `"highs_model"`

Value

A new solver instance.

Examples

```
model <- example_model()
solver <- hi_new_solver(model)
```

`hi_reset_global_scheduler` *Reset Global Scheduler*

Description

This function resets the global scheduler used by the solver.

Usage

```
hi_reset_global_scheduler(blocking)
```

Arguments

`blocking`	A logical value indicating whether to wait for completion.

Value

Invisible NULL.

Examples

```
hi_reset_global_scheduler(TRUE)
```

`hi_solver_add_cols` *Add Variables to Model*

Description

This function adds new variables (columns) to the optimization model.

Usage

```
hi_solver_add_cols(solver, costs, lower, upper, start, index, value)
```

Arguments

`solver`	An object of class "highs_solver".
`costs`	A numeric vector of objective coefficients.
`lower`	A numeric vector giving the lower bounds of the new variables.
`upper`	A numeric vector giving the upper bounds of the new variables.
`start`	An integer vector of starting positions in the sparse matrix.
`index`	An integer vector of row indices.
`value`	A numeric vector of coefficient values.

Value

The solver instance with new variables added.

Examples

```
solver <- example_solver()
hi_solver_add_cols(solver, c(1), c(0), c(10), c(0, 1), c(0), c(2))
```

hi_solver_add_rows *Add Constraints to Model*

Description

This function adds new constraints (rows) to the optimization model.

Usage

```
hi_solver_add_rows(solver, lhs, rhs, start, index, value)
```

Arguments

solver	An object of class "highs_solver".
lhs	A numeric vector of left-hand side bounds.
rhs	A numeric vector of right-hand side bounds.
start	An integer vector of starting positions in the sparse matrix.
index	An integer vector of column indices.
value	A numeric vector of coefficient values.

Value

The solver instance with new constraints added.

Examples

```
solver <- example_solver()
hi_solver_add_rows(solver, c(-Inf), c(10), c(0, 2), c(0, 1), c(1, 2))
```

hi_solver_add_vars *Add Variables to the Solver*

Description

This function adds new variables to the solver with specified bounds.

Usage

```
hi_solver_add_vars(solver, lower, upper)
```

Arguments

solver	An object of class "highs_solver".
lower	A numeric vector of lower bounds for the new variables.
upper	A numeric vector of upper bounds for the new variables.

Value

The solver instance with the new variables added.

Examples

```
solver <- example_solver()
hi_solver_add_vars(solver, lower = c(0, 0, 0), upper = c(10, 10, 10))
```

`hi_solver_change_constraint_bounds`
Change Constraint Bounds

Description

This function updates the bounds of an existing constraint in the model.

Usage

```
hi_solver_change_constraint_bounds(solver, idx, lhs, rhs)
```

Arguments

`solver`	An object of class "highs_solver".
`idx`	An integer vector specifying the constraint indices. Note that idx starts at 0.
`lhs`	The new left-hand side bound.
`rhs`	The new right-hand side bound.

Details

This is a low-level wrapper for ‘highs.changeRowsBounds‘ that only does basic checks to prevent segfaults, but otherwise behaves exactly like ‘highs.changeRowsBounds‘.

Value

The solver instance with updated constraint bounds.

Examples

```
solver <- example_solver()
hi_solver_change_constraint_bounds(solver, 1, -Inf, 100)
```

hi_solver_change_variable_bounds *Change Variable Bounds*

Description

This function updates the bounds of an existing variable in the model.

Usage

```
hi_solver_change_variable_bounds(solver, idx, lower, upper)
```

Arguments

solver	An object of class "highs_solver".
idx	An integer specifying the variable index.
lower	The new lower bound.
upper	The new upper bound.

Value

The solver instance with updated bounds.

Examples

```
solver <- example_solver()
hi_solver_change_variable_bounds(solver, 1, 0, 10)
```

hi_solver_clear *Clear All Solver Data*

Description

This function clears all data from the solver instance, including the model and solution.

Usage

```
hi_solver_clear(solver)
```

Arguments

solver	An object of class "highs_solver".

Value

The cleared solver instance.

Examples

```
solver <- example_solver()
hi_solver_clear(solver)
```

---

hi_solver_clear_basis *Clear Basis*

Description

Invalidates the solver's current basis, allowing presolve to run on the next call to `hi_solver_run`.

Usage

```
hi_solver_clear_basis(solver)
```

Arguments

`solver`	An object of class "highs_solver".

Value

An integer status code (0 = OK).

---

hi_solver_clear_model *Clear Model Data*

Description

This function clears only the model data from the solver instance.

Usage

```
hi_solver_clear_model(solver)
```

Arguments

`solver`	An object of class "highs_solver".

Value

The solver instance with cleared model data.

Examples

```
solver <- example_solver()
hi_solver_clear_model(solver)
```

`hi_solver_clear_solver` *Clear Solver State*

Description

This function clears the internal solver state while preserving the model.

Usage

```
hi_solver_clear_solver(solver)
```

Arguments

`solver`	An object of class "highs_solver".

Value

The solver instance with cleared solver state.

Examples

```
solver <- example_solver()
hi_solver_clear_solver(solver)
```

`hi_solver_get_basis` *Get Basis*

Description

Retrieves the current simplex basis from the solver.

Usage

```
hi_solver_get_basis(solver)
```

Arguments

`solver`	An object of class "highs_solver".

Value

A list with components:

valid Logical, whether the basis is valid.

col_status Integer vector of column basis statuses (0=Lower, 1=Basic, 2=Upper, 3=Zero, 4=Non-basic).

row_status Integer vector of row basis statuses.

---

hi_solver_get_bool_option

Get Boolean Option Value

Description

This function retrieves the value of a boolean solver option.

Usage

```
hi_solver_get_bool_option(solver, key)
```

Arguments

solver	An object of class "highs_solver".
key	A character string specifying the option name.

Value

A logical value.

Examples

```
solver <- example_solver()
value <- hi_solver_get_bool_option(solver, "mip_detect_symmetry")
```

hi_solver_get_constraint_bounds
Get Constraint Bounds

Description

This function retrieves the bounds for all constraints in the model.

Usage

```
hi_solver_get_constraint_bounds(solver)
```

Arguments

solver	An object of class "highs_solver".

Value

A list containing lower and upper bounds for all constraints.

Examples

```
solver <- example_solver()
bounds <- hi_solver_get_constraint_bounds(solver)
```

hi_solver_get_constraint_matrix
Get Constraint Matrix

Description

This function retrieves the constraint matrix of the optimization model.

Usage

```
hi_solver_get_constraint_matrix(solver)
```

Arguments

solver	An object of class "highs_solver".

Value

A sparse matrix representing the constraints.

Examples

```
solver <- example_solver()
matrix <- hi_solver_get_constraint_matrix(solver)
```

`hi_solver_get_dbl_option` *Get Double Option Value*

Description

This function retrieves the value of a double precision solver option.

Usage

```
hi_solver_get_dbl_option(solver, key)
```

Arguments

`solver`	An object of class "highs_solver".
`key`	A character string specifying the option name.

Value

A numeric value.

Examples

```
solver <- example_solver()
value <- hi_solver_get_dbl_option(solver, "time_limit")
```

`hi_solver_get_dual_ray` *Get the Dual Ray (Farkas Infeasibility Certificate)*

Description

For a primal-infeasible LP, returns the dual unbounded ray: a Farkas certificate of primal infeasibility, with one entry per constraint (row).

Usage

```
hi_solver_get_dual_ray(solver)
```

Arguments

solver	An object of class "highs_solver".

Details

A dual ray is available only after an LP detected to be **infeasible** and solved by the **simplex** method (the interior-point solver does not produce a ray). Disable presolve, or rely on HiGHS's postsolve, to recover the ray on the original model. Requesting the ray may solve an auxiliary LP.

Value

A list with 'status' (integer, 0 = OK), 'has_dual_ray' (logical), and 'dual_ray' (numeric vector of length 'n_row', or 'NULL' when no ray exists).

---

hi_solver_get_int_option

Get Integer Option Value

Description

This function retrieves the value of an integer solver option.

Usage

```
hi_solver_get_int_option(solver, key)
```

Arguments

solver	An object of class "highs_solver".
key	A character string specifying the option name.

Value

An integer value.

Examples

```
solver <- example_solver()
value <- hi_solver_get_int_option(solver, "log_dev_level")
```

hi_solver_get_lp_costs *Get Objective Coefficients*

Description

This function retrieves the objective coefficients of the linear program.

Usage

```
hi_solver_get_lp_costs(solver)
```

Arguments

solver	An object of class "highs_solver".

Value

A numeric vector of objective coefficients.

Examples

```
solver <- example_solver()
costs <- hi_solver_get_lp_costs(solver)
```

hi_solver_get_num_col *Get Number of Variables*

Description

This function returns the number of variables (columns) in the optimization model.

Usage

```
hi_solver_get_num_col(solver)
```

Arguments

solver	An object of class "highs_solver".

Value

An integer representing the number of variables.

Examples

```
solver <- example_solver()
n_vars <- hi_solver_get_num_col(solver)
```

`hi_solver_get_num_row` *Get Number of Constraints*

Description

This function returns the number of constraints (rows) in the optimization model.

Usage

```
hi_solver_get_num_row(solver)
```

Arguments

`solver`	An object of class "highs_solver".

Value

An integer representing the number of constraints.

Examples

```
solver <- example_solver()
n_constraints <- hi_solver_get_num_row(solver)
```

`hi_solver_get_option` *Get a HiGHS Solver Option*

Description

Retrieves the value of a specific option from a HiGHS solver instance.

Usage

```
hi_solver_get_option(
  solver,
  key,
  type = c("auto", "bool", "integer", "double", "string")
)
```

Arguments

solver	A HiGHS solver object of class "highs_solver".
key	A character string specifying the option name to retrieve.
type	Type of the option. Can be one of "auto", "bool", "integer", "double", or "string". When set to "auto" (default), the function will attempt to determine the type from the available options list. Specify a type directly if the option is valid but not listed in the available options.

Value

The value of the specified option with the appropriate type.

Examples

```
solver <- example_solver()
hi_solver_get_option(solver, "output_flag")
hi_solver_get_option(solver, "solver", type = "string")
```

---

hi_solver_get_options *Get multiple HiGHS Solver Options*

Description

Retrieves the values of multiple options from a HiGHS solver instance.

Usage

```
hi_solver_get_options(solver, keys = NULL)
```

Arguments

solver	A HiGHS solver object of class "highs_solver".
keys	A character vector of option names to retrieve.

Value

A named list of option values with the appropriate types.

Examples

```
solver <- example_solver()
hi_solver_get_options(solver, c("output_flag", "solver"))
```

hi_solver_get_primal_ray *Get the Primal Ray (Unboundedness Certificate)*

Description

For a primal-unbounded LP (i.e. dual-infeasible), returns the primal unbounded ray: a certificate of unboundedness, with one entry per variable (column).

Usage

```
hi_solver_get_primal_ray(solver)
```

Arguments

solver	An object of class "highs_solver".

Details

As with the dual ray, this is a simplex-method certificate; requesting it may solve an auxiliary LP.

Value

A list with ‘status‘ (integer, 0 = OK), ‘has_primal_ray‘ (logical), and ‘primal_ray‘ (numeric vector of length ‘n_col‘, or ‘NULL‘ when no ray exists).

hi_solver_get_ranging *Get Ranging (Sensitivity Analysis)*

Description

Computes ranging information for the solved LP. The solver must have an optimal LP solution before calling this function.

Usage

```
hi_solver_get_ranging(solver)
```

Arguments

solver	An object of class "highs_solver".

Value

A list with components:

valid Logical, whether the ranging data is valid.

col_cost_up, col_cost_dn Lists with value, objective, in_var, ou_var vectors for cost ranging (length num_col).

col_bound_up, col_bound_dn Lists for column bound ranging.

row_bound_up, row_bound_dn Lists for row bound ranging (length num_row).

---

hi_solver_get_run_time *Get Run Time*

Description

Returns the elapsed time of the last solver run.

Usage

```
hi_solver_get_run_time(solver)
```

Arguments

solver	An object of class "highs_solver".

Value

A numeric value (seconds).

---

hi_solver_get_sense *Get the optimization sense of the solver instance.*

Description

This function returns the optimization sense (e.g., minimization or maximization) of the provided solver instance.

Usage

```
hi_solver_get_sense(solver)
```

Arguments

solver	An object of class "highs_solver" representing the solver instance.

Value

The optimization sense of the solver instance.

Examples

```
solver <- example_solver()
hi_solver_get_sense(solver)
```

`hi_solver_get_solution` *Get Solution*

Description

This function retrieves the solution from the solver after optimization.

Usage

```
hi_solver_get_solution(solver)
```

Arguments

`solver`	An object of class "highs_solver".

Value

A list containing the solution information.

Examples

```
solver <- example_solver()
hi_solver_run(solver)
solution <- hi_solver_get_solution(solver)
```

hi_solver_get_str_option
Get String Option Value

Description

This function retrieves the value of a string solver option.

Usage

```
hi_solver_get_str_option(solver, key)
```

Arguments

solver	An object of class "highs_solver".
key	A character string specifying the option name.

Value

A character string.

Examples

```
solver <- example_solver()
value <- hi_solver_get_str_option(solver, "solver")
```

hi_solver_get_variable_bounds
Get Variable Bounds

Description

This function retrieves the bounds for all variables in the model.

Usage

```
hi_solver_get_variable_bounds(solver)
```

Arguments

solver	An object of class "highs_solver".

Value

A list containing lower and upper bounds for all variables.

Examples

```
solver <- example_solver()
bounds <- hi_solver_get_variable_bounds(solver)
```

`hi_solver_get_vartype` *Get Variable Types*

Description

This function retrieves the type (continuous, integer, etc.) of all variables.

Usage

```
hi_solver_get_vartype(solver)
```

Arguments

`solver`	An object of class "highs_solver".

Value

A character vector of variable types.

Examples

```
solver <- example_solver()
types <- hi_solver_get_vartype(solver)
```

`hi_solver_infinity` *Get Solver Infinity Value*

Description

This function returns the value that the solver uses to represent infinity.

Usage

```
hi_solver_infinity()
```

Value

A numeric value representing infinity in the solver.

Examples

```
inf <- hi_solver_infinity()
```

hi_solver_info *Get Solver Information*

Description

This function retrieves detailed information about the solver's state and performance.

Usage

```
hi_solver_info(solver)
```

Arguments

solver	An object of class "highs_solver".

Value

A list containing solver information.

Examples

```
solver <- example_solver()
info <- hi_solver_info(solver)
```

hi_solver_postsolve *Postsolve*

Description

Performs postsolve using a given solution from the presolved model.

Usage

```
hi_solver_postsolve(solver, col_value, col_dual, row_value, row_dual)
```

Arguments

solver	An object of class "highs_solver".
col_value	Numeric vector of column values.
col_dual	Numeric vector of column duals.
row_value	Numeric vector of row values.
row_dual	Numeric vector of row duals.

Value

An integer status code (0 = OK).

hi_solver_presolve *Presolve*

Description

Runs presolve on the current model without solving.

Usage

```
hi_solver_presolve(solver)
```

Arguments

solver	An object of class "highs_solver".

Value

An integer status code (0 = OK).

hi_solver_read_model *Read Model from File*

Description

Reads a model from a file (MPS or LP format).

Usage

```
hi_solver_read_model(solver, filename)
```

Arguments

solver	An object of class "highs_solver".
filename	Path to the model file.

Value

An integer status code (0 = OK).

hi_solver_run *Run the Solver*

Description

Executes the optimization solver on the current model.

Usage

```
hi_solver_run(solver, verbose = FALSE)
```

Arguments

solver	An object of class "highs_solver".
verbose	a logical if TRUE, prints solver log messages to the console.

Details

Solver output is controlled by the ‘output_flag‘ option: use ‘hi_solver_set_option(solver, "output_flag", "off")‘ to suppress logging.

Value

An integer status code (-1 = error, 0 = success, 1 = warning).

Examples

```
solver <- example_solver()
hi_solver_run(solver)
```

hi_solver_set_basis *Set Basis for Warm-Start*

Description

Sets a simplex basis on the solver for warm-starting. Use basis status values: 0=Lower, 1=Basic, 2=Upper, 3=Zero, 4=Nonbasic.

Usage

```
hi_solver_set_basis(solver, col_status, row_status)
```

Arguments

solver	An object of class "highs_solver".
col_status	Integer vector of column basis statuses.
row_status	Integer vector of row basis statuses.

Value

An integer status code (0 = OK).

---

hi_solver_set_coeff *Set a coefficient in the constraint matrix.*

Description

This function assigns a coefficient value to a specific entry in the constraint matrix.

Usage

```
hi_solver_set_coeff(solver, row, col, val)
```

Arguments

solver	An object of class "highs_solver".
row	The row index.
col	The column index.
val	The coefficient value.

Value

The solver instance with the updated coefficient.

Examples

```
solver <- example_solver()
hi_solver_set_coeff(solver, 1, 1, 4.2)
```

hi_solver_set_constraint_bounds

Set constraint bounds for a given constraint.

Description

This function sets the lower and upper bounds for a specific constraint.

Usage

```
hi_solver_set_constraint_bounds(solver, index, lower, upper)
```

Arguments

solver	An object of class "highs_solver".
index	The constraint index.
lower	The lower bound.
upper	The upper bound.

Value

The solver instance with updated constraint bounds.

Examples

```
solver <- example_solver()
hi_solver_set_constraint_bounds(solver, 1, -Inf, 100)
```

hi_solver_set_integrality

Set integrality for a set of variables in the solver.

Description

This function defines whether a variable is categorized as integral or continuous.

Usage

```
hi_solver_set_integrality(solver, index, type)
```

Arguments

solver	An object of class "highs_solver".
index	An integer vector specifying the variable index.
type	An integer vector representing the integrality type.

Value

The solver instance with updated integrality settings.

Examples

```
solver <- example_solver()
hi_solver_set_integrality(solver, 1, 1)
```

`hi_solver_set_objective`
Set the objective coefficient for a set of variables.

Description

This function assigns a coefficient to a variable in the objective function.

Usage

```
hi_solver_set_objective(solver, index, coeff)
```

Arguments

`solver`	An object of class "highs_solver".
`index`	The variable index.
`coeff`	A numeric value representing the objective coefficient.

Value

The solver instance with the updated objective.

Examples

```
solver <- example_solver()
hi_solver_set_objective(solver, 2, 3.5)
```

hi_solver_set_offset *Set the objective offset for the solver.*

Description

This function sets the objective offset in the solver instance.

Usage

```
hi_solver_set_offset(solver, ext_offset)
```

Arguments

solver	An object of class "highs_solver".
ext_offset	A numeric value representing the offset.

Value

The solver instance with the updated offset.

Examples

```
solver <- example_solver()
hi_solver_set_offset(solver, 5.0)
```

hi_solver_set_option *Set a HiGHS Solver Option*

Description

Sets the value of a specific option for a HiGHS solver instance.

Usage

```
hi_solver_set_option(
  solver,
  key,
  value,
  type = c("auto", "bool", "integer", "double", "string")
)
```

Arguments

solver	A HiGHS solver object of class "highs_solver".
key	A character string specifying the option name to set.
value	The value to set for the specified option. Will be coerced to the appropriate type.
type	Type of the option. Can be one of "auto", "bool", "integer", "double", or "string". When set to "auto" (default), the function will attempt to determine the type from the available options list. Specify a type directly if the option is valid but not listed in the available options.

Value

Invisibly returns NULL.

Examples

```
solver <- example_solver()
hi_solver_set_option(solver, "output_flag", FALSE)
hi_solver_set_option(solver, "solver", "simplex", type = "string")
```

---

hi_solver_set_options *Set Multiple HiGHS Solver Options*

Description

Sets multiple options for a HiGHS solver instance at once.

Usage

```
hi_solver_set_options(solver, control = list())
```

Arguments

solver	A HiGHS solver object of class "highs_solver".
control	A named list of options to set. Names should be valid option names and values will be coerced to the appropriate types.

Value

Invisibly returns NULL.

Examples

```
solver <- example_solver()
hi_solver_set_options(solver, list(output_flag = FALSE, solver = "simplex"))

control <- list(
  presolve = "on",
  solver = "simplex",
  parallel = "on",
  ranging = "off",
  time_limit = 100.0,
  primal_feasibility_tolerance = 1e-7,
  dual_feasibility_tolerance = 1e-7,
  random_seed = 1234,
  threads = 4,
  output_flag = TRUE,
  log_to_console = TRUE,
  run_crossover = "on",
  allow_unbounded_or_infeasible = FALSE,
  mip_detect_symmetry = TRUE,
  mip_max_nodes = 10000,
  mip_max_leaves = 5000,
  mip_feasibility_tolerance = 1e-6
)
hi_solver_set_options(solver, control)
```

---

hi_solver_set_sense *Set the optimization sense of the solver instance.*

Description

This function updates the optimization sense for the given solver instance. Use TRUE for maximization and FALSE for minimization.

Usage

```
hi_solver_set_sense(solver, maximum)
```

Arguments

solver	An object of class "highs_solver".
maximum	A boolean indicating whether to set maximization (TRUE) or minimization (FALSE).

Value

The updated solver instance with the new optimization sense.

Examples

```
solver <- example_solver()
hi_solver_set_sense(solver, TRUE)
```

`hi_solver_set_solution` *Set Solution for Warm-Start*

Description

Sets a complete primal and/or dual solution on the solver for warm-starting.

Usage

```
hi_solver_set_solution(
  solver,
  col_value,
  row_value,
  col_dual,
  row_dual,
  value_valid = TRUE,
  dual_valid = TRUE
)
```

Arguments

`solver`	An object of class "highs_solver".
`col_value`	Numeric vector of primal column values.
`row_value`	Numeric vector of primal row values.
`col_dual`	Numeric vector of dual column values.
`row_dual`	Numeric vector of dual row values.
`value_valid`	Logical, whether the primal values are valid.
`dual_valid`	Logical, whether the dual values are valid.

Value

An integer status code (0 = OK).

hi_solver_set_sparse_solution *Set Sparse Solution for Warm-Start*

Description

Sets a sparse primal solution (by index/value pairs) for warm-starting.

Usage

```
hi_solver_set_sparse_solution(solver, index, value)
```

Arguments

solver	An object of class "highs_solver".
index	Integer vector of column indices (0-based).
value	Numeric vector of values for those columns.

Value

An integer status code (0 = OK).

hi_solver_set_start *Set Starting Solution for the Solver*

Description

This function sets the starting solution for the solver.

Usage

```
hi_solver_set_start(solver, val)
```

Arguments

solver	An object of class "highs_solver".
val	A numeric vector of values for the variables. Missing values can be used to specify that particular variables should have their starting values calculated automatically.

Value

An integer value indicating the status.

Examples

```
solver <- example_solver()
hi_solver_set_start(solver, c(0, NA, NA))
```

`hi_solver_set_variable_bounds` *Set variable bounds for a set of variables.*

Description

This function sets the lower and upper bounds for a set of variables.

Usage

```
hi_solver_set_variable_bounds(solver, index, lower, upper)
```

Arguments

`solver`	An object of class "highs_solver".
`index`	The variable index.
`lower`	The lower bound.
`upper`	The upper bound.

Value

The solver instance with updated variable bounds.

Examples

```
solver <- example_solver()
hi_solver_set_variable_bounds(solver, 2, 0, 10)
```

hi_solver_status *Get Solver Status*

Description

This function returns the current status of the solver.

Usage

```
hi_solver_status(solver)
```

Arguments

solver	An object of class "highs_solver".

Value

A status code indicating the solver state.

Examples

```
solver <- example_solver()
hi_solver_run(solver)
status <- hi_solver_status(solver)
```

hi_solver_status_message *Get Solver Status Message*

Description

This function returns a human-readable message describing the current solver status.

Usage

```
hi_solver_status_message(solver)
```

Arguments

solver	An object of class "highs_solver".

Value

A character string containing the status message.

Examples

```
solver <- example_solver()
hi_solver_run(solver)
message <- hi_solver_status_message(solver)
```

`hi_solver_version` *Get HiGHS Version*

Description

Returns the version string of the bundled HiGHS solver.

Usage

```
hi_solver_version(solver)
```

Arguments

`solver`	An object of class "highs_solver".

Value

A character string with the HiGHS version.

`hi_solver_write_basis` *Write Basis to File*

Description

This function writes the current basis information to a file.

Usage

```
hi_solver_write_basis(solver, filename)
```

Arguments

`solver`	An object of class "highs_solver".
`filename`	A character string specifying the output file path.

Value

Invisible NULL.

Examples

```
solver <- example_solver()
basis_file <- tempfile(fileext = ".txt")
hi_solver_write_basis(solver, basis_file)
```

`hi_solver_write_model` *Write Model to File*

Description

This function writes the current optimization model to a file.

Usage

```
hi_solver_write_model(solver, filename)
```

Arguments

`solver`	An object of class "highs_solver".
`filename`	A character string specifying the output file path.

Value

Invisible NULL.

Examples

```
solver <- example_solver()
model_file <- tempfile(fileext = ".mps")
hi_solver_write_model(solver, model_file)
```

Index