Changing the rules of business™

ILOG CPLEX 10.2 Parameters

Reference Manual

March 2007

COPYRIGHT NOTICE

General Use Restrictions

Trademarks

document version 10.2

Table of Contents

About Parameters of CPLEX

The behavior of CPLEX is controlled by a variety of parameters that are each accessible and settable by the user. This manual lists these parameters and explains their settings in the CPLEX Component Libraries and the Interactive Optimizer. It also explains how to read and write parameter settings of the C API to a file.

Accessing Parameters

The following methods set and access parameters for objects of the class `IloCplex` in C++ and Java or the class `Cplex` in the .NET API:

```
setParam
getParam
getMin
```

```
getMax
getDefault
setDefaults
```

The names of the corresponding accessors in the class `Cplex` in .NET follow the usual conventions of names and capitalization of languages in that framework. For example, the class `Cplex` and its method `Solve` are denoted `Cplex.Solve`.

C applications and applications written in other languages callable from C access and set parameters with the following routines:

`CPXgetdblparam`	Accesses a parameter of type double
`CPXsetdblparam`	Changes a parameter of type double
`CPXinfodblparam`	Gets the default value and range of a parameter of type double
`CPXgetintparam`	Accesses a parameter of type integer
`CPXsetintparam`	Changes a parameter of type integer
`CPXinfointparam`	Gets the default value and range of a parameter of type integer
`CPXgetstrparam`	Accesses a parameter of type string
`CPXsetstrparam`	Changes a parameter of type string
`CPXinfostrparam`	Gets the default value of a parameter of type string
`CPXsetdefaults`	Resets all parameters to their standard default values
`CPXgetparamname`	Accesses the name of a parameter
`CPXgetparamnum`	Access the identifying number assigned to a parameter
`CPXgetchgparams`	Accesses all parameters not currently at their default value

Parameter Names

In the parameter table, each parameter has a name (that is, a symbolic constant) to refer to it within a program.

- For the C API, these constants are capitalized and start with `CPX_PARAM_`; for example, `CPX_PARAM_ITLIM`. They are used as the second argument in all parameter routines (except `CPXsetdefaults` which does not require them).

- For C++ applications, the parameters are defined in nested enumeration types for Boolean, integer, floating-point, and string parameters. The `enum` names use mixed (lower and upper) case letters and must be prefixed with the class name `IloCplex::` for scope. For example, `IloCplex::ItLim` is the `IloCplex` equivalent of `CPX_PARAM_ITLIM`.
- For Java applications, the parameters are defined as final static objects in nested classes called `IloCplex.BooleanParam`, `IloCplex.IntParam`, `IloCplex.DoubleParam`, and `IloCplex.StringParam` for Boolean, integer, floating-point, and string parameters, respectively. The parameter object names use mixed (lower and upper) case letters and must be prefixed with the appropriate class for scope. For example, `IloCplex.IntParam.ItLim` is the object representing the parameter `CPX_PARAM_ITLIM`.
- For .NET applications, the parameters follow the usual conventions for capitalizing attributes and defining scope within a namespace.

An integer that serves as a reference number for each parameter is shown in the table. That integer reference number corresponds to the value that each symbolic constant represents, as found in the `cplex.h` header file, but it is strongly recommended that the symbolic constants be used instead of their integer equivalents whenever possible, for the sake of portability to future versions of CPLEX.

Correspondence of Parameters

Some parameters available for the C API are not supported as parameters for the object oriented APIs or have a slightly different name there. In particular:

- `EpLin`, the parameter specifying the tolerance to use in linearization in the object oriented APIs (C++, Java, .NET), is not applicable in the C API.
- `CPX_PARAM_MIPCBREDLP`, the parameter indicating whether to use the reduced or original model in MIP callbacks, has no equivalent in the object oriented APIs.
- Logging output is controlled by a parameter in the C API (`CPX_PARAM_SCRIND`), but when using the object oriented APIs, you control logging by configuring the output channel:
    - `IloCplex::out` in C++

      For example, to turn off output to the screen, use `cplex.setOut(env.getNullStream())`.
    - `IloCplex.output` in Java

      For example, to turn off output to the screen, use `cplex.setOut(null)`.
    - `Cplex.Out` in .NET

For example, to turn off output to the screen, use `Cplex.SetOut(Null)`.

- The parameter `IloCplex::RootAlg` in the C++ API corresponds to these parameters in the C API:
  - `CPX_PARAM_STARTALG`
  - `CPX_PARAM_LPMETHOD`
  - `CPX_PARAM_QPMETHOD`
- The parameter `IloCplex::NodeAlg` in the C++ API corresponds to the parameter `CPX_PARAM_SUBALG` in the C API.

Saving Parameter Settings to a File

It is possible to read and write a file of parameter settings with the C API. The file extension is `.prm`. The C routine `CPXreadcopyparam` reads parameter values from a file with the `.prm` extension. The routine `CPXwriteparam` writes a file of the current nondefault parameter settings to a file with the `.prm` extension. Here is the format of such a file:

```
CPLEX Parameter File Version number
  parameter_name   parameter_value
```

CPLEX reads the entire file before changing any of the parameter settings. After successfully reading a parameter file, the C API first sets all parameters to their default value. Then it applies the settings it read in the parameter file. No changes are made if the parameter file contains errors, such as missing or illegal values. There is no checking for duplicate entries in the file. In the case of duplicate entries, the last setting in the file is applied.

When you write a parameter file from the C API, only the non-default values are written to the file. String values may be double-quoted or not, but are always written with double quotation marks.

The comment character in a parameter file is #. After that character, CPLEX ignores the rest of the line.

The C API issues a warning if the version recorded in the parameter file does not match the version of the product. A warning is also issued of a nonintegral value is given for an integer-valued parameter.

Here is an example of a correct CPLEX parameter file:

```
CPLEX Parameter File Version 10.0
CPX_PARAM_EPPER                  3.45000000000000e-06
CPX_PARAM_OBJULIM                1.23456789012345e+05
CPX_PARAM_PERIND                 1
CPX_PARAM_SCRIND                 1
CPX_PARAM_WORKDIR                "tmp"
```

Topical List of Parameters

The following lists offer you access to the documentation of CPLEX parameters, organized by topics.

Simplex

Advanced start switch on page 26

Lower objective value limit on page 109

Upper objective value limit on page 110

Dual simplex pricing algorithm on page 63

Primal simplex pricing algorithm on page 114

Simplex crash ordering on page 53

Markowitz tolerance on page 68

Optimality tolerance on page 69

Perturbation constant on page 70

Simplex perturbation switch on page 111

Simplex perturbation limit on page 112

Relaxation for `feasOpt` on page 71

Feasibility tolerance on page 72

Simplex maximum iteration limit on page 84

Reduces use of memory on page 86

Numerical precision emphasis on page 106

Simplex pricing candidate list size on page 120

Simplex refactoring frequency on page 126

Sifting subproblem algorithm on page 139

Simplex iteration information display on page 142

Simplex singularity repair limit on page 143

Barrier

Advanced start switch on page 26

Barrier algorithm on page 30

Barrier starting point algorithm on page 41

Barrier crossover algorithm on page 32

Sifting subproblem algorithm on page 139

Barrier ordering algorithm on page 39

Barrier display information on page 33

Barrier growth limit on page 35

Barrier column nonzeros on page 31

Barrier iteration limit on page 36

Barrier maximum correction limit on page 37

Barrier objective range on page 38

Barrier thread limit on page 42

Convergence tolerance for LP and QP problems on page 34

Convergence tolerance for quadratically constrained problems (QCP) on page 40

Reduces use of memory on page 86

Numerical precision emphasis on page 106

MIP

The parameters controling MIP behavior are accessible through the following topics:

- *MIP General* on page 19
- *MIP Strategies* on page 20
- *MIP Cuts* on page 20
- *MIP Tolerances* on page 21
- *MIP Limits* on page 21

MIP General

Advanced start switch on page 26

MIP emphasis switch on page 89

MIP subproblem algorithm on page 102

Reapply presolve after processing the root node on page 130

Relaxed LP presolve switch on page 127

Indefinite MIQP switch on page 123

Bound strengthening switch on page 44

Reduces use of memory on page 86

Numerical precision emphasis on page 106

MIP callback switch between original model and reduced, presolved model on page 87

MIP node log display information on page 88

MIP node log interval on page 90

Node storage file switch on page 103

MIP Strategies

MIP starting algorithm on page 135

MIP variable selection strategy on page 152

MIP strategy best bound interval on page 43

MIP branching direction on page 45

Backtracking tolerance on page 46

MIP dive strategy on page 62

MIP heuristic frequency on page 81

Local branching heuristic on page 85

MIP priority order switch on page 91

MIP priority order generation on page 92

MIP node selection strategy on page 105

Node presolve selector on page 119

MIP probing level on page 121

RINS heuristic frequency on page 131

MIP Cuts

Constraint aggregation limit for cut generation on page 27

Row multiplier factor for cuts on page 57

MIP cliques switch on page 47

MIP covers switch on page 52

MIP disjunctive cuts switch on page 61

MIP flow cover cuts switch on page 75

MIP flow path cut switch on page 76

MIP Gomory fractional cuts switch on page 78

MIP GUB cuts switch on page 80

MIP implied bound cuts switch on page 82

MIP MIR (mixed integer rounding) cut switch on page 94

Pass limit for generating Gomory fractional cuts on page 79

Candidate limit for generating Gomory fractional cuts on page 77

Number of cutting plane passes on page 56

MIP Tolerances

Backtracking tolerance on page 46

Lower cutoff on page 55

Upper cutoff on page 58

Absolute objective difference cutoff on page 108

Relative objective difference cutoff on page 128

Absolute mipgap tolerance on page 64

Relative MIP gap tolerance on page 65

Integrality tolerance on page 66

Relaxation for `feasOpt` on page 71

MIP Limits

MIP solution limit on page 83

Pass limit for generating Gomory fractional cuts on page 79

Candidate limit for generating Gomory fractional cuts on page 77

Constraint aggregation limit for cut generation on page 27

Row multiplier factor for cuts on page 57

Number of cutting plane passes on page 56

MIP thread limit on page 93

MIP parallel threads limit on page 146

MIP node limit on page 104

Time spent polishing a solution on page 113

Time spent probing on page 122

Frequency to try to repair infeasible MIP start on page 129

MIP candidate list limit on page 144

MIP simplex iterations limit on page 145

Limit on nodes explored when a subMIP is being solved on page 147

Tree memory limit on page 151

Network

Optimality tolerance for network optimization on page 97

Feasibility tolerance for network primal optimization on page 98

Simplex network extraction level on page 99

Network simplex iteration limit on page 100

Network simplex pricing algorithm on page 101

Network logging display switch on page 96

Sifting

Sifting subproblem algorithm on page 139

Sifting information display on page 140

Upper limit on sifting iterations on page 141

Preprocessing: Aggregator, Presolver

Symmetry breaking on page 148

Preprocessing aggregator fill on page 28

Preprocessing aggregator application limit on page 29

Bound strengthening switch on page 44

Coefficient reduction setting on page 49

Dependency switch on page 60

Presolve dual setting on page 115

Presolve switch on page 116

Linear reduction switch on page 117

Limit on the number of presolve passes made on page 118

Node presolve selector on page 119

Relaxed LP presolve switch on page 127

Reapply presolve after processing the root node on page 130

Primal and dual reduction type on page 125

Tolerances

Convergence tolerance for LP and QP problems on page 34

Convergence tolerance for quadratically constrained problems (QCP) on page 40

Backtracking tolerance on page 46

Lower cutoff on page 55

Upper cutoff on page 58

Absolute mipgap tolerance on page 64

Relative MIP gap tolerance on page 65

Integrality tolerance on page 66

Epsilon used in linearization on page 67

Markowitz tolerance on page 68

Optimality tolerance on page 69

Optimality tolerance for network optimization on page 97

Feasibility tolerance on page 72

Relaxation for `feasOpt` on page 71

Absolute objective difference cutoff on page 108

Relative objective difference cutoff on page 128

Perturbation constant on page 70

Limits

Memory available for working storage on page 154

Global default thread count on page 149

Global time limit on page 150

Variable (column) read limit on page 50

Nonzero element read limit on page 107

QP Q matrix nonzero read limit on page 124

Constraint (row) read limit on page 136

Display and Output

Messages to screen switch on page 138

Barrier display information on page 33

Simplex iteration information display on page 142

Sifting information display on page 140

MIP node log display information on page 88

MIP node log interval on page 90

Network logging display switch on page 96

Computation time reporting on page 48

Conflict information display on page 51

Data consistency checking switch on page 59

Precision of numerical output in MPS and REW file formats on page 95

Directory for working files on page 153

Alphabetic List of Parameters

The following list offers you access to documentation of CPLEX parameters in alphabetic order. To browse through parameters organized topically, see *Topical List of Parameters* on page 17.

Summary Advanced start switch

C Name `CPX_PARAM_ADVIND`

C++ Name `AdvInd`

Java Name `AdvInd`

.NET Name `AdvInd`

InteractiveOptimizer `advance`

Identifier `1001`

Description If set to 1 or 2, this parameter indicates that CPLEX should use advanced starting information when optimization is initiated.

For MIP models, settings 1 and 2 are currently identical. Both will cause CPLEX to continue with a partially explored MIP tree if one is available. If tree exploration has not yet begun, settings 1 or 2 indicate that CPLEX should use a loaded MIP start, if available.

For continuous models solved with simplex, setting 1 will use the currently loaded basis. If a basis is available only for the original, unpresolved model, or if CPLEX has a start vector rather than a simplex basis, then the simplex algorithm will proceed on the unpresolved model. With setting 2, CPLEX will first perform presolve on the model and on the basis or start vector, and then proceed with optimization on the presolved problem.

Setting 2 can be particularly useful for solving fixed MIP models, where a start vector but no corresponding basis is available.

For continuous models solved with the barrier algorithm, settings 1 or 2 will continue optimization from the last available barrier iterate.

Values

Value	Meaning
0	Do not use advanced start information
1	Use an advanced basis supplied by the user; **default**
2	Crush an advanced basis or starting vector supplied by the user

Summary Constraint aggregation limit for cut generation

C Name `CPX_PARAM_AGGCUTLIM`

C++ Name `AggCutLim`

Java Name `AggCutLim`

.NET Name `AggCutLim`

InteractiveOptimizer `mip limits aggforcut`

Identifier `2054`

Description Limits the number of constraints that can be aggregated for generating flow cover and mixed integer rounding (MIR) cuts.

Values Any nonnegative integer; **default**: 3

Summary	Preprocessing aggregator fill
C Name	`CPX_PARAM_AGGFILL`
C++ Name	`AggFill`
Java Name	`AggFill`
.NET Name	`AggFill`
InteractiveOptimizer	`preprocessing fill`
Identifier	`1002`
Description	Limits variable substitutions by the aggregator. If the net result of a single substitution is more nonzeros than this value, the substitution is not made.
Values	Any nonnegative integer ; **default**: 10

Summary Preprocessing aggregator application limit

C Name `CPX_PARAM_AGGIND`

C++ Name `AggInd`

Java Name `AggInd`

.NET Name `AggInd`

InteractiveOptimizer `preprocessing aggregator`

Identifier `1003`

Description Invokes the aggregator to use substitution where possible to reduce the number of rows and columns before the problem is solved. If set to a positive value, the aggregator is applied the specified number of times or until no more reductions are possible.

Values

Value	Meaning
-1	Automatic (1 for LP, infinite for MIP) **default**
0	Do not use any aggregator
Any positive integer	Number of times to apply aggregator

Summary Barrier algorithm

C Name `CPX_PARAM_BARALG`

C++ Name `BarAlg`

Java Name `BarAlg`

.NET Name `BarAlg`

InteractiveOptimizer `barrier algorithm`

Identifier `3007`

Description The default setting 0 uses the "infeasibility - estimate start" algorithm (setting 1) when solving subproblems in a MIP problem, and the standard barrier algorithm (setting 3) in other cases. The standard barrier algorithm is almost always fastest. However, on problems that are primal or dual infeasible (common for MIP subproblems), the standard algorithm may not work as well as the alternatives. The two alternative algorithms (settings 1 and 2) may eliminate numerical difficulties related to infeasibility, but are generally slower.

Values

Value	Meaning
0	**Default** setting
1	Infeasibility-estimate start
2	Infeasibility-constant start
3	Standard barrier

Summary Barrier column nonzeros

C Name `CPX_PARAM_BARCOLNZ`

C++ Name `BarColNz`

Java Name `BarColNz`

.NET Name `BarColNz`

InteractiveOptimizer `barrier colnonzeros`

Identifier `3009`

Description Used in the recognition of dense columns. If columns in the presolved and aggregated problem exist with more entries than this value, such columns are considered dense and are treated specially by the CPLEX Barrier Optimizer to reduce their effect.

Values

Value	Meaning
0	Dynamically calculated; **default**
Any positive integer	Number of nonzero entries that make a column dense

Summary Barrier crossover algorithm

C Name `CPX_PARAM_BARCROSSALG`

C++ Name `BarCrossAlg`

Java Name `BarCrossAlg`

.NET Name `BarCrossAlg`

InteractiveOptimizer `barrier crossover`

Identifier `3018`

Description Determines which, if any, crossover is performed at the end of a barrier optimization. This parameter also applies when CPLEX uses the Barrier Optimizer to solve an LP or QP problem, or when it is used to solve the continuous relaxation of an MILP or MIQP at a node in a MIP.

Values

Value	Meaning
-1	No crossover
0	Automatic: let CPLEX choose; **default**
1	Primal crossover
2	Dual crossover

Summary Barrier display information

C Name `CPX_PARAM_BARDISPLAY`

C++ Name `BarDisplay`

Java Name `BarDisplay`

.NET Name `BarDisplay`

InteractiveOptimizer `barrier display`

Identifier `3010`

Description Determines the level of barrier progress information to be displayed.

Values

Value	Meaning
0	No progress information
1	Normal setup and iteration information; **default**
2	Diagnostic information

Summary Convergence tolerance for LP and QP problems

C Name `CPX_PARAM_BAREPCOMP`

C++ Name `BarEpComp`

Java Name `BarEpComp`

.NET Name `BarEpComp`

InteractiveOptimizer `barrier convergetol`

Identifier `3002`

Description Sets the tolerance on complementarity for convergence. The barrier algorithm terminates with an optimal solution if the relative complementarity is smaller than this value.

Changing this tolerance to a smaller value may result in greater numerical precision of the solution, but also increases the chance of failure to converge in the algorithm and consequently may result in no solution at all. Therefore, caution is advised in deviating from the default setting.

Values Any positive number greater than or equal to 1e-12; **default**: 1e-8.

See Also For problems with quadratic constraints (QCP), see `CPX_PARAM_BARQCPEPCOMP`, `BarQCPEpComp`.

Summary Barrier growth limit

C Name `CPX_PARAM_BARGROWTH`

C++ Name `BarGrowth`

Java Name `BarGrowth`

.NET Name `BarGrowth`

InteractiveOptimizer `barrier limits growth`

Identifier `3003`

Description Used to detect unbounded optimal faces. At higher values, the barrier algorithm is less likely to conclude that the problem has an unbounded optimal face, but more likely to have numerical difficulties if the problem has an unbounded face.

Values 1.0 or greater; **default**: 1e12.

Summary

Barrier iteration limit

C Name

`CPX_PARAM_BARITLIM`

C++ Name

`BarItLim`

Java Name

`BarItLim`

.NET Name

`BarItLim`

InteractiveOptimizer

`barrier limits iterations`

Identifier

`3012`

Description

Sets the number of barrier iterations before termination. When this parameter is set to 0 (zero), no barrier iterations occur, but problem setup occurs and information about the setup is displayed (such as Cholesky factor statistics).

Values

Value	Meaning
0	No barrier iterations
2 100 000 000	**default**
Any positive integer	Number of barrier iterations before termination

Summary Barrier maximum correction limit

C Name `CPX_PARAM_BARMAXCOR`

C++ Name `BarMaxCor`

Java Name `BarMaxCor`

.NET Name `BarMaxCor`

InteractiveOptimizer `barrier limits corrections`

Identifier `3013`

Description Sets the maximum number of centering corrections done on each iteration. An explicit value greater than 0 (zero) may improve the numerical performance of the algorithm at the expense of computation time.

Values

Value	Meaning
-1	Automatic; let CPLEX choose; **default**
0	None
Any positive integer	Maximum number of centering corrections per iteration

Summary	Barrier objective range
C Name	`CPX_PARAM_BAROBJRNG`
C++ Name	`BarObjRng`
Java Name	`BarObjRng`
.NET Name	`BarObjRng`
InteractiveOptimizer	`barrier limits objrange`
Identifier	`3004`
Description	Sets the maximum absolute value of the objective function. The barrier algorithm looks at this limit to detect unbounded problems.
Values	Any nonnegative number; **default**: 1e20

Summary Barrier ordering algorithm

C Name `CPX_PARAM_BARORDER`

C++ Name `BarOrder`

Java Name `BarOrder`

.NET Name `BarOrder`

InteractiveOptimizer `barrier ordering`

Identifier `3014`

Description Sets the algorithm to be used to permute the rows of the constraint matrix in order to reduce fill in the Cholesky factor.

Values

Value	Meaning
0	Automatic: let CPLEX choose; **default**
1	Approximate minimum degree (AMD)
2	Approximate minimum fill (AMF)
3	Nested dissection (ND)

Summary	Convergence tolerance for quadratically constrained problems (QCP)
C Name	`CPX_PARAM_BARQCPEPCOMP`
C++ Name	`BarQCPEpComp`
Java Name	`BarQCPEpComp`
.NET Name	`BarQCPEpComp`
InteractiveOptimizer	`barrier qcpconvergetol`
Identifier	`3020`

Description

Sets the tolerance on complementarity for convergence in quadratically constrained problems (QCPs). The barrier algorithm terminates with an optimal solution if the relative complementarity is smaller than this value.

Changing this tolerance to a smaller value may result in greater numerical precision of the solution, but also increases the chance of a convergence failure in the algorithm and consequently may result in no solution at all. Therefore, caution is advised in deviating from the default setting.

Values

Any positive number greater than or equal to 1e-12; **default**: 1e17.

See Also

For LPs and for QPs (that is, when all the constraints are linear) see `CPX_PARAM_BAREPCOMP`, `BarEpComp`.

Summary Barrier starting point algorithm

C Name `CPX_PARAM_BARSTARTALG`

C++ Name `BarStartAlg`

Java Name `BarStartAlg`

.NET Name `BarStartAlg`

InteractiveOptimizer `barrier startalg`

Identifier `3017`

Description Sets the algorithm to be used to compute the initial starting point for the barrier optimizer.

Values

Value	Meaning
1	Dual is 0 (zero); **default**
2	Estimate dual
3	Average of primal estimate, dual 0 (zero)
4	Average of primal estimate, estimate dual

Summary Barrier thread limit

C Name `CPX_PARAM_BARTHREADS`

C++ Name `BarThreads`

Java Name `BarThreads`

.NET Name `BarThreads`

InteractiveOptimizer `barrier limits threads`

Identifier `3016`

Description Determines the maximum number of parallel processes (threads) that will be invoked by the parallel barrier optimizer. The default value of 0 (zero) means that the limit will be determined by the value of the global thread limit parameter (`CPX_PARAM_THREADS`, `Threads`). A positive value will override the value found in the global thread limit parameter.

Values

Value	Meaning
0	Number of threads determined by global thread limit; **default**
greater than 0	Upper limit on number of threads for parallel barrier algorithm

See Also `CPX_PARAM_THREADS`, `Threads`

Summary MIP strategy best bound interval

C Name `CPX_PARAM_BBINTERVAL`

C++ Name `BBInterval`

Java Name `BBInterval`

.NET Name `BBInterval`

InteractiveOptimizer `mip strategy bbinterval`

Identifier `2039`

Description Sets the best bound interval for MIP strategy.

When you set this parameter to best estimate node selection, the best bound interval is the interval at which the best bound node, instead of the best estimate node, is selected from the tree. A best bound interval of 0 (zero) means “never select the best bound node.” A best bound interval of 1 (one) means “always select the best bound node,” and is thus equivalent to nodeselect 1 (one).

Higher values of this parameter mean that the best bound node will be selected less frequently; experience has shown it to be beneficial to select the best bound node occaisionally, and therefore the default value of this parameter is 7.

Values

Value	Meaning
0	Never select best bound node; always select best estimate
1	Always select best bound node
7	Select best bound node occaisionally; **default**
Any positive integer	Select best bound node less frequently than best estimate node

See Also `CPX_PARAM_NODESEL`, `NodeSel`

Summary Bound strengthening switch

C Name `CPX_PARAM_BNDSTRENIND`

C++ Name `BndStrenInd`

Java Name `BndStrenInd`

.NET Name `BndStrenInd`

InteractiveOptimizer `preprocessing boundstrength`

Identifier `2029`

Description Determines whether to apply bound strengthening in mixed integer programs (MIPs). Bound strengthening tightens the bounds on variables, perhaps to the point where the variable can be fixed and thus removed from consideration during branch & cut.

Values

Value	Meaning
-1	Automatic: let CPLEX choose; **default**
0	Do not apply bound strengthening
1	Apply bound strengthening

Summary MIP branching direction

C Name `CPX_PARAM_BRDIR`

C++ Name `BrDir`

Java Name `BrDir`

.NET Name `BrDir`

InteractiveOptimizer `mip strategy branch`

Identifier `2001`

Description Determines which branch, the up or the down branch, should be taken first at each node.

Values

Value	Symbol	Meaning
-1	`CPX_BRDIR_DOWN`	Down branch selected first
0	`CPX_BRDIR_AUTO`	Automatic: let CPLEX choose; **default**
1	`CPX_BRDIR_UP`	Up branch selected first

Summary Backtracking tolerance

C Name `CPX_PARAM_BTTOL`

C++ Name `BtTol`

Java Name `BtTol`

.NET Name `BtTol`

InteractiveOptimizer `mip strategy backtrack`

Identifier `2002`

Description Controls how often backtracking is done during the branching process. The decision when to backtrack depends on three values that change during the course of the optimization:

- the objective function value of the best integer feasible solution (*incumbent*)
- the best remaining objective function value of any unexplored node (*best node*)
- the objective function value of the most recently solved node (*current objective*).

If a cutoff tolerance (`CPX_PARAM_CUTUP`, `CutUp` or `CPX_PARAM_CUTLO`, `CutLo`) has been set by the user, then that value is used as the incumbent until an integer feasible solution is found.

The *target gap* is defined to be the absolute value of the difference between the incumbent and the best node, multiplied by this backtracking parameter. CPLEX does not backtrack until the absolute value of the difference between the objective of the current node and the best node is at least as large as the target gap.

Low values of this backtracking parameter thus tend to increase the amount of backtracking, which makes the search process more of a pure best-bound search. Higher parameter values tend to decrease backtracking, making the search more of a pure depth-first search.

The backtracking value has effect only after an integer feasible solution is found or when a cutoff has been specified. Note that this backtracking value merely permits backtracking but does not force it; CPLEX may choose to continue searching a limb of the tree if that limb seems a promising candidate for finding an integer feasible solution.

Values Any number from 0.0 to 1.0; **default**: 0.9999

See Also `CPX_PARAM_CUTUP`, `CutUp` *and* `CPX_PARAM_CUTLO` , `CutLo`.

Summary MIP cliques switch

C Name `CPX_PARAM_CLIQUES`

C++ Name `Cliques`

Java Name `Cliques`

.NET Name `Cliques`

InteractiveOptimizer `mip cuts cliques`

Identifier `2003`

Description Determines whether or not clique cuts should be generated for the problem. Setting the value to 0 (zero), the default, indicates that the attempt to generate cliques should continue only if it seems to be helping.

Values

Value	Meaning
-1	Do not generate clique cuts
0	Automatic: let CPLEX choose; **default**
1	Generate clique cuts moderately
2	Generate clique cuts aggressively
3	Generate clique cuts very aggressively

Summary Computation time reporting

C Name `CPX_PARAM_CLOCKTYPE`

C++ Name `ClockType`

Java Name `ClockType`

.NET Name `ClockType`

InteractiveOptimizer `clocktype`

Identifier `1006`

Description Determines how computation times are measured on UNIX platforms. Computation time on Windows systems is always measured as wall clock time. Small variations in measured time on identical runs may be expected on any computer system under either setting of this parameter.

Values

Value	Meaning
1	CPU time; **default**
2	Wall clock time (total physical time elapsed)

Summary Coefficient reduction setting

C Name `CPX_PARAM_COEREDIND`

C++ Name `CoeRedInd`

Java Name `CoeRedInd`

.NET Name `CoeRedInd`

InteractiveOptimizer `preprocessing coeffreduce`

Identifier `2004`

Description Determines how coefficient reduction is used. Coefficient reduction improves the objective value of the initial (and subsequent) LP relaxations solved during branch & cut by reducing the number of non-integral vertices.

Values

Value	Meaning
0	Do not use coefficient reduction
1	Reduce only to integral coefficients
2	Reduce all potential coefficients; **default**

Summary Variable (column) read limit

C Name `CPX_PARAM_COLREADLIM`

C++ Name `ColReadLim`

Java Name `ColReadLim`

.NET Name `ColReadLim`

InteractiveOptimizer `read variables`

Identifier `1023`

Description Specifies a limit for the number of columns (variables) to read for an allocation of memory.

This parameter does not restrict the size of a problem. Rather, it indirectly specifies the default amount of memory that will be pre-allocated before a problem is read from a file. If the limit is exceeded, more memory is automatically allocated.

Values Any integer from 0 to 268 435 450; **default**: 60 000.

Summary Conflict information display

C Name `CPX_PARAM_CONFLICTDISPLAY`

C++ Name `ConflictDisplay`

Java Name `ConflictDisplay`

.NET Name `ConflictDisplay`

InteractiveOptimizer `display conflict`

Identifier `1074`

Description Determines how much information CPLEX reports when the conflict refiner is working.

Values

Value	Meaning
0	No display
1	Summary display; **default**
2	Detailed display

Summary MIP covers switch

C Name `CPX_PARAM_COVERS`

C++ Name `Covers`

Java Name `Covers`

.NET Name `Covers`

InteractiveOptimizer `mip cuts covers`

Identifier `2005`

Description Determines whether or not cover cuts should be generated for the problem. Setting the value to 0 (zero), the default, indicates that the attempt to generate covers should continue only if it seems to be helping.

Values

Value	Meaning
-1	Do not generate cover cuts
0	Automatic: let CPLEX choose; **default**
1	Generate cover cuts moderately
2	Generate cover cuts aggressively
3	Generate cover cuts very aggressively

Summary	Simplex crash ordering
C Name	`CPX_PARAM_CRAIND`
C++ Name	`CraInd`
Java Name	`CraInd`
.NET Name	`CraInd`
InteractiveOptimizer	`simplex crash`
Identifier	`1007`

Description Determines how CPLEX orders variables relative to the objective function when selecting an initial basis.

Values

Value	Meaning
LP Primal	
-1	Alternate ways of using objective coefficients
0	Ignore objective coefficients during crash
1	Alternate ways of using objective coefficients; **default**
LP Dual	
-1	Aggressive starting basis
0	Aggressive starting basis
1	Default starting basis; **default**
QP Primal	
-1	Slack basis
0	Ignore Q terms and use LP solver for crash
1	Ignore objective and use LP solver for crash; **default**
QP Dual	
-1	Slack basis
0	Use Q terms for crash
1	Use Q terms for crash; **default**

Summary Lower cutoff

C Name `CPX_PARAM_CUTLO`

C++ Name `CutLo`

Java Name `CutLo`

.NET Name `CutLo`

InteractiveOptimizer `mip tolerances lowercutoff`

Identifier `2006`

Description Sets lower cutoff tolerance. When the problem is a maximization problem, the lower cutoff parameter is used to cut off any nodes that have an objective value at or below the lower cutoff value. When a mixed integer optimization problem is continued, the larger of these values and the updated cutoff found during optimization are used during the next mixed integer optimization.

A too-restrictive value for the lower cutoff parameter may result in no integer solutions being found.

Values Any number; **default**: -1e+75.

Summary Number of cutting plane passes

C Name `CPX_PARAM_CUTPASS`

C++ Name `CutPass`

Java Name `CutPass`

.NET Name `CutPass`

InteractiveOptimizer `mip limits cutpasses`

Identifier `2056`

Description Sets the upper limit on the number of cutting plane passes CPLEX performs when solving the root node of a MIP model.

Values

Value	Meaning
-1	None
0	Automatic: let CPLEX choose; **default**
Any positive integer	Number of passes to perform

Summary Row multiplier factor for cuts

C Name `CPX_PARAM_CUTSFACTOR`

C++ Name `CutsFactor`

Java Name `CutsFactor`

.NET Name `CutsFactor`

InteractiveOptimizer `mip limits cutsfactor`

Identifier `2033`

Description Limits the number of cuts that can be added. The number of rows in the problem with cuts added is limited to `CutsFactor` times the original number of rows. If the problem is presolved, the original number of rows is that from the presolved problem.

A `CutsFactor` of 1.0 or less means that no cuts will be generated.

Because cuts can be added and removed during the course of optimization, `CutsFactor` may not correspond directly to the number of cuts seen in the node log or in the summary table at the end of optimization.

Values Any nonnegative number; **default**: 4.0

Summary	Upper cutoff
C Name	`CPX_PARAM_CUTUP`
C++ Name	`CutUp`
Java Name	`CutUp`
.NET Name	`CutUp`
InteractiveOptimizer	`mip tolerances uppercutoff`
Identifier	`2007`

Description

Sets the upper cutoff tolerance. When the problem is a minimization problem, CPLEX cuts off any nodes that have an objective value at or above the upper cutoff value . When a mixed integer optimization problem is continued, the smaller of these values and the updated cutoff found during optimization are used during the next mixed integer optimization.

A too-restrictive value for the upper cutoff parameter may result in no integer solutions being found.

Values

Any number; **default**: 1e+75.

Summary Data consistency checking switch

C Name `CPX_PARAM_DATACHECK (int)`

C++ Name `DataCheck (bool)`

Java Name `DataCheck`

.NET Name `DataCheck`

InteractiveOptimizer `read datacheck`

Identifier `1056`

Description Determines whether data should be checked for consistency. When this parameter is on, the routines CPXcopy____, CPXread____ and CPXchg____ of the C API perform extensive checking of data in their array arguments, such as checking that indices are within range, that there are no duplicate entries, and that values are valid for the type of data or are valid numbers. This checking is useful for debugging applications. When this checking identifies trouble, you can gather more specific detail by calling one of the routines in `check.c`.

Values

int	bool	Symbol	Meaning
0	false	CPX_OFF	Data checking off; do not check; **default**
1	true	CPX_ON	Data checking on

Summary Dependency switch

C Name `CPX_PARAM_DEPIND`

C++ Name `DepInd`

Java Name `DepInd`

.NET Name `DepInd`

InteractiveOptimizer `preprocessing dependency`

Identifier `1008`

Description Determines whether to activate the dependency checker. If on, the dependency checker searches for dependent rows during preprocessing. If off, dependent rows are not identified.

Values

Value	Meaning
-1	Automatic: let CPLEX choose; **default**
0	Off: do not use dependency checker
1	Turn on only at the beginning of preprocessing
2	Turn on only at the end of preprocessing
3	Turn on at the beginning and at the end of preprocessing

Summary MIP disjunctive cuts switch

C Name `CPX_PARAM_DISJCUTS`

C++ Name `DisjCuts`

Java Name `DisjCuts`

.NET Name `DisjCuts`

InteractiveOptimizer `mip cuts disjunctive`

Identifier `2053`

Description Determines whether or not disjunctive cuts should be generated for the problem. Setting the value to 0 (zero), the default, indicates that the attempt to generate disjunctive cuts should continue only if it seems to be helping.

Values

Value	Meaning
-1	Do not generate disjunctive cuts
0	Automatic: let CPLEX choose; **default**
1	Generate disjunctive cuts moderately
2	Generate disjunctive cuts aggressively
3	Generate disjunctive cuts very aggressively

Summary MIP dive strategy

C Name `CPX_PARAM_DIVETYPE`

C++ Name `DiveType`

Java Name `DiveType`

.NET Name `DiveType`

InteractiveOptimizer `mip strategy dive`

Identifier `2060`

Description Controls the MIP dive strategy. The MIP traversal strategy occasionally performs probing dives, where it looks ahead at both children nodes before deciding which node to choose. The default (automatic) setting lets CPLEX choose when to perform a probing dive, 1 (one) directs CPLEX never to perform probing dives, 2 always to probe, 3 to spend more time exploring potential solutions that are similar to the current incumbent. Setting 2, always to probe, is helpful for finding integer solutions.

Values

Value	Meaning
0	Automatic: let CPLEX choose; **default**
1	Traditional dive
2	Probing dive
3	Guided dive

Summary Dual simplex pricing algorithm

C Name CPX_PARAM_DPRIIND

C++ Name DPriInd

Java Name DPriInd

.NET Name DPriInd

InteractiveOptimizer simplex dgradient

Identifier 1009

Description Determines the type of pricing applied in the dual simplex algorithm. The default pricing (0) usually provides the fastest solution time, but many problems benefit from alternate settings.

Values

Value	Symbol	Meaning
0	CPX_DPRIIND_AUTO	Automatic: let CPLEX choose; **default**
1	CPX_DPRIIND_FULL	Standard dual pricing
2	CPX_DPRIIND_STEEP	Steepest-edge pricing
3	CPX_DPRIIND_FULL_STEEP	Steepest-edge pricing in slack space
4	CPX_DPRIIND_STEEPQSTART	Steepest-edge pricing, unit initial norms
5	CPX_DPRIIND_DEVEX	devex pricing

See Also CPX_PARAM_FRACCAND, FracCand

CPX_PARAM_FRACCUTS, FracCuts

CPX_PARAM_FRACPASS, FracPass

Summary	Absolute mipgap tolerance
C Name	`CPX_PARAM_EPAGAP`
C++ Name	`EpAGap`
Java Name	`EpAGap`
.NET Name	`EpAGap`
InteractiveOptimizer	`mip tolerances absmipgap`
Identifier	`2008`
Description	Sets an absolute tolerance on the gap between the best integer objective and the objective of the best node remaining. When this difference falls below the value of this parameter, the mixed integer optimization is stopped.
Values	Any nonnegative number; **default**: 1e-06.

Summary Relative MIP gap tolerance

C Name `CPX_PARAM_EPGAP`

C++ Name `EpGap`

Java Name `EpGap`

.NET Name `EpGap`

InteractiveOptimizer `mip tolerances mipgap`

Identifier `2009`

Description Sets a relative tolerance on the gap between the best integer objective and the objective of the best node remaining. When the value

`|bestnode-bestinteger|/(1e-10+|bestinteger|)`

falls below the value of this parameter, the mixed integer optimization is stopped.

For example, to instruct CPLEX to stop as soon as it has found a feasible integer solution proved to be within five percent of optimal, set the relative mipgap tolerance to 0.05.

Values Any number from 0.0 to 1.0; **default**: 1e-04.

Summary Integrality tolerance

C Name `CPX_PARAM_EPINT`

C++ Name `EpInt`

Java Name `EpInt`

.NET Name `EpInt`

InteractiveOptimizer `mip tolerances integrality`

Identifier `2010`

Description Specifies the amount by which an integer variable can be different from an integer and still be considered feasible.

A value of zero is permitted, and the optimizer will attempt to meet this tolerance.

However, in some models, computer roundoff may still result in small, nonzero deviations from integrality. If any of these deviations exceed the value of this parameter, or exceed 1e-10 in the case where this parameter has been set to a value less than that, a solution status of `CPX_STAT_OPTIMAL_INFEAS` will be returned instead of the usual `CPX_STAT_OPTIMAL`.

Values Any number from 0.0 to 0.5; **default**: 1e-05.

Summary	Epsilon used in linearization
C Name	`CPX_PARAM_EPLIN but not applicable in the C API`
C++ Name	`EpLin`
Java Name	`EpLin`
.NET Name	`EpLin`
InteractiveOptimizer	`not available in the Interactive Optimizer`
Identifier	`2068`

Description

Sets the epsilon (degree of tolerance) used in linearization in the object-oriented APIs.

Not applicable in the C API.

Not available in the Interactive Optimizer.

This parameter controls how strict inequalities are managed during linearization. In other words, it provides an epsilon for determining when two values are not equal during linearization. For example, when x is a numeric variable (that is, an instance of `IloNumVar`),

`x < a`

becomes

`x <= a-eplin`.

Similarly, `x!=a`

becomes

`{(x < a) || (x > a)}`

which is linearized automatically for you in the object-oriented APIs as

`{( x <= a-eplin) || (x >= a+eplin)}`.

Exercise caution in changing this parameter from its default value: the smaller the epsilon, the more numerically unstable the model will tend to become. If you are not getting an expected solution for an object-oriented model that uses linearization, it might be that this solution is cut off because of the relatively high `EpLin` value. In such a case, carefully try reducing it.

Values

Any positive value greater than zero; **default**: 1e-3.

Summary	Markowitz tolerance
C Name	`CPX_PARAM_EPMRK`
C++ Name	`EpMrk`
Java Name	`EpMrk`
.NET Name	`EpMrk`
InteractiveOptimizer	`simplex tolerances markowitz`
Identifier	`1013`

Description Influences pivot selection during basis factoring. Increasing the Markowitz threshold may improve the numerical properties of the solution.

Values Any number from 0.0001 to 0.99999; **default**: 0.01.

Summary Optimality tolerance

C Name `CPX_PARAM_EPOPT`

C++ Name `EpOpt`

Java Name `EpOpt`

.NET Name `EpOpt`

InteractiveOptimizer `simplex tolerances optimality`

Identifier `1014`

Description Influences the reduced-cost tolerance for optimality. This parameter governs how closely CPLEX must approach the theoretically optimal solution.

Values Any number from 1e-9 to 1e-1; **default**: 1e-06.

Summary Perturbation constant

C Name `CPX_PARAM_EPPER`

C++ Name `EpPer`

Java Name `EpPer`

.NET Name `EpPer`

InteractiveOptimizer `simplex perturbation`

Identifier `1015`

Description Sets the amount by which CPLEX perturbs the upper and lower bounds or objective coefficients on the variables when a problem is perturbed in the simplex algorithm. This parameter can be set to a smaller value if the default value creates too large a change in the problem.

Values Any positive number greater than or equal to 1e-8; **default**: 1e-6.

Summary Relaxation for `feasOpt`

C Name `CPX_PARAM_EPRELAX`

C++ Name `EpRelax`

Java Name `EpRelax`

.NET Name `EpRelax`

InteractiveOptimizer `feasopt tolerance`

Identifier `2073`

Description Controls the amount of relaxation for the routine `CPXfeasopt` in the C API or for the method `feasOpt` in the object-oriented APIs.

In the case of a MIP, it serves the purpose of the absolute gap for the `feasOpt` model in Phase I (the phase to minimize relaxation).

Using this parameter, you can implement other stopping criteria as well. To do so, first call `feasOpt` with the stopping criteria that you prefer; then set this parameter to the resulting objective of the Phase I model; unset the other stopping criteria, and call `feasOpt` again. Since the solution from the first call already matches this parameter, Phase I will terminate immediately in this second call to `feasOpt`, and Phase II will start.

In the case of an LP, this parameter controls the lower objective limit for Phase I of `feasOpt` and is thus relevant only when the primal optimizer is in use.

Values Any nonnegative value; **default**: 1e-6.

See Also `CPX_PARAM_OBJLLIM`, `ObjLLim`

Summary	Feasibility tolerance
C Name	`CPX_PARAM_EPRHS`
C++ Name	`EpRHS`
Java Name	`EpRHS`
.NET Name	`EpRHS`
InteractiveOptimizer	`simplex tolerances feasibility`
Identifier	`1016`

Description

Specifies the feasibility tolerance, that is, the degree to which the basic variables of a model may violate their bounds. Feasibility influences the selection of an optimal basis and can be reset to a higher value when a problem is having difficulty maintaining feasibility during optimization. You may also wish to lower this tolerance after finding an optimal solution if there is any doubt that the solution is truly optimal. If the feasibility tolerance is set too low, CPLEX may falsely conclude that a problem is infeasible. If you encounter reports of infeasibility during Phase II of the optimization, a small adjustment in the feasibility tolerance may improve performance.

Values

Any number from 1e-9 to 1e-1; **default**: 1e-06.

Summary	Mode of FeasOpt
C Name	`CPX_PARAM_FEASOPTMODE`
C++ Name	`FeasOptMode`
Java Name	`FeasOptMode`
.NET Name	`FeasOptMode`
InteractiveOptimizer	`feasopt mode`
Identifier	`1084`

Description

Determines how FeasOpt measures the relaxation when finding a minimal relaxation in an infeasible model. FeasOpt works in two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution among those that require only as much relaxation as it found necessary in the first phase. Values of this parameter indicate two aspects to CPLEX:

- whether to stop in phase one or continue to phase two and
- how to measure the relaxation, according to one of the following criteria:
  - as a sum of required relaxations;
  - as the number of constraints and bounds required to be relaxed;
  - as a sum of the squares of required relaxations.

Values

Value	Symbol	Symbol (C API)	Meaning
0	MinSum	CPX_FEASOPT_MIN_SUM	Minimize the sum of all required relaxations in first phase only; **default**
1	OptSum	CPX_FEASOPT_OPT_SUM	Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations
2	MinInf	CPX_FEASOPT_MIN_INF	Minimize the number of constraints and bounds requiring relaxation in first phase only
3	OptInf	CPX_FEASOPT_OPT_INF	Minimize the number of constraints and bounds requiring relaxation in first phase and execute second phase to find optimum among minimal relaxations
4	MinQuad	CPX_FEASOPT_MIN_QUAD	Minimize the sum of squares of required relaxations in first phase only
5	OptQuad	CPX_FEASOPT_OPT_QUAD	Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations

Summary MIP flow cover cuts switch

C Name `CPX_PARAM_FLOWCOVERS`

C++ Name `FlowCovers`

Java Name `FlowCovers`

.NET Name `FlowCovers`

InteractiveOptimizer `mip cuts flowcovers`

Identifier `2040`

Description Determines whether or not to generate flow cover cuts for the problem. Setting the value to 0 (zero), the default, indicates that the attempt to generate flow cover cuts should continue only if it seems to be helping.

Values

Vaue	Meaning
-1	Do not generate flow cover cuts
0	Automatic: let CPLEX choose; **default**
1	Generate flow cover cuts moderately
2	Generate flow cover cuts aggressively

Summary MIP flow path cut switch

C Name `CPX_PARAM_FLOWPATHS`

C++ Name `FlowPaths`

Java Name `FlowPaths`

.NET Name `FlowPaths`

InteractiveOptimizer `mip cuts pathcut`

Identifier `2051`

Description Determines whether or not flow path cuts should be generated for the problem. Setting the value to 0 (zero), the default, indicates that the attempt to generate flow path cuts should continue only if it seems to be helping.

Values

Value	Meaning
-1	Do not generate flow path cuts
0	Automatic: let CPLEX choose; **default**
1	Generate flow path cuts moderately
2	Generate flow path cuts aggressively

Summary	Candidate limit for generating Gomory fractional cuts
C Name	`CPX_PARAM_FRACCAND`
C++ Name	`FracCand`
Java Name	`FracCand`
.NET Name	`FracCand`
InteractiveOptimizer	`mip limits gomorycand`
Identifier	`2048`
Description	Limits the number of candidate variables for generating Gomory fractional cuts.
Values	Any positive integer; **default**: 200.

Summary MIP Gomory fractional cuts switch

C Name `CPX_PARAM_FRACCUTS`

C++ Name `FracCuts`

Java Name `FracCuts`

.NET Name `FracCuts`

InteractiveOptimizer `mip cuts gomory`

Identifier `2049`

Description Determines whether or not Gomory fractional cuts should be generated for the problem. Setting the value to 0 (zero), the default, indicates that the attempt to generate Gomory fractional cuts should continue only if it seems to be helping.

Values

Value	Meaning
-1	Do not generate Gomory fractional cuts
0	Automatic: let CPLEX choose; **default**
1	Generate Gomory fractional cuts moderately
2	Generate Gomory fractional cuts aggressively

Summary Pass limit for generating Gomory fractional cuts

C Name `CPX_PARAM_FRACPASS`

C++ Name `FracPass`

Java Name `FracPass`

.NET Name `FracPass`

InteractiveOptimizer `mip limits gomorypass`

Identifier `2050`

Description Limits the number of passes for generating Gomory fractional cuts. At the default setting of 0 (zero), CPLEX decides the number of passes to make. The parameter is ignored if the Gomory fractional cut parameter (`CPX_PARAM_FRACCUTS`, `FracCuts`) is set to a nonzero value.

Values

Value	Meaning
0	Automatic: let CPLEX choose; **default**
Any positive integer	Number of passes to generate Gomory fractional cuts

Summary MIP GUB cuts switch

C Name `CPX_PARAM_GUBCOVERS`

C++ Name `GUBCovers`

Java Name `GUBCovers`

.NET Name `GUBCovers`

InteractiveOptimizer `mip cuts gubcovers`

Identifier `2044`

Description Determines whether or not to generate GUB cuts for the problem. Setting the value to 0 (zero), the default, indicates that the attempt to generate GUB cuts should continue only if it seems to be helping.

Values

Value	Meaning
-1	Do not generate GUB cuts
0	Automatic: let CPLEX choose; **default**
1	Generate GUB cuts moderately
2	Generate GUB cuts aggressively

Summary MIP heuristic frequency

C Name `CPX_PARAM_HEURFREQ`

C++ Name `HeurFreq`

Java Name `HeurFreq`

.NET Name `HeurFreq`

InteractiveOptimizer `mip strategy heuristicfreq`

Identifier `2031`

Description Determines how often to apply the periodic heuristic. Setting the value to -1 turns off the periodic heuristic. Setting the value to 0 (zero), the default, applies the periodic heuristic at an interval chosen automatically. Setting the value to a positive number applies the heuristic at the requested node interval. For example, setting this parameter to 20 dictates that the heuristic be called at node 0, 20, 40, 60, etc.

Values

Value	Meaning
-1	None
0	Automatic: let CPLEX choose; **default**
Any positive integer	Apply the periodic heuristic at this frequency

Summary MIP implied bound cuts switch

C Name `CPX_PARAM_IMPLBD`

C++ Name `ImplBd`

Java Name `ImplBd`

.NET Name `ImplBd`

InteractiveOptimizer `mip cuts implied`

Identifier `2041`

Description Determines whether or not to generate implied bound cuts for the problem. Setting the value to 0 (zero), the default, indicates that the attempt to generate implied bound cuts should continue only if it seems to be helping.

Values

Value	Meaning
-1	Do not generate implied bound cuts
0	Automatic: let CPLEX choose; **default**
1	Generate implied bound cuts moderately
	Generate implied bound cuts aggressively

Summary MIP solution limit

C Name `CPX_PARAM_INTSOLLIM`

C++ Name `IntSolLim`

Java Name `IntSolLim`

.NET Name `IntSolLim`

InteractiveOptimizer `mip limits solutions`

Identifier `2015`

Description Sets the number of MIP solutions to be found before stopping.

Values Any positive integer; **default**: 2 100 000 000.

Summary Simplex maximum iteration limit

C Name `CPX_PARAM_ITLIM`

C++ Name `ItLim`

Java Name `ItLim`

.NET Name `ItLim`

InteractiveOptimizer `simplex limits iterations`

Identifier `1020`

Description Sets the maximum number of simplex iterations to be performed before the algorithm terminates without reaching optimality. When set to 0 (zero), no simplex method iteration occurs. However, CPLEX factors the initial basis from which solution routines provide information about the associated initial solution.

Values Any nonnegative integer; **default**: 2 100 000 000.

Summary Local branching heuristic

C Name `CPX_PARAM_LBHEUR (int)`

C++ Name `LBHeur (bool)`

Java Name `LBHeur`

.NET Name `LBHeur`

InteractiveOptimizer `mip strategy lbheur`

Identifier `2063`

Description Controls whether CPLEX applies a local branching heuristic to try to improve new incumbents found during a MIP search. By default, this parameter is off. If you turn it on, CPLEX will invoke a local branching heuristic only when it finds a new incumbent. If CPLEX finds multiple incumbents at a single node, the local branching heuristic will be applied only to the last one found.

Values

Value	bool	Symbol	Meaning
0	false	`CPX_OFF`	Local branching heuristic is off; **default**
1	true	`CPX_ON`	Apply local branching heuristic to new incumbent

Summary Reduces use of memory

C Name `CPX_PARAM_MEMORYEMPHASIS (int)`

C++ Name `MemoryEmphasis (bool)`

Java Name `MemoryEmphasis`

.NET Name `MemoryEmphasis`

InteractiveOptimizer `emphasis memory`

Identifier `1082`

Description Directs CPLEX that it should conserve memory where possible. When you set this parameter to its nondefault value, CPLEX will choose tactics, such as data compression or disk storage, for some of the data computed by the simplex, barrier, and MIP optimizers. Of course, conserving memory may impact performance in some models. Also, while solution information will be available after optimization, certain computations that require a basis that has been factored (for example, for the computation of the condition number Kappa) may be unavailable.

Values

Value	bool	Symbol	Meaning
0	false	`CPX_OFF`	Off; do not conserve memory; **default**
1	true	`CPX_ON`	On; conserve memory where possible

Summary MIP callback switch between original model and reduced, presolved model

C Name `CPX_PARAM_MIPCBREDLP`

C++ Name `MIP callback reduced LP parameter not available in this API`

Java Name `not available`

.NET Name `not available`

InteractiveOptimizer `not available`

Identifier `2055`

Description Controls whether your callback accesses node information of the original model (off) or node information of the reduced, presolved model (on, default). Advanced routines to control MIP callbacks (such as `CPXgetcallbacklp`, `CPXsetheuristiccallbackfunc`, `CPXsetbranchcallbackfunc`, `CPXgetbranchcallbackfunc`, `CPXsetcutcallbackfunc`, `CPXsetincumbentcallbackfunc`, `CPXgetcallbacksosinfo`, `CPXcutcallbackadd`, `CPXcutcallbackaddlocal`, and others with the prefix `CPXgetcallback`) consider the setting of this parameter and access the original model or the reduced, presolved model accordingly.

The routine `CPXgetcallbacknodelp` is an exception: it always accesses the current node LP associated with the presolved model, regardless of the setting of this parameter.

For certain routines, such as `CPXcutcallbackadd`, when you set the parameter `CPX_PARAM_MIPCBREDLP` to zero, you should also set `CPX_PARAM_PRELINEAR` to zero as well.

In the C++, Java, and .NET APIs of CPLEX, only the original model is available to callbacks. In other words, this parameter is effective only for certain advanced routines of the C API.

Values

Value	Symbol	Meaning
0	`CPX_OFF`	Off: use original model
1	`CPX_ON`	On: use reduced, presolved model; **default**

Summary MIP node log display information

C Name `CPX_PARAM_MIPDISPLAY`

C++ Name `MIPDisplay`

Java Name `MIPDisplay`

.NET Name `MIPDisplay`

InteractiveOptimizer `mip display`

Identifier `2012`

Description Determines what CPLEX reports to the screen during mixed integer optimization (MIP). The amount of information displayed increases with increasing values of this parameter. A setting of 0 (zero) causes no node log to be displayed until the optimal solution is found. A setting of 1 (one) displays an entry for each integer feasible solution found. Each entry contains the value of the objective function, the node count, the number of unexplored nodes in the tree, and the current optimality gap. A setting of 2 also generates an entry for every n-th node (where n is the setting of the `MIPInterval` parameter). A setting of 3 additionally generates an entry for every n-th node giving the number of cuts added to the problem for the previous `MIPInterval` number of nodes. A setting of 4 additionally generates entries for the LP root relaxation according to the setting of the simplex display. A setting of 5 additionally generates entries for the LP subproblems, also according to the setting of the simplex display.

Values

Value	Meaning
0	No display until optimal solution has been found
1	Display integer feasible solutions
2	Display integer feasible solutions plus an entry for every n-th node; **default**
3	Display integer feasible solutions, every n-th node entry, and number of cuts added
4	Display integer feasible solutions, every n-th node entry, number of cuts added, and information about the LP subproblem at root
5	Display integer feasible solutions, every n-th node entry, number of cuts added, and information about the LP subproblem at root and at nodes

See Also `CPX_PARAM_MIPINTERVAL`, `MIPInterval` *and* `CPX_PARAM_SIMDISPLAY`, `SimDisplay`, *and* `CPX_PARAM_NETDISPLAY`, `NetDisplay`, *and* `CPX_PARAM_SCRIND`

Summary MIP emphasis switch

C Name `CPX_PARAM_MIPEMPHASIS`

C++ Name `MIPEmphasis`

Java Name `MIPEmphasis`

.NET Name `MIPEmphasis`

InteractiveOptimizer `emphasis mip`

Identifier `2058`

Description Controls trade-offs between speed, feasibility, optimality, and moving bounds in MIP.

With the default setting of BALANCED, CPLEX works toward a rapid proof of an optimal solution, but balances that with effort toward finding high quality feasible solutions early in the optimization.

When this parameter is set to FEASIBILITY, CPLEX frequently will generate more feasible solutions as it optimizes the problem, at some sacrifice in the speed to the proof of optimality.

When set to OPTIMALITY, less effort may be applied to finding feasible solutions early.

With the setting BESTBOUND, even greater emphasis is placed on proving optimality through moving the best bound value, so that the detection of feasible solutions along the way becomes almost incidental.

When the parameter is set to HIDDENFEAS, the MIP optimizer works hard to find high quality feasible solutions that are otherwise very difficult to find, so consider this setting when the FEASIBILITY setting has difficulty finding solutions of acceptable quality.

Values

Value	Symbol	Meaning
0	`CPX_MIPEMPHASIS_BALANCED`	Balance optimality and feasibility; **default**
1	`CPX_MIPEMPHASIS_FEASIBILITY`	Emphasize feasibility over optimality
2	`CPX_MIPEMPHASIS_OPTIMALITY`	Emphasize optimality over feasibility
3	`CPX_MIPEMPHASIS_BESTBOUND`	Emphasize moving best bound
4	`CPX_MIPEMPHASIS_HIDDENFEAS`	Emphasize finding hidden feasible solutions

Summary MIP node log interval

C Name `CPX_PARAM_MIPINTERVAL`

C++ Name `MIPInterval`

Java Name `MIPInterval`

.NET Name `MIPInterval`

InteractiveOptimizer `mip interval`

Identifier `2013`

Description Controls the frequency of node logging when the MIP display parameter (`CPX_PARAM_MIPDISPLAY`, `MIPDisplay`) is set higher than 1 (one).

Values Any positive integer; **default**: 100.

See Also `CPX_PARAM_MIPDISPLAY`, `MIPDisplay`

Summary MIP priority order switch

C Name `CPX_PARAM_MIPORDIND (int)`

C++ Name `MIPOrdInd (bool)`

Java Name `MIPOrdInd`

.NET Name `MIPOrdInd`

InteractiveOptimizer `mip strategy order`

Identifier `2020`

Description Determines whether to use the priority order, if one exists, for the next mixed integer optimization.

Values

Value	bool	Symbol	Meaning
	false	`CPX_OFF`	Off: do not use priority order
	true	`CPX_ON`	On: use priority order, if it exists; **default**

Summary MIP priority order generation

C Name CPX_PARAM_MIPORDTYPE

C++ Name MIPOrdType

Java Name MIPOrdType

.NET Name MIPOrdType

InteractiveOptimizer mip ordertype

Identifier 2032

Description Selects the type of generic priority order to generate when no priority order is present.

Values

Value	Symbol	Meaning
0		Do not generate a priority order
1	CPX_MIPORDER_COST	Use decreasing cost
2	CPX_MIPORDER_BOUNDS	Use increasing bound range
3	CPX_MIPORDER_SCALEDCOST	Use increasing cost per coefficient count

Summary MIP thread limit

C Name `CPX_PARAM_MIPTHREADS`

C++ Name `MIPThreads`

Java Name `MIPThreads`

.NET Name `MIPThreads`

InteractiveOptimizer `mip limits threads`

Identifier `2014`

Description Determines the maximum number of parallel processes (threads) that will be invoked by the Parallel MIP optimizer. The default value of 0 (zero) means that the limit will be determined by the value of the global thread limit parameter (`CPX_PARAM_THREADS`, `Threads`). A positive value will override the value of the global thread limit parameter.

Values

Value	Meaning
0 **default**	MIP thread limit determined by global thread limit
Any positive integer greater than or equal to 1	Upper limit on threads for Parallel MIP

See Also `CPX_PARAM_THREADS`, `Threads`

Summary MIP MIR (mixed integer rounding) cut switch

C Name `CPX_PARAM_MIRCUTS`

C++ Name `MIRCuts`

Java Name `MIRCuts`

.NET Name `MIRCuts`

InteractiveOptimizer `mip cuts mircut`

Identifier `2052`

Description Determines whether or not to generate MIR cuts (mixed integer rounding cuts) for the problem. Setting the value to 0 (zero), the default, indicates that the attempt to generate MIR cuts should continue only if it seems to be helping.

Values

Value	Meaning
-1	Do not generate MIR cuts
0	Automatic: let CPLEX choose; **default**
1	Generate MIR cuts moderately
2	Generate MIR cuts aggressively

Summary Precision of numerical output in MPS and REW file formats

C Name `CPX_PARAM_MPSLONGNUM (int)`

C++ Name `MPSLongNum (bool)`

Java Name `MPSLongNum`

.NET Name `MPSLongNum`

InteractiveOptimizer `output mpslong`

Identifier `1081`

Description Determines the precision of numerical output in the MPS and REW file formats. When this parameter is set to its default value 1 (one), numbers are written to MPS files in full-precision; that is, up to 15 significant digits may be written. The setting 0 (zero) writes files that correspond to the standard MPS format, where at most 12 characters can be used to represent a value. This limit may result in loss of precision.

Values

Value	bool	Symbol	Meaning
0	false	`CPX_OFF`	Off: use limited MPS precision
1	true	`CPX_ON`	On: use full-precision; **default**

See Also *MPS File Format on page 14 in the File Format Reference Manual for more details about that topic.*

Summary Network logging display switch

C Name `CPX_PARAM_NETDISPLAY`

C++ Name `NetDisplay`

Java Name `NetDisplay`

.NET Name `NetDisplay`

InteractiveOptimizer `network display`

Identifier `5005`

Description Determines what CPLEX reports to the screen during network optimization. Settings 1 and 2 differ only during Phase I. Setting 2 shows monotonic values, whereas 1 usually does not.

Values

Value	Symbol	Meaning
0	`CPXNET_NO_DISPLAY_OBJECTIVE`	No display
1	`CPXNET_TRUE_OBJECTIVE`	Display true objective values
2	`CPXNET_PENALIZE_OBJECTIVE`	Display penalized objective values; **default**

Summary	Optimality tolerance for network optimization
C Name	`CPX_PARAM_NETEPOPT`
C++ Name	`NetEpOpt`
Java Name	`NetEpOpt`
.NET Name	`NetEpOpt`
InteractiveOptimizer	`network tolerances optimality`
Identifier	`5002`
Description	Specifies the optimality tolerance for network optimization; that is, the amount a reduced cost may violate the criterion for an optimal solution.
Values	Any number from 1e-11 to 1e-1; **default**: 1e-6.

Summary	Feasibility tolerance for network primal optimization
C Name	`CPX_PARAM_NETEPRHS`
C++ Name	`NetEpRHS`
Java Name	`NetEpRHS`
.NET Name	`NetEpRHS`
InteractiveOptimizer	`network tolerances feasibility`
Identifier	`5003`

Description

Specifies feasibility tolerance for network primal optimization. The feasibility tolerance specifies the degree to which the flow value of a model may violate its bounds. This tolerance influences the selection of an optimal basis and can be reset to a higher value when a problem is having difficulty maintaining feasibility during optimization. You may also wish to lower this tolerance after finding an optimal solution if there is any doubt that the solution is truly optimal. If the feasibility tolerance is set too low, CPLEX may falsely conclude that a problem is infeasible. If you encounter reports of infeasibility during Phase II of the optimization, a small adjustment in the feasibility tolerance may improve performance.

Values

Any number from 1e-11 to 1e-1; **default**: 1e-6.

Summary Simplex network extraction level

C Name `CPX_PARAM_NETFIND`

C++ Name `NetFind`

Java Name `NetFind`

.NET Name `NetFind`

InteractiveOptimizer `network netfind`

Identifier `1022`

Description Establishes the level of network extraction for network simplex optimization. The default value is suitable for recognizing commonly used modeling approaches when representing a network problem within an LP formulation.

Values

Value	Symbol	Meaning
1	`CPX_NETFIND_PURE`	Extract pure network only
2	`CPX_NETFIND_REFLECT`	Try reflection scaling; **default**
3	`CPX_NETFIND_SCALE`	Try general scaling

Summary Network simplex iteration limit

C Name `CPX_PARAM_NETITLIM`

C++ Name `NetItLim`

Java Name `NetItLim`

.NET Name `NetItLim`

InteractiveOptimizer `network iterations`

Identifier `5001`

Description Sets the maximum number of iterations to be performed before the algorithm terminates without reaching optimality.

Values Any nonnegative integer; **default**: 2 100 000 000.

Summary Network simplex pricing algorithm

C Name `CPX_PARAM_NETPPRIIND`

C++ Name `NetPPriInd`

Java Name `NetPPriInd`

.NET Name `NetPPriInd`

InteractiveOptimizer `network pricing`

Identifier `5004`

Description Specifies the pricing algorithm for network simplex optimization. The default (0) shows best performance for most problems, and currently is equivalent to 3.

Values

Value	Symbol	Meaning
0	`CPXNET_PRICE_AUTO`	Automatic: let CPLEX choose; **default**
1	`CPXNET_PRICE_PARTIAL`	Partial pricing
2	`CPXNET_PRICE_MULT_PART`	Multiple partial pricing
3	`CPXNET_PRICE_SORT_MULT_PART`	Multiple partial pricing with sorting

Summary MIP subproblem algorithm

C Name `CPX_PARAM_SUBALG`

C++ Name `NodeAlg`

Java Name `NodeAlg`

.NET Name `NodeAlg`

InteractiveOptimizer `mip strategy subalgorithm`

Identifier `2026`

Description Determines which continuous optimizer will be used to solve the subproblems in a MIP, after the initial relaxation.

The default Automatic setting (0 zero) of this parameter currently selects the dual simplex optimizer for subproblem solution for MILP and MIQP. The Automatic setting may be expanded in the future so that CPLEX chooses the algorithm based on additional characteristics of the model.

For MILP (integer constraints and otherwise continuous variable), all settings are permitted.

For MIQP (integer constraints and positive semi-definite quadratic terms in objective), setting 3 (Network) is not permitted, and setting 5 (Sifting) reverts to 0 (Automatic).

For MIQCP (integer constraints and positive semi-definite quadratic terms among the constraints), only the Barrier optimizer is implemented, and therefore no settings other than 0 (Automatic) and 4 (Barrier) are permitted.

Values

Value	Symbol	Meaning
0	`CPX_ALG_AUTOMATIC`	Automatic: let CPLEX choose; **default**
1	`CPX_ALG_PRIMAL`	Primal simplex
2	`CPX_ALG_DUAL`	Dual simplex
3	`CPX_ALG_NET`	Network simplex
4	`CPX_ALG_BARRIER`	Barrier
5	`CPX_ALG_SIFTING`	Sifting

Summary Node storage file switch

C Name `CPX_PARAM_NODEFILEIND`

C++ Name `NodeFileInd`

Java Name `NodeFileInd`

.NET Name `NodeFileInd`

InteractiveOptimizer `mip strategy file`

Identifier `2016`

Description Used when working memory (`CPX_PARAM_WORKMEM`, `WorkMem`) has been exceeded by the size of the tree. If the node file parameter is set to zero when the tree memory limit is reached, optimization is terminated. Otherwise, a group of nodes is removed from the in-memory set as needed. By default, CPLEX transfers nodes to node files when the in-memory set is larger than 128 MBytes, and it keeps the resulting node files in compressed form in memory. At settings 2 and 3, the node files are transferred to disk, in uncompressed and compressed form respectively, into a directory named by the working directory parameter (`CPX_PARAM_WORKDIR`, `WorkDir`), and CPLEX actively manages which nodes remain in memory for processing.

Values

Value	Meaning
0	No node file
1	Node file in memory and compressed; **default**
2	Node file on disk
3	Node file on disk and compressed

See Also `CPX_PARAM_WORKMEM`, `WorkMem` *and* `CPX_PARAM_WORKDIR`, `WorkDir`

Summary MIP node limit

C Name `CPX_PARAM_NODELIM`

C++ Name `NodeLim`

Java Name `NodeLim`

.NET Name `NodeLim`

InteractiveOptimizer `mip limits nodes`

Identifier `2017`

Description Sets the maximum number of nodes solved before the algorithm terminates without reaching optimality. When this parameter is set to 0 (zero), CPLEX completes processing at the root; that is, it creates cuts and applies heuristics at the root. When this parameter is set to 1 (one), it allows branching from the root; that is, nodes are created but not solved.

Values Any nonnegative integer; **default**: 2 100 000 000.

Summary MIP node selection strategy

C Name `CPX_PARAM_NODESEL`

C++ Name `NodeSel`

Java Name `NodeSel`

.NET Name `NodeSel`

InteractiveOptimizer `mip strategy nodeselect`

Identifier `2018`

Description Used to set the rule for selecting the next node to process when backtracking. The depth-first search strategy chooses the most recently created node. The best-bound strategy chooses the node with the best objective function for the associated LP relaxation. The best-estimate strategy selects the node with the best estimate of the integer objective value that would be obtained from a node once all integer infeasibilities are removed. An alternative best-estimate search is also available.

Values

Value	Symbol	Meaning
0	`CPX_NODESEL_DFS`	Depth-first search
1	`CPX_NODESEL_BESTBOUND`	Best-bound search; **default**
2	`CPX_NODESEL_BESTEST`	Best-estimate search
3	`CPX_NODESEL_BESTEST_ALT`	Alternative best-estimate search

Summary Numerical precision emphasis

C Name `CPX_PARAM_NUMERICALEMPHASIS (int)`

C++ Name `NumericalEmphasis (bool)`

Java Name `NumericalEmphasis`

.NET Name `NumericalEmphasis`

InteractiveOptimizer `emphasis numerical`

Identifier `1083`

Description Emphasizes precision in numerically unstable or difficult problems. This parameter lets you indicate to CPLEX that it should emphasize precision in numerically difficult or unstable problems, with consequent performance trade-offs in time and memory.

Values

Value	bool	Symbol	Meaning
0	false	`CPX_OFF`	Do not emphasize numerical precision; **default**
1	true	`CPX_ON`	Exercise extreme caution in computation

Summary Nonzero element read limit

C Name `CPX_PARAM_NZREADLIM`

C++ Name `NzReadLim`

Java Name `NzReadLim`

.NET Name `NzReadLim`

InteractiveOptimizer `read nonzeros`

Identifier `1024`

Description Specifies a limit for the number of nonzero elements to read for an allocation of memory. This parameter does not restrict the size of a problem. Rather, it indirectly specifies the default amount of memory that will be pre-allocated before a problem is read from a file. If the limit is exceeded, more memory is automatically allocated.

Values Any integer from 0 to 268 435 450; **default**: 250 000.

Summary Absolute objective difference cutoff

C Name `CPX_PARAM_OBJDIF`

C++ Name `ObjDif`

Java Name `ObjDif`

.NET Name `ObjDif`

InteractiveOptimizer `mip tolerances objdifference`

Identifier `2019`

Description Used to update the cutoff each time a mixed integer solution is found. This absolute value is subtracted from (added to) the newly found integer objective value when minimizing (maximizing). This forces the mixed integer optimization to ignore integer solutions that are not at least this amount better than the best one found so far.

The objective difference parameter can be adjusted to improve problem solving efficiency by limiting the number of nodes; however, setting this parameter at a value other than zero (the default) can cause some integer solutions, including the true integer optimum, to be missed.

Negative values for this parameter can result in some integer solutions that are worse than or the same as those previously generated, but does not necessarily result in the generation of all possible integer solutions.

Values Any number; **default**: 0.0.

See Also `CPX_PARAM_RELOBJDIF`, `RelObjDif`

Summary	Lower objective value limit
C Name	`CPX_PARAM_OBJLLIM`
C++ Name	`ObjLLim`
Java Name	`ObjLLim`
.NET Name	`ObjLLim`
InteractiveOptimizer	`simplex limits lowerobj`
Identifier	`1025`
Description	Sets a lower limit on the value of the objective function in the simplex algorithms. Setting a lower objective function limit causes CPLEX to halt the optimization process once the minimum objective function value limit has been exceeded. This limit applies only during Phase II of the simplex algorithm.
Values	Any number; **default**: -1e+75.

Summary	Upper objective value limit
C Name	`CPX_PARAM_OBJULIM`
C++ Name	`ObjULim`
Java Name	`ObjULim`
.NET Name	`ObjULim`
InteractiveOptimizer	`simplex limits upperobj`
Identifier	`1026`
Description	Setting an upper objective function limit causes CPLEX to halt the optimization process once the maximum objective function value limit has been exceeded. This limit applies only during Phase II of the simplex algorithm.
Values	Any number; **default**: 1e+75.

Summary Simplex perturbation switch

C Name `CPX_PARAM_PERIND (int)`

C++ Name `PerInd (bool)`

Java Name `PerInd`

.NET Name `PerInd`

InteractiveOptimizer `simplex perturbation`

Identifier `1027`

Description Setting this parameter to 1 (one) causes all problems to be automatically perturbed as optimization begins. A setting of 0 (zero) allows CPLEX to determine dynamically, during solution, whether progress is slow enough to merit a perturbation. The situations in which a setting of 1 helps are rare and restricted to problems that exhibit extreme degeneracy.

Values

Value	bool	Symbol	Meaning
0	false	`CPX_OFF`	Automatic: let CPLEX choose; **default**
1	true	`CPX_ON`	Turn on perturbation from beginning

Summary Simplex perturbation limit

C Name `CPX_PARAM_PERLIM`

C++ Name `PerLim`

Java Name `PerLim`

.NET Name `PerLim`

InteractiveOptimizer `simplex limits perturbation`

Identifier `1028`

Description Sets the number of degenerate iterations before perturbation is performed.

Values

Value	Meaning
o	Automatic: let CPLEX choose; default
Any positive integer	Number of degenerate iterations before perturbation

Summary Time spent polishing a solution

C Name `CPX_PARAM_POLISHTIME`

C++ Name `PolishTime`

Java Name `PolishTime`

.NET Name `PolishTime`

InteractiveOptimizer `mip limit polishtime`

Identifier `2066`

Description Tells CPLEX how much time in seconds to spend after a normal mixed integer optimization in polishing a solution. Default is zero, no polishing time.

Values Any nonnegative value in seconds; **default**: 0.0 (zero) seconds.

Summary Primal simplex pricing algorithm

C Name `CPX_PARAM_PPRIIND`

C++ Name `PPriInd`

Java Name `PPriInd`

.NET Name `PPriInd`

InteractiveOptimizer `simplex pgradient`

Identifier `1029`

Description Determines the primal simplex pricing algorithm. The default pricing (0) usually provides the fastest solution time, but many problems benefit from alternative settings.

Values

Value	Symbol	Meaning
-1	`CPX_PPRIIND_PARTIAL`	Reduced-cost pricing
0	`CPX_PPRIIND_AUTO`	Hybrid reduced-cost & devex pricing; **default**
1	`CPX_PPRIIND_DEVEX`	Devex pricing
2	`CPX_PPRIIND_STEEP`	Steepest-edge pricing
3	`CPX_PPRIIND_STEEPQSTART`	Steepest-edge pricing with slack initial norms
4	`CPX_PPRIIND_FULL`	Full pricing

Summary Presolve dual setting

C Name `CPX_PARAM_PREDUAL`

C++ Name `PreDual`

Java Name `PreDual`

.NET Name `PreDual`

InteractiveOptimizer `preprocessing dual`

Identifier `1044`

Description Determines whether CPLEX presolve should pass the primal or dual linear programming problem to the linear programming optimization algorithm. By default, CPLEX chooses automatically.

If this parameter is set to 1 (one), the CPLEX presolve algorithm is applied to the primal problem, but the resulting dual linear program is passed to the optimizer. This is a useful technique for problems with more constraints than variables.

Values

Value	Meaning
-1	Turn off this feature
0	Automatic: let CPLEX choose; **default**
1	Turn on this feature

Summary Presolve switch

C Name `CPX_PARAM_PREIND (int)`

C++ Name `PreInd (bool)`

Java Name `PreInd`

.NET Name `PreInd`

InteractiveOptimizer `preprocessing presolve`

Identifier `1030`

Description Determines whether CPLEX applies presolve during preprocessing. When set to 1 (one), the default, this parameter invokes the CPLEX presolve to simplify and reduce problems.

Values

Value	bool	Symbol	Meaning
0	false	`CPX_OFF`	Do not apply presolve
1	true	`CPX_ON`	Apply presolve; **default**

Summary Linear reduction switch

C Name `CPX_PARAM_PRELINEAR`

C++ Name `PreLinear`

Java Name `PreLinear`

.NET Name `PreLinear`

InteractiveOptimizer `preprocessing linear`

Identifier `1058`

Description Determines whether linear or full reductions occur during preprocessing. If only linear reductions are performed, each variable in the original model can be expressed as a linear form of variables in the presolved model. This condition guarantees, for example, that users can add their own custom cuts to the presolved model.

Values

Value	Meaning
0	Perform only linear reductions
1	Perform full reductions; **default**

Summary Limit on the number of presolve passes made

C Name `CPX_PARAM_PREPASS`

C++ Name `PrePass`

Java Name `PrePass`

.NET Name `PrePass`

InteractiveOptimizer `preprocessing numpass`

Identifier `1052`

Description Limits the number of presolve passes that CPLEX makes during preprocessing. When this parameter is set to a nonzero value, invokes CPLEX presolve to simplify and reduce problems.

When this parameter is set to a positive value, presolve is applied the specified number of times, or until no more reductions are possible.

At the default value of -1, presolve should continue only if it seems to be helping.

When this parameter is set to zero, CPLEX does not apply presolve, but other reductions may occur, depending on settings of other parameters and specifics of your model.

Values

Value	Meaning
-1	Automatic: let CPLEX choose; presolve continues as long as helpful; **default**
0	Do not use presolve; other reductions may still occur
Any positive integer	Apply presolve specified number of times

Summary Node presolve selector

C Name `CPX_PARAM_PRESLVND`

C++ Name `PreslvNd`

Java Name `PreslvNd`

.NET Name `PreslvNd`

InteractiveOptimizer `mip strategy presolvenode`

Identifier `2037`

Description Determines whether node presolve should be performed at the nodes of a mixed integer programming (MIP) solution. Node presolve can significantly reduce solution time for some models. The default setting is generally effective at determining whether to apply node presolve, although runtimes can be reduced for some models by turning node presolve off.

Values

Value	Meaning
-1	No node presolve
0	Automatic: let CPLEX choose; **default**
1	Force presolve at nodes
2	Perform probing on integer-infeasible variables

Summary Simplex pricing candidate list size

C Name `CPX_PARAM_PRICELIM`

C++ Name `PriceLim`

Java Name `PriceLim`

.NET Name `PriceLim`

InteractiveOptimizer `simplex pricing`

Identifier `1010`

Description Sets the maximum number of variables kept in the list of pricing candidates for the simplex algorithms.

Values

Value	Meaning
0	Automatic: let CPLEX choose; **default**
Any positive integer	Number of pricing candidates

Summary MIP probing level

C Name `CPX_PARAM_PROBE`

C++ Name `Probe`

Java Name `Probe`

.NET Name `Probe`

InteractiveOptimizer `mip strategy probe`

Identifier `2042`

Description Determines the amount of probing on variables to be performed before MIP branching. Higher settings perform more probing. Probing can be very powerful but very time-consuming at the start. Setting the parameter to values above the default of 0 (automatic) can result in dramatic reductions or dramatic increases in solution time, depending on the model.

Values

Value	Meaning
-1	No probing
0	Automatic: let CPLEX choose; **default**
1	Moderate probing level
2	Aggressive probing level
3	Very aggressive probing level

Summary	Time spent probing
C Name	`CPX_PARAM_PROBETIME`
C++ Name	`ProbeTime`
Java Name	`ProbeTime`
.NET Name	`ProbeTime`
InteractiveOptimizer	`mip limit probetime`
Identifier	`2065`
Description	Limits the amount of time in seconds spent probing.
Values	Any nonnegative number; **default**: 1e+75.

Summary Indefinite MIQP switch

C Name `CPX_PARAM_QPMAKEPSDIND (int)`

C++ Name `QPmakePSDInd (bool)`

Java Name `QPmakePSDInd`

.NET Name `QPmakePSDInd`

InteractiveOptimizer `preprocessing qpmakepsd`

Identifier `4010`

Description Determines whether CPLEX will attempt to reformulate a MIQP or MIQCP model that contains only binary variables. When this feature is active, adjustments will be made to the elements of a quadratic matrix that is not nominally positive semi-definite (PSD, as required by CPLEX for all QP and most QCP formulations), to make it PSD, and CPLEX will also attempt to tighten an already PSD matrix for better numerical behavior. The default setting of 1 (one) means yes, CPLEX should attempt to reformulate, but you can turn it off if necessary; most models should benefit from the default setting.

Values

Value	bool	Symbol	Meaning
0	false	`CPX_OFF`	Turn off attempts to make binary model PSD
1	true	`CPX_ON`	On: CPLEX attempts to make binary model PSD; **default**

Summary	QP Q matrix nonzero read limit
C Name	`CPX_PARAM_QPNZREADLIM`
C++ Name	`QPNzReadLim`
Java Name	`QPNzReadLim`
.NET Name	`QPNzReadLim`
InteractiveOptimizer	`read qpnonzeros`
Identifier	`4001`

Description

Specifies a limit for the number of nonzero elements to read for an allocation of memory in a model with a quadratic matrix.

This parameter does not restrict the size of a problem. Rather, it indirectly specifies the default amount of memory that will be pre-allocated before a problem is read from a file. If the limit is exceeded, more memory is automatically allocated.

Values

Any integer from 0 to 268 435 450; **default**: 5 000.

Summary Primal and dual reduction type

C Name `CPX_PARAM_REDUCE`

C++ Name `Reduce`

Java Name `Reduce`

.NET Name `Reduce`

InteractiveOptimizer `preprocessing reduce`

Identifier `1057`

Description Determines whether primal reductions, dual reductions, both, or neither are performed during preprocessing.

Values

Value	Symbol	Meaning
0	`CPX_PREREDUCE_NOPRIMALORDUAL`	No primal or dual reductions
1	`CPX_PREREDUCE_PRIMALONLY`	Only primal reductions
2	`CPX_PREREDUCE_DUALONLY`	Only dual reductions
3	`CPX_PREREDUCE_PRIMALANDDUAL`	Both primal and dual reductions; **default**

Summary Simplex refactoring frequency

C Name `CPX_PARAM_REINV`

C++ Name `ReInv`

Java Name `ReInv`

.NET Name `ReInv`

InteractiveOptimizer `simplex refactor`

Identifier `1031`

Description Sets the number of iterations between refactoring of the basis matrix.

Values

Value	**Meaning**
0	Automatic: let CPLEX choose; **default**
Integer from 1 to 10 000	Number of iterations between refactoring of the basis matrix

Summary Relaxed LP presolve switch

C Name `CPX_PARAM_RELAXPREIND`

C++ Name `RelaxPreInd`

Java Name `RelaxPreInd`

.NET Name `RelaxPreInd`

InteractiveOptimizer `preprocessing relax`

Identifier `2034`

Description Determines whether LP presolve is applied to the root relaxation in a mixed integer program (MIP). Sometimes additional reductions can be made beyond any MIP presolve reductions that were already done. By default, CPLEX applies presolve to the initial relaxation in order to hasten time to the initial solution.

Values

Value	Symbol	Meaning
-1		Automatic: let CPLEX choose; **default**
0	`CPX_OFF`	Off: do not use presolve on initial relaxation
1	`CPX_ON`	On: use presolve on initial relaxation

Summary Relative objective difference cutoff

C Name `CPX_PARAM_RELOBJDIF`

C++ Name `RelObjDif`

Java Name `RelObjDif`

.NET Name `RelObjDif`

InteractiveOptimizer `mip tolerances relobjdifference`

Identifier `2022`

Description

Used to update the cutoff each time a mixed integer solution is found. The value is multiplied by the absolute value of the integer objective and subtracted from (added to) the newly found integer objective when minimizing (maximizing). This computation forces the mixed integer optimization to ignore integer solutions that are not at least this amount better than the one found so far.

The relative objective difference parameter can be adjusted to improve problem solving efficiency by limiting the number of nodes; however, setting this parameter at a value other than zero (the default) can cause some integer solutions, including the true integer optimum, to be missed.

If both the relative objective difference and the absolute objective difference (`CPX_PARAM_OBJDIF`, `ObjDif`) are nonzero, the value of the absolute objective difference is used.

Values Any number from 0.0 to 1.0; **default**: 0.0.

See Also `CPX_PARAM_OBJDIF`, `ObjDif`

Summary Frequency to try to repair infeasible MIP start

C Name `CPX_PARAM_REPAIRTRIES`

C++ Name `RepairTries`

Java Name `RepairTries`

.NET Name `RepairTries`

InteractiveOptimizer `mip limits repairtries`

Identifier `2067`

Description Limits the attempts to repair an infeasible MIP start. This parameter lets you tell CPLEX whether and how many times it should try to repair an infeasible MIP start that you supplied. The parameter has no effect if the MIP start you supplied is feasible. It has no effect if no MIP start was supplied.

Values

Value	Meaning
-1	None: do not try to repair
0	Automatic: let CPLEX choose; **default**
Any positive integer	Frequency to attempt repairs

Summary Reapply presolve after processing the root node

C Name `CPX_PARAM_REPEATPRESOLVE`

C++ Name `RepeatPresolve`

Java Name `RepeatPresolve`

.NET Name `RepeatPresolve`

InteractiveOptimizer `preprocessing repeatpresolve`

Identifier `2064`

Description Determines whether to re-apply presolve, with or without cuts, to a MIP model after processing at the root is otherwise complete.

Values

Value	Symbol
-1	Automatic: let CPLEX choose; **default**
0	Turn off represolve
1	Represolve without cuts
2	Represolve with cuts
3	Represovle with cuts and allow new root cuts

Summary RINS heuristic frequency

C Name `CPX_PARAM_RINSHEUR`

C++ Name `RINSHeur`

Java Name `RINSHeur`

.NET Name `RINSHeur`

InteractiveOptimizer `mip strategy rinsheur`

Identifier `2061`

Description Determines how often to apply the relaxation induced neighborhood search (RINS) heuristic. This heuristic attempts to improve upon the best solution found so far. It will not be applied until CPLEX has found at least one incumbent solution.

Setting the value to -1 turns off the RINS heuristic. Setting the value to 0 (zero), the default, applies the RINS heuristic at an interval chosen automatically by CPLEX. Setting the value to a positive number applies the RINS heuristic at the requested node interval. For example,setting RINSHeur to 20 dictates that the RINS heuristic be called at node 0, 20, 40, 60, etc.

RINS is a powerful heuristic for finding high quality feasible solutions, but it may be expensive.

Values

Value	Meaning
-1	None: do not apply RINS heuristic
0	Automatic: let CPLEX choose; **default**
Any positive integer	Frequency to apply RINS heuristic

Summary	Solution algorithm for continuous problems
C Name	`CPX_PARAM_LPMETHOD`
C++ Name	`RootAlg`
Java Name	`RootAlg`
.NET Name	`RootAlg`
InteractiveOptimizer	`lpmethod`
Identifier	`1062`

Description

Controls which algorithm is used to solve continuous models or to solve the root relaxation of a MIP. In the object-oriented APIs, you make this selection through the `RootAlg` parameter. In the C API and the Interactive Optimizer, there are separate parameters to control LP, QP, and MIP optimizers, depending on the problem type.

In all cases, the default setting is 0 (zero). The default setting means that CPLEX will select the algorithm in a way that should give best overall performance.

For specific problem classes, the following details document the automatic settings. Note that future versions of CPLEX could adopt different strategies. Therefore, if you select any nondefault settings, you should review them periodically.

Currently, the behavior of the automatic setting is that CPLEX almost always invokes the dual simplex algorithm when it is solving an LP model from scratch. When it is continuing from an advanced basis, it will check whether the basis is primal or dual feasible, and choose the primal or dual simplex algorithm accordingly.

If multiple threads have been requested, the concurrent optimization algorithm is selected by the automatic setting.

The automatic setting may be expanded in the future so that CPLEX chooses the algorithm based on additional problem characteristics.

Values

Value	Symbol	Meaning
0	`CPX_ALG_AUTOMATIC`	Automatic: let CPLEX choose; **default**
1	`CPX_ALG_PRIMAL`	Primal simplex
2	`CPX_ALG_DUAL`	Dual simplex
3	`CPX_ALG_NET`	Network simplex
4	`CPX_ALG_BARRIER`	Barrier
5	`CPX_ALG_SIFTING`	Sifting
6	`CPX_ALG_CONCURRENT`	Concurrent (Dual, Barrier, and Primal)

Summary Algorithm for continuous quadratic optimization

C Name `CPX_PARAM_QPMETHOD`

C++ Name `RootAlg`

Java Name `RootAlg`

.NET Name `RootAlg`

InteractiveOptimizer `qpmethod`

Identifier `1063`

Description Determines which algorithm is used when the C routine `CPXqpopt` (or the command `optimize` in the Interactive Optimizer) is invoked.

Currently, the behavior of the Automatic setting is that CPLEX invokes the Barrier Optimizer for continuous QP models. The Automatic setting may be expanded in the future so that CPLEX chooses the algorithm based on additional problem characteristics.

Values

Value	Symbol	Meaning
0	`CPX_ALG_AUTOMATIC`	Automatic: let CPLEX choose; **default**
4	`CPX_ALG_BARRIER`	Barrier

Summary MIP starting algorithm

C Name `CPX_PARAM_STARTALG`

C++ Name `RootAlg`

Java Name `RootAlg`

.NET Name `RootAlg`

InteractiveOptimizer `mip strategy startalgorithm`

Identifier `2025`

Description Determines which continuous optimizer will be used to solve the initial relaxation of a MIP.

The default Automatic setting (0 zero) of this parameter currently selects the dual simplex optimizer for root relaxations for MILP and MIQP. The Automatic setting may be expanded in the future so that CPLEX chooses the algorithm based on additional characteristics of the model.

For MILP (integer constraints and otherwise continuous variables), all settings are permitted.

For MIQP (integer constraints and positive semi-definite quadratic terms in the objective), settings 5 (Sifting) and 6 (Concurrent) are **not** implemented; if you happen to choose them, setting 5 (Sifting) reverts to 0 (ero) and setting 6 (Concurrent) reverts to 4.

For MIQCP (integer constraints and positive semi-definite quadratic terms among the constraints), only the Barrier Optimizer is implemented, and therefore no settings other than 0 (zero) and 4 are permitted.

Values

Value	Symbol	Meaning
0	`CPX_ALG_AUTOMATIC`	Automatic: let CPLEX choose; **default**
1	`CPX_ALG_PRIMAL`	Primal Simplex
2	`CPX_ALG_DUAL`	Dual Simplex
3	`CPX_ALG_NET`	Network Simplex
4	`CPX_ALG_BARRIER`	Barrier
5	`CPX_ALG_SIFTING`	Sifting
6	`CPX_ALG_CONCURRENT`	Concurrent (Dual, Barrier, and Primal)

Summary	Constraint (row) read limit
C Name	`CPX_PARAM_ROWREADLIM`
C++ Name	`RowReadLim`
Java Name	`RowReadLim`
.NET Name	`RowReadLim`
InteractiveOptimizer	`read constraints`
Identifier	`1021`

Description

Specifies a limit for the number of rows (constraints) to read for an allocation of memory.

This parameter does not restrict the size of a problem. Rather, it indirectly specifies the default amount of memory that will be pre-allocated before a problem is read from a file. If the limit is exceeded, more memory is automatically allocated.

Values

Any integer from 0 to 268 435 450; **default**: 30 000.

Summary Scale parameter

C Name `CPX_PARAM_SCAIND`

C++ Name `ScaInd`

Java Name `ScaInd`

.NET Name `ScaInd`

InteractiveOptimizer `read scale`

Identifier `1034`

Description Determines how to scale the problem matrix.

Values

Value	Meaning
-1	No scaling
0	Equilibration scaling; **default**
1	More aggressive scaling

Summary Messages to screen switch

C Name `CPX_PARAM_SCRIND`

C++ Name `screen indicator not available in this API`

Java Name `screen indicator not available in this API`

.NET Name `screen indicator not available in this API`

InteractiveOptimizer `screen indicator not available in this interface`

Identifier `1035`

Description Determines whether or not results are displayed on screen in an application of the C API.

To turn off output to the screen, in a C++ application, , where `cplex` is an instance of the class `IloCplex` and `env` is an instance of the class `IloEnv`, the environment, use `cplex.setOut(env.getNullStream())`.

In a Java application, use `cplex.setOut(null)`.

In a .NET application, use `Cplex.SetOut(Null)` .

Values

Value	Symbol	Meaning
0	`CPX_OFF`	Turn off display of messages to screen; **default**
1	`CPX_ON`	Display messages on screen

Summary Sifting subproblem algorithm

C Name `CPX_PARAM_SIFTALG`

C++ Name `SiftAlg`

Java Name `SiftAlg`

.NET Name `SiftAlg`

InteractiveOptimizer `sifting algorithm`

Identifier `1077`

Description Sets the algorithm to be used for solving sifting subproblems. The default automatic setting will typically use a mix of barrier and primal simplex.

Values

Value	Symbol	Meaning
0	`CPX_ALG_AUTOMATIC`	Automatic: let CPLEX choose; **default**
1	`CPX_ALG_PRIMAL`	Primal Simplex
2	`CPX_ALG_DUAL`	Dual Simplex
3	`CPX_ALG_NET`	Network Simplex
4	`CPX_ALG_BARRIER`	Barrier

Summary Sifting information display

C Name `CPX_PARAM_SIFTDISPLAY`

C++ Name `SiftDisplay`

Java Name `SiftDisplay`

.NET Name `SiftDisplay`

InteractiveOptimizer `sifting display`

Identifier `1076`

Description Determines the amount of information to display about the progress of sifting.

Values

Value	Meaning
0	No display of sifting information
1	Display major iterations; **default**
2	Display LP subproblem information within each sifting iteration

Summary Upper limit on sifting iterations

C Name `CPX_PARAM_SIFTITLIM`

C++ Name `SiftItLim`

Java Name `SiftItLim`

.NET Name `SiftItLim`

InteractiveOptimizer `sifting iterations`

Identifier `1078`

Description Determines the maximum number of sifting iterations that may be performed if convergence to optimality has not been reached.

Values Any nonnegative integer; **default**: 2 100 000 000.

Summary Simplex iteration information display

C Name `CPX_PARAM_SIMDISPLAY`

C++ Name `SimDisplay`

Java Name `SimDisplay`

.NET Name `SimDisplay`

InteractiveOptimizer `simplex display`

Identifier `1019`

Description Determines how often CPLEX reports about iterations during simplex optimization.

Values

Value	Meaning
0	No iteration messages until solution
1	Iteration information after each refactoring; **default**
2	Iteration information for each iteration

Summary	Simplex singularity repair limit
C Name	`CPX_PARAM_SINGLIM`
C++ Name	`SingLim`
Java Name	`SingLim`
.NET Name	`SingLim`
InteractiveOptimizer	`simplex limits singularity`
Identifier	`1037`
Description	Restricts the number of times CPLEX attempts to repair the basis when singularities are encountered during the simplex algorithm. When this limit is exceeded, CPLEX replaces the current basis with the best factorable basis that has been found.
Values	Any nonnegative integer; **default**: 10.

Summary	MIP candidate list limit
C Name	`CPX_PARAM_STRONGCANDLIM`
C++ Name	`StrongCandLim`
Java Name	`StrongCandLim`
.NET Name	`StrongCandLim`
InteractiveOptimizer	`mip limits strongcand`
Identifier	`2045`

Description Controls the length of the candidate list when CPLEX uses variable selection as the setting for strong branching:

- `VarSel` in the C++, Java, or .NET API;
- `CPX_PARAM_VARSEL` in the C API;
- `set mip strategy variableselect 3` in the Interactive Optimizer.

Values Any positive number; **default**: 10.

See Also `CPX_PARAM_VARSEL`, `VarSel`

Summary MIP simplex iterations limit

C Name `CPX_PARAM_STRONGITLIM`

C++ Name `StrongItLim`

Java Name `StrongItLim`

.NET Name `StrongItLim`

InteractiveOptimizer `mip limits strongit`

Identifier `2046`

Description Controls the number of simplex iterations performed on each variable in the candidate list when CPLEX uses variable selection as the setting for strong branching:

- `VarSel` in the C++, Java, or .NET API;
- `CPX_PARAM_VARSEL` in the C API;
- `set mip strategy variableselect 3` in the Interactive Optimizer.

The default setting 0 (zero) chooses the iteration limit automatically.

Values

Value	Meaning
0	Automatic: let CPLEX choose; **default**
Any positive integer	Limit of the simplex iterations performed on each candidate variable

See Also `CPX_PARAM_VARSEL`, `VarSel`

Summary MIP parallel threads limit

C Name `CPX_PARAM_STRONGTHREADLIM`

C++ Name `StrongThreadLim`

Java Name `StrongThreadLim`

.NET Name `StrongThreadLim`

InteractiveOptimizer `mip limits strongthreads`

Identifier `2047`

Description Controls the number of parallel threads used to perform strong branching. This parameter does nothing if the MIP thread limit is greater than 1 (one). The MIP thread limit is controlled by:

- `CPX_PARAM_MIPTHREADS` in the C API;
- `MIPThreads` in the C++, Java, or .NET APIs;
- `set mip limits threads` in the Interactive Optimizer.

The global thread limit (`CPX_PARAM_THREADS`, `Threads`) does **not** affect this parameter.

Values Any positive integer; **default**: 1.

See Also `CPX_PARAM_MIPTHREADS`, `MIPThreads`

Summary Limit on nodes explored when a subMIP is being solved

C Name `CPX_PARAM_SUBMIPNODELIM`

C++ Name `SubMIPNodeLim`

Java Name `SubMIPNodeLim`

.NET Name `SubMIPNodeLim`

InteractiveOptimizer `mip limits submipnodelim`

Identifier `2062`

Description Restricts the number of nodes explored when CPLEX is solving a subMIP. CPLEX solves subMIPs when it builds a solution from a partial MIP start, when repairing an infeasible MIP start, when executing the relaxation induced neighborhood search (RINS) heuristic, when branching locally, or when polishing a solution.

Values Any positive integer; **default**: 500.

Summary Symmetry breaking

C Name `CPX_PARAM_SYMMETRY`

C++ Name `Symmetry`

Java Name `Symmetry`

.NET Name `Symmetry`

InteractiveOptimizer `preprocessing symmetry`

Identifier `2059`

Description Determines whether symmetry breaking reductions will be automatically executed, during the preprocessing phase, in a MIP model.

Values

Value	Meaning
-1	Automatic: let CPLEX choose; **default**
0	Turn off symmetry breaking
1	Exert a moderate level of symmetry breaking
2	Exert an aggressive level of symmetry breaking
3	Exert a very aggressive level of symmetry breaking
4	Exert symmetry breaking with constraints
5	Exert symmetry breaking with bounds

Summary Global default thread count

C Name `CPX_PARAM_THREADS`

C++ Name `Threads`

Java Name `Threads`

.NET Name `Threads`

InteractiveOptimizer `threads`

Identifier `1067`

Description Determines the default number of parallel threads that will be invoked by any CPLEX parallel optimizer. This provides a convenient way to control parallelism with a single parameter setting. The value in place for this parameter can be overridden for any particular CPLEX parallel optimizer by setting the appropriate thread limit (`CPX_PARAM_BARTHREADS`, `BarThreads` or `CPX_PARAM_MIPTHREADS`, `MIPThreads`).

Values

Value	Meaning
1	Minimum; **default**
N	Maximum; determined by license and computer

See Also `CPX_PARAM_BARTHREADS`, `BarThreads`, `CPX_PARAM_MIPTHREADS`, `MIPThreads`

Summary	Global time limit
C Name	`CPX_PARAM_TILIM`
C++ Name	`TiLim`
Java Name	`TiLim`
.NET Name	`TiLim`
InteractiveOptimizer	`timelimit`
Identifier	`1039`

Description

Sets the maximum time, in seconds, for a call to an optimizer. This time limit applies also to the conflict refiner.

The time is measured in terms of either CPU time or elapsed time, according to the setting of the clock type parameter (`CPX_PARAM_CLOCKTYPE`, `ClockType`).

The time limit for an optimizer applies to the sum of all its steps, such as preprocessing,crossover, and internal calls to other optimizers.

In a sequence of calls to optimizers, the limit is not cumulative but applies to each call individually. For example, if you set a time limit of 10 seconds, and you call mipopt twice then there could be a total of (at most) 20 seconds of running time if each call consumes its maximum allotment.

Values

Any nonnegative number; **default**: 1e+75.

See Also

`CPX_PARAM_CLOCKTYPE`, `ClockType`

Summary	Tree memory limit
C Name	`CPX_PARAM_TRELIM`
C++ Name	`TreLim`
Java Name	`TreLim`
.NET Name	`TreLim`
InteractiveOptimizer	`mip limits treememory`
Identifier	`2027`

Description Sets an absolute upper limit on the size (in megabytes) of the branch & cut tree. If this limit is exceeded, CPLEX terminates optimization.

Values Any nonnegative number; **default**: 1e+75.

Summary MIP variable selection strategy

C Name `CPX_PARAM_VARSEL`

C++ Name `VarSel`

Java Name `VarSel`

.NET Name `VarSel`

InteractiveOptimizer `mip strategy variableselect`

Identifier `2028`

Description Sets the rule for selecting the branching variable at the node which has been selected for branching.

The minimum infeasibility rule chooses the variable with the value closest to an integer but still fractional. The minimum infeasibility rule (-1) may lead more quickly to a first integer feasible solution, but is usually slower overall to reach the optimal integer solution.

The maximum infeasibility rule chooses the variable with the value furtherest from an integer. The maximum infeasibility rule (1 one) forces larger changes earlier in the tree.

Pseudo cost (2) variable selection is derived from pseudo-shadow prices.

Strong branching (3) causes variable selection based on partially solving a number of subproblems with tentative branches to see which branch is the most promising. This strategy can be effective on large, difficult MIP problems.

Pseudo reduced costs (4) are a computationally less-intensive form of pseudo costs.

The default value (0 zero) allows CPLEX to select the best rule based on the problem and its progress.

Values

Value	Symbol	Meaning
-1	CPX_VARSEL_MININFEAS	Branch on variable with minimum infeasibility
0	CPX_VARSEL_DEFAULT	Automatic: let CPLEX choose variable to branch on
1	CPX_VARSEL_MAXINFEAS	Branch on variable with maximum infeasibility
2	CPX_VARSEL_PSEUDO	Branch based on pseudo costs
3	CPX_VARSEL_STRONG	Strong branching
4	CPX_VARSEL_PSEUDOREDUCED	Branch based on pseudo reduced costs

Summary Directory for working files

C Name `CPX_PARAM_WORKDIR`

C++ Name `WorkDir`

Java Name `WorkDir`

.NET Name `WorkDir`

InteractiveOptimizer `workdir`

Identifier `1064`

Description Specifies the name of an existing directory into which CPLEX may store temporary working files, such as for MIP node files or for out-of-core barrier files. The default is the current working directory.

Values Any existing directory; **default**: '.'

Summary Memory available for working storage

C Name `CPX_PARAM_WORKMEM`

C++ Name `WorkMem`

Java Name `WorkMem`

.NET Name `WorkMem`

InteractiveOptimizer `workmem`

Identifier `1065`

Description Specifies an upper limit on the amount of central memory, in megabytes, that CPLEX is permitted to use for working memory before swapping to disk files.

Values Any nonnegative number, in megabytes; **default**: 128.0

See Also `CPX_PARAM_WORKDIR`, `WorkMem`

Index