IBM® ILOG® CPLEX®

Callable Library

version 12.1

C API

Reference Manual

2009

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

About This Manual

This reference manual documents the Callable Library, the C application programming interface (API) of IBM(R) ILOG(R) CPLEX(R). There are separate reference manuals for the C++, Java, C#.NET, and Python APIs of CPLEX. Following this table that summarizes the groups in this manual, you will find more information:

- What Are the CPLEX Component Libraries?
- What You Need to Know
- Notation and Naming Conventions
- Related Documentation

Group Summary	
optim.cplex.callable	The API of the CPLEX Callable Library for users of C.
optim.cplex.callable.accessmipresults	The routines in the CPLEX Callable Library to access MIP results.
optim.cplex.callable.accessnetworkresults	The routines in the CPLEX Callable Library to access network results.
optim.cplex.callable.accessqcpresults	The routines in the CPLEX Callable Library to access QCP or SOCP results.
optim.cplex.callable.accessresults	The routines in the CPLEX Callable Library to access results.
optim.cplex.callable.advanced	The API of the advanced C routines of the CPLEX Callable Library.
optim.cplex.callable.advanced.callbacks	The API of the advanced C callback routines of the CPLEX Callable Library.
optim.cplex.callable.analyzesolution	The routines in the CPLEX Callable Library to analyze solutions.
optim.cplex.callable.callbacks	The CPLEX Callable Library routines for managing callbacks.
optim.cplex.callable.createdeletecopy	The routines in the CPLEX Callable Library to create and delete problems and to copy data.
optim.cplex.callable.debug	The CPLEX Callable Library routines for debugging data.
optim.cplex.callable.manageparameters	The routines in the CPLEX Callable Library to manage parameters (that is, set parameters, get current values of parameters, and get information about parameters).
optim.cplex.callable.message	The CPLEX Callable Library routines for managing messages.
optim.cplex.callable.modifynetwork	The routines in the CPLEX Callable Library to modify a network.
optim.cplex.callable.modifyproblem	The routines in the CPLEX Callable Library to modify a problem created by `CPXcreateprob`.
optim.cplex.callable.network	The network routines in the CPLEX Callable Library.
optim.cplex.callable.optimizers	The routines in the CPLEX Callable Library to launch an optimizer.
optim.cplex.callable.portability	The portability routines in the CPLEX Callable Library.
optim.cplex.callable.querygeneralproblem	The routines in the CPLEX Callable Library to query general problem data.
optim.cplex.callable.querymip	The routines in the CPLEX Callable Library to query MIP problem data.
optim.cplex.callable.querynetwork	The routines in the CPLEX Callable Library to query network problem data.
optim.cplex.callable.queryqcp	The routines in the CPLEX Callable Library to query QCP problem data (that is, problems with one or more quadratic constraints), including the special case of second order cone programming (SOCP) problems.
optim.cplex.callable.queryqp	The routines in the CPLEX Callable Library to query QP problem data (that is, problems with a quadratic objective function).

optim.cplex.callable.readfiles	The routines in the CPLEX Callable Library to read files.
optim.cplex.callable.readnetworkfiles	The routines in the CPLEX Callable Library to read network files.
optim.cplex.callable.solutionpool	The routines in the CPLEX Callable Library for the solution pool.
optim.cplex.callable.util	The general utilities in the CPLEX Callable Library.
optim.cplex.callable.writefiles	The routines in the CPLEX Callable Library to write files.
optim.cplex.callable.writenetworkfiles	The routines in the CPLEX Callable Library to write network files.
optim.cplex.errorcodes	The Callable Library macros that define error codes, their symbolic constants, their short message strings, and their explanations. There is a key to the symbols in the short message strings after the table.
optim.cplex.solutionquality	The Callable Library macros that indicate the qualities of a solution, their symbolic constants, and their meaning. Methods for accessing solution quality are mentioned after the table.
optim.cplex.solutionstatus	The Callable Library macros that define solution status, their symbolic constants, their equivalent in Concert Technology enumerations, and their meaning. There is a note about unboundedness after the table.

What Are the CPLEX Component Libraries?

The CPLEX Component Libraries are designed to facilitate the development of applications to solve, modify, and interpret the results of linear, mixed integer, continuous convex quadratic, quadratically constrained, and mixed integer quadratic or quadratically constrained programming.

The CPLEX Component Libraries consist of:

- the CPLEX Callable Library, a C application programming interface (API), and
- Concert Technology, an object-oriented API for C++, Java, and C#.NET users.

Concert Technology is also part of CP Optimizer, enabling cooperative strategies using CPLEX and CP Optimizer together for solving difficult optimization problems.

What You Need to Know

This manual assumes that you are familiar with the operating system on which you are using CPLEX.

The CPLEX Callable Library is written in the C programming language. If you use this product, this manual assumes you can write code in the appropriate language, and that you have a working knowledge of a supported integrated development environment (IDE) for that language.

Notation and Naming Conventions

Throughout this manual:

- The names of routines and parameters defined in the CPLEX Callable Library begin with `CPX`. This convention helps prevent name space conflicts with user-written routines and other code libraries.
- The names of Component Library routines and arguments of routines appear in `this typeface` (examples: `CPXprimopt`, `numcols`)

Related Documentation

In addition to this *Reference Manual* documenting the Callable Library (C API), CPLEX also comes with these resources:

- *Getting Started with CPLEX* introduces you to ways of specifying models and solving problems with CPLEX.
- The *CPLEX User's Manual* explores programming with CPLEX in greater depth. It provides practical ideas about how to use CPLEX in your own applications and shows how and why design and implementation decisions in the examples were made.
- The *CPLEX Release Notes* highlight the new features and important changes in this version.
- The *CPLEX C++ Reference Manual* documents the classes and member functions of the Concert Technology and CPLEX C++ API.
- The *CPLEX Java Reference Manual* supplies detailed definitions of the Concert Technology Java interfaces and CPLEX Java classes.
- The *CPLEX C#.NET Reference Manual* documents the Concert Technology C#.NET interfaces and CPLEX C#.NET classes.
- The *CPLEX Python Reference Manual* supplies detailed definitions of the Python classes, interfaces, modules, and methods.
- Source code for examples is delivered in the standard distribution.
- A file named `readme.html` is delivered in the standard distribution. This file contains the most current information about platform prerequisites for CPLEX.

All of the manuals and Release Notes are available in online versions. The online documentation, in HTML format, can be accessed through standard HTML browsers.

Concepts

Branch and cut

CPLEX uses *branch-and-cut search* when solving mixed integer programming (MIP) models. The branch-and-cut procedure manages a search tree consisting of *nodes*. Every node represents an LP or QP subproblem to be processed; that is, to be solved, to be checked for integrality, and perhaps to be analyzed further. Nodes are called *active* if they have not yet been processed. After a node has been processed, it is no longer active. Cplex processes active nodes in the tree until either no more active nodes are available or some limit has been reached.

A *branch* is the creation of two new nodes from a parent node. Typically, a branch occurs when the bounds on a single variable are modified, with the new bounds remaining in effect for that new node and for any of its descendants. For example, if a branch occurs on a binary variable, that is, one with a lower bound of 0 (zero) and an upper bound of 1 (one), then the result will be two new nodes, one node with a modified upper bound of 0 (the downward branch, in effect requiring this variable to take only the value 0), and the other node with a modified lower bound of 1 (the upward branch, placing the variable at 1). The two new nodes will thus have completely distinct solution domains.

A *cut* is a constraint added to the model. The purpose of adding any cut is to limit the size of the solution domain for the continuous LP or QP problems represented at the nodes, while not eliminating legal integer solutions. The outcome is thus to reduce the number of branches required to solve the MIP.

As an example of a cut, first consider the following constraint involving three binary (0-1) variables:

```
20x + 25y + 30z <= 40
```

That sample constraint can be strengthened by adding the following cut to the model:

```
1x + 1y + 1z <= 1
```

No feasible integer solutions are ruled out by the cut, but some fractional solutions, for example (0.0, 0.4, 1.0), can no longer be obtained in any LP or QP subproblems at the nodes, possibly reducing the amount of searching needed.

The branch-and-cut procedure, then, consists of performing branches and applying cuts at the nodes of the tree. Here is a more detailed outline of the steps involved.

First, the branch-and-cut tree is initialized to contain the root node as the only active node. The root node of the tree represents the entire problem, ignoring all of the explicit integrality requirements. Potential cuts are generated for the root node but, in the interest of keeping the problem size reasonable, not all such cuts are applied to the model immediately. If possible, an incumbent solution (that is, the best known solution that satisfies all the integrality requirements) is established at this point for later use in the algorithm. Such a solution may be established either by CPLEX or by a user who specifies a starting solution by means of the Callable Library routine `CPXcopymipstart` or the Concert Technology method `IloCplex::setVectors`.

When processing a node, CPLEX starts by solving the continuous relaxation of its subproblem, that is, the subproblem without integrality constraints. If the solution violates any cuts, CPLEX may add some or all of them to the node problem and may resolve it, if CPLEX has added cuts. This procedure is iterated until no more violated cuts are detected (or deemed worth adding at this time) by the algorithm. If at any point in the addition of cuts the node becomes infeasible, the node is pruned (that is, it is removed from the tree).

Otherwise, CPLEX checks whether the solution of the node-problem satisfies the integrality constraints. If so, and if its objective value is better than that of the current incumbent, the solution of the node-problem is used as the new incumbent. If not, branching will occur, but first a heuristic method may be tried at this point to see if a new incumbent can be inferred from the LP-QP solution at this node, and other methods of analysis may be performed on this node. The branch, when it occurs, is performed on a variable where the value of the present solution violates its integrality requirement. This practice results in two new nodes being added to the tree for later processing.

Each node, after its relaxation is solved, possesses an optimal objective function value Z. At any given point in the algorithm, there is a node whose Z value is better (less, in the case of a minimization problem, or greater for a maximization problem) than all the others. This Best Node value can be compared to the objective function value of the incumbent solution. The resulting MIP Gap, expressed as a percentage of the incumbent solution, serves as a measure of progress toward finding and proving optimality. When active nodes no longer exist, then these two values will have converged toward each other, and the MIP Gap will thus be zero, signifying that optimality of the incumbent has been proven.

It is possible to tell CPLEX to terminate the branch-and-cut procedure sooner than a completed proof of optimality. For example, a user can set a time limit or a limit on the number of nodes to be processed. Indeed, with default settings, CPLEX will terminate the search when the MIP Gap has been brought lower than 0.0001 (0.01%), because it is often the case that much computation is invested in moving the Best Node value after the eventual optimal incumbent has been located. This termination criterion for the MIP Gap can be changed by the user, of course.

Callbacks in the Callable Library

Callbacks are also known as an interrupt routines. CPLEX supports various types of callbacks.

- **Informational callbacks** allow your application to gather information about the progress of MIP optimization without interfering with performance of the search. In addition, an informational callback also enables your application to terminate optimization. Specifically, informational callbacks check to determine whether your application has invoked the routine `CPXsetterminate` to set a signal to terminate optimization, in which case informational callbacks will terminate optimization for you.
- **Query callbacks**, also known as diagnostic callbacks, make it possible for your application to access information about the progress of optimization, whether continuous or discrete, while optimization is in process. The information available depends on the algorithm (primal simplex, dual simplex, barrier, mixed integer, or network) that you are using. For example, a query callback can return the current objective value, the number of simplex iterations that have been completed, and other details. Query callbacks can also be called from presolve, probing, fractional cuts, and disjunctive cuts. Query callbacks may impede performance because the internal data structures that support query callbacks must be updated frequently. Furthermore, they make assumptions about the path of the search, assumptions that are correct with respect to conventional branch and cut but that may be false with respect to dynamic search. For this reason, query or diagnostic callbacks are **not** compatible with dynamic search. In other words, CPLEX normally turns off dynamic search in the presence of query or diagnostic callbacks in an application.
- **Control callbacks** make it possible for you to define your own user-written routines and for your application to call those routines to interrupt and resume optimization. Control callbacks enable you to direct the search when you are solving a MIP. For example, control callbacks enable you to select the next node to process or to control the creation of subnodes (among other possibilities). Control callbacks are an advanced feature of CPLEX, and as such, they require a greater degree of familiarity with CPLEX algorithms. Because control callbacks can alter the search path in this way, control callbacks are **not** compatible with dynamic search. In other words, CPLEX normally turns off dynamic search in the presence of control callbacks in an application.

If you want to take advantage of dynamic search in your application, you should restrict your use of callbacks to the informational callbacks.

If you see a need for query, diagnostic, or control callbacks in your application, you can override the normal behavior of CPLEX by nondefault settings of the parameters `CPX_PARAM_MIPSEARCH`, `CPX_PARAM_PARALLELMODE`, and `CPX_PARAM_THREADS`. For more details about these parameters and their settings, see the *CPLEX Parameters Reference Manual*.

Callbacks may be called repeatedly at various points during optimization; for each place a callback is called, CPLEX provides a separate callback routine for that particular point.

See also the group `optim.cplex.callable.callbacks` for a list of query and control callbacks.

Infeasibility Tools

When you problem is infeasible, CPLEX offers tools to help you diagnose the cause or causes of infeasibility in your model and possibly repair it: `CPXrefineconflict` and `CPXfeasopt`.

Conflict Refiner

Given an infeasible model, the conflict refiner can identify conflicting constraints and bounds within the model to help you identify the causes of the infeasibility. In this context, a conflict is a subset of the constraints and bounds of the model which are mutually contradictory. The conflict refiner first examines the full infeasible model to identify portions of the conflict that it can remove. By this process of refinement, the conflict refiner arrives at a minimal conflict. A minimal conflict is usually smaller than the full infeasible model and thus makes infeasibility analysis easier. To invoke the conflict refiner, call the routine `CPXrefineconflict`.

If a model happens to include multiple independent causes of infeasibility, then it may be necessary for the user to repair one such cause and then repeat the diagnosis with further conflict analysis.

A conflict does not provide information about the magnitude of change in data values needed to achieve feasibility. The techniques that CPLEX uses to refine a conflict include or remove constraints or bounds in trial conflicts; the techniques do not vary the data in constraints nor in bounds. To gain insight about changes in bounds on variables and constraints, consider the FeasOpt feature.

Also consider FeasOpt for an approach to automatic repair of infeasibility.

Refining a conflict in an infeasible model as defined here is similar to finding an irreducibly inconsistent set (IIS), an established technique in the published literature, long available within CPLEX. Both tools (conflict refiner and IIS finder) attempt to identify an infeasible subproblem in an infeasible model. However, the conflict refiner is more general than the IIS finder. The IIS finder is applicable only in continuous (that is, LP) models, whereas the conflict refiner can work on any type of problem, even mixed integer programs (MIP) and those containing quadratic elements (QP or QCP).

Also the conflict refiner differs from the IIS finder in that a user may organize constraints into one or more groups for a conflict. When a user specifies a group, the conflict refiner will make sure that either the group as a whole will be present in a conflict (that is, all its members will participate in the conflict, and removal of one will result in a feasible subproblem) or that the group will not participate in the conflict at all.

See the Callable Library routine `CPXrefineconflictext` for more about groups.

A user may also assign a numeric preference to constraints or to groups of constraints. In the case of an infeasible model having more than one possible conflict, preferences guide the conflict refiner toward identifying constraints in a conflict as the user prefers.

In these respects, the conflict refiner represents an extension and generalization of the IIS finder.

FeasOpt

Alternatively, after a model have been proven infeasible, `CPXfeasopt` performs an additional optimization that computes a minimal relaxation of the constraints over variables, of the bounds on variables, and of the righthand sides of constraints to make the model feasible. The parameter `CPX_PARAM_FEASOPTMODE` lets you guide `CPXfeasopt` in its computation of this relaxation.

`CPXfeasopt` 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.

Your choice of values for the parameter `CPX_PARAM_FEASOPTMODE` indicates two aspects to CPLEX:

- whether to stop in phase one or continue to phase two:
  - Min means stop in phase one with a minimal relaxation.

    - Opt means continue to phase two for an optimum among those minimal relaxations.
- how to measure the minimality of the relaxation:
    - Sum means CPLEX should minimize the sum of all relaxations
    - Inf means that CPLEX should minimize the number of constraints and bounds relaxed.

The possible values of `CPX_PARAM_FEASOPTMODE` are documented in the routine.

See the group `optim.cplex.solutionstatus` for documentation of the status of a relaxation returned by a call of `CPXfeasopt`.

Unboundedness

The treatment of models that are unbounded involves a few subtleties. Specifically, a declaration of unboundedness means that CPLEX has determined that the model has an unbounded ray. Given any feasible solution x with objective z, a multiple of the unbounded ray can be added to x to give a feasible solution with objective z-1 (or z+1 for maximization models). Thus, if a feasible solution exists, then the optimal objective is unbounded. Note that CPLEX has not necessarily concluded that a feasible solution exists. Users can call the routine `CPXsolninfo` to determine whether CPLEX has also concluded that the model has a feasible solution.

Group optim.cplex.callable

The API of the CPLEX Callable Library for users of C.

Macro Summary
CPX_SOLNPOOL_DIV
CPX_SOLNPOOL_FIFO
CPX_SOLNPOOL_FILTER_DIVERSITY
CPX_SOLNPOOL_FILTER_RANGE
CPX_SOLNPOOL_OBJ

Function Summary
CPXaddchannel
CPXaddcols
CPXaddfpdest
CPXaddfuncdest
CPXaddindconstr
CPXaddmipstarts
CPXaddqconstr
CPXaddrows
CPXaddsolnpooldivfilter
CPXaddsolnpoolrngfilter
CPXaddsos
CPXbaropt
CPXboundsa
CPXcheckaddcols
CPXcheckaddrows
CPXcheckchgcoeflist
CPXcheckcopyctype
CPXcheckcopylp
CPXcheckcopylpwnames
CPXcheckcopyqpsep
CPXcheckcopyquad
CPXcheckcopysos
CPXcheckvals
CPXchgbds
CPXchgcoef
CPXchgcoeflist
CPXchgcolname
CPXchgctype
CPXchgmipstart
CPXchgmipstarts

CPXchgname
CPXchgobj
CPXchgobjsen
CPXchgprobname
CPXchgprobtype
CPXchgprobtypesolnpool
CPXchgqpcoef
CPXchgrhs
CPXchgrngval
CPXchgrowname
CPXchgsense
CPXcleanup
CPXcloneprob
CPXcloseCPLEX
CPXclpwrite
CPXcompletelp
CPXcopybase
CPXcopyctype
CPXcopylp
CPXcopylpwnames
CPXcopymipstart
CPXcopynettolp
CPXcopyobjname
CPXcopyorder
CPXcopypartialbase
CPXcopyqpsep
CPXcopyquad
CPXcopysos
CPXcopystart
CPXcreateprob
CPXdelchannel
CPXdelcols
CPXdelfpdest
CPXdelfuncdest
CPXdelindconstrs
CPXdelmipstarts
CPXdelnames
CPXdelqconstrs
CPXdelrows
CPXdelsetcols
CPXdelsetmipstarts

CPXdelsetrows
CPXdelsetsolnpoolfilters
CPXdelsetsolnpoolsolns
CPXdelsetsos
CPXdelsolnpoolfilters
CPXdelsolnpoolsolns
CPXdisconnectchannel
CPXdperwrite
CPXdualopt
CPXdualwrite
CPXembwrite
CPXfclose
CPXfeasopt
CPXfeasoptext
CPXfltwrite
CPXflushchannel
CPXflushstdchannels
CPXfopen
CPXfputs
CPXfreeprob
CPXgetax
CPXgetbaritcnt
CPXgetbase
CPXgetbestobjval
CPXgetcallbackinfo
CPXgetchannels
CPXgetchgparam
CPXgetcoef
CPXgetcolindex
CPXgetcolinfeas
CPXgetcolname
CPXgetcols
CPXgetconflict
CPXgetconflictext
CPXgetcrossdexchcnt
CPXgetcrossdpushcnt
CPXgetcrosspexchcnt
CPXgetcrossppushcnt
CPXgetctype
CPXgetcutoff
CPXgetdblparam

CPXgetdblquality
CPXgetdj
CPXgetdsbcnt
CPXgeterrorstring
CPXgetgrad
CPXgetindconstr
CPXgetindconstrindex
CPXgetindconstrinfeas
CPXgetindconstrname
CPXgetindconstrslack
CPXgetinfocallbackfunc
CPXgetintparam
CPXgetintquality
CPXgetitcnt
CPXgetlb
CPXgetlogfile
CPXgetlpcallbackfunc
CPXgetmethod
CPXgetmipcallbackfunc
CPXgetmipitcnt
CPXgetmiprelgap
CPXgetmipstart
CPXgetmipstartindex
CPXgetmipstartname
CPXgetmipstarts
CPXgetnetcallbackfunc
CPXgetnodecnt
CPXgetnodeint
CPXgetnodeleftcnt
CPXgetnumbin
CPXgetnumcols
CPXgetnumcuts
CPXgetnumindconstrs
CPXgetnumint
CPXgetnummipstarts
CPXgetnumnz
CPXgetnumqconstrs
CPXgetnumqpnz
CPXgetnumquad
CPXgetnumrows
CPXgetnumsemicont

CPXgetnumsemiint
CPXgetnumsos
CPXgetobj
CPXgetobjname
CPXgetobjsen
CPXgetobjval
CPXgetorder
CPXgetparamname
CPXgetparamnum
CPXgetparamtype
CPXgetphase1cnt
CPXgetpi
CPXgetprobname
CPXgetprobtype
CPXgetpsbcnt
CPXgetqconstr
CPXgetqconstrindex
CPXgetqconstrinfeas
CPXgetqconstrname
CPXgetqconstrslack
CPXgetqpcoef
CPXgetquad
CPXgetrhs
CPXgetrngval
CPXgetrowindex
CPXgetrowinfeas
CPXgetrowname
CPXgetrows
CPXgetsense
CPXgetsiftitcnt
CPXgetsiftphase1cnt
CPXgetslack
CPXgetsolnpooldblquality
CPXgetsolnpooldivfilter
CPXgetsolnpoolfilterindex
CPXgetsolnpoolfiltername
CPXgetsolnpoolfiltertype
CPXgetsolnpoolintquality
CPXgetsolnpoolmeanobjval
CPXgetsolnpoolmipstart
CPXgetsolnpoolnumfilters

CPXgetsolnpoolnummipstarts
CPXgetsolnpoolnumreplaced
CPXgetsolnpoolnumsolns
CPXgetsolnpoolnumsolns
CPXgetsolnpoolobjval
CPXgetsolnpoolqconstrslack
CPXgetsolnpoolrngfilter
CPXgetsolnpoolslack
CPXgetsolnpoolsolnindex
CPXgetsolnpoolsolnname
CPXgetsolnpoolx
CPXgetsos
CPXgetsosindex
CPXgetsosinfeas
CPXgetsosname
CPXgetstat
CPXgetstatstring
CPXgetstrparam
CPXgetsubmethod
CPXgetsubstat
CPXgettime
CPXgettuningcallbackfunc
CPXgetub
CPXgetx
CPXgetxqxax
CPXhybbaropt
CPXhybnetopt
CPXinfodblparam
CPXinfointparam
CPXinfostrparam
CPXlpopt
CPXmbasewrite
CPXmipopt
CPXmsg
CPXmsgstr
CPXmstwrite
CPXmstwritesolnpool
CPXmstwritesolnpoolall
CPXNETaddarcs
CPXNETaddnodes
CPXNETbasewrite

CPXNETcheckcopynet
CPXNETchgarcname
CPXNETchgarcnodes
CPXNETchgbds
CPXNETchgname
CPXNETchgnodename
CPXNETchgobj
CPXNETchgobjsen
CPXNETchgsupply
CPXNETcopybase
CPXNETcopynet
CPXNETcreateprob
CPXNETdelarcs
CPXNETdelnodes
CPXNETdelset
CPXNETextract
CPXNETfreeprob
CPXNETgetarcindex
CPXNETgetarcname
CPXNETgetarcnodes
CPXNETgetbase
CPXNETgetdj
CPXNETgetitcnt
CPXNETgetlb
CPXNETgetnodearcs
CPXNETgetnodeindex
CPXNETgetnodename
CPXNETgetnumarcs
CPXNETgetnumnodes
CPXNETgetobj
CPXNETgetobjsen
CPXNETgetobjval
CPXNETgetphase1cnt
CPXNETgetpi
CPXNETgetprobname
CPXNETgetslack
CPXNETgetstat
CPXNETgetsupply
CPXNETgetub
CPXNETgetx
CPXNETprimopt

CPXNETreadcopybase
CPXNETreadcopyprob
CPXNETsolninfo
CPXNETsolution
CPXNETwriteprob
CPXnewcols
CPXnewrows
CPXobjsa
CPXopenCPLEX
CPXordwrite
CPXpopulate
CPXpperwrite
CPXpreslvwrite
CPXprimopt
CPXputenv
CPXqpindefcertificate
CPXqpopt
CPXreadcopybase
CPXreadcopymipstart
CPXreadcopymipstarts
CPXreadcopyorder
CPXreadcopyparam
CPXreadcopyprob
CPXreadcopysol
CPXreadcopysolnpoolfilters
CPXrefineconflict
CPXrefineconflictext
CPXrefinemipstartconflict
CPXrefinemipstartconflictext
CPXrhssa
CPXsetdblparam
CPXsetdefaults
CPXsetinfocallbackfunc
CPXsetintparam
CPXsetlogfile
CPXsetlpcallbackfunc
CPXsetmipcallbackfunc
CPXsetnetcallbackfunc
CPXsetstrparam
CPXsetterminate
CPXsettuningcallbackfunc

CPXsiftopt
CPXsolninfo
CPXsolution
CPXsolwrite
CPXsolwritesolnpool
CPXsolwritesolnpoolall
CPXstrcpy
CPXstrlen
CPXtuneparam
CPXtuneparamprobset
CPXversion
CPXwritemipstarts
CPXwriteparam
CPXwriteprob

For access to the routines of the Callable Library organized by their purpose, see the Overview of the API or see the groups of `optim.cplex.callable`.

Group optim.cplex.callable.accessmipresults

The routines in the CPLEX Callable Library to access MIP results.

Function Summary
CPXgetbestobjval
CPXgetcutoff
CPXgetindconstrinfeas
CPXgetindconstrslack
CPXgetmipitcnt
CPXgetmiprelgap
CPXgetmipstart
CPXgetmipstartindex
CPXgetmipstartname
CPXgetmipstarts
CPXgetnodecnt
CPXgetnodeint
CPXgetnodeleftcnt
CPXgetnumcuts
CPXgetnummipstarts
CPXgetsolnpoolmipstart
CPXgetsolnpoolnummipstarts
CPXgetsosinfeas
CPXgetsubmethod
CPXgetsubstat

Solution query routines are used to access information about the results of applying an optimization method to a problem object. For MIP problem objects, you can access the values of variables and constraint slacks. Methods and routines are also available to retrieve other information about the optimization process (such as the number of nodes used).

Group optim.cplex.callable.accessnetworkresults

The routines in the CPLEX Callable Library to access network results.

Function Summary
CPXNETgetbase
CPXNETgetdj
CPXNETgetitcnt
CPXNETgetobjval
CPXNETgetphase1cnt
CPXNETgetpi
CPXNETgetslack
CPXNETgetstat
CPXNETgetx
CPXNETsolninfo
CPXNETsolution

Use these routines to access results after you have used the network optimizer on a problem object created as a network flow structure.

Group optim.cplex.callable.accessqcpresults

The routines in the CPLEX Callable Library to access QCP or SOCP results.

Function Summary
CPXgetqconstrinfeas
CPXgetqconstrslack
CPXgetxqxax

Solution query routines are used to access information about the results of applying an optimization method to a quadratically constrained (QCP) problem object or to the special case of second order cone programming (SOCP) problems. For QCP problem objects, you can access the constraint slacks and constraint activity levels, in addition to the information that you can access through routines in the groups `optim.cplex.callable.accessmipresults` and `optim.cplex.callable.accessresults`.

Group optim.cplex.callable.accessresults

The routines in the CPLEX Callable Library to access results.

Function Summary
CPXgetax
CPXgetbaritcnt
CPXgetbase
CPXgetcolinfeas
CPXgetcrossdexchcnt
CPXgetcrossdpushcnt
CPXgetcrosspexchcnt
CPXgetcrossppushcnt
CPXgetdj
CPXgetdsbcnt
CPXgetitcnt
CPXgetmethod
CPXgetobjval
CPXgetphase1cnt
CPXgetpi
CPXgetpsbcnt
CPXgetrowinfeas
CPXgetsiftitcnt
CPXgetsiftphase1cnt
CPXgetslack
CPXgetstat
CPXgetstatstring
CPXgetx
CPXsolninfo
CPXsolution

Solution query routines are used to access information about the results of applying an optimization method to a problem object.

For MIP problem objects, you can access variable and slack values. (Other values specific to MIPs are accessible through routines in the group `optim.cplex.callable.accessmipresults`.)

For LP and QP problem objects, you can access the values of variables, constraint slacks, reduced costs, and dual variables. Additionally, for an LP or QP problem object solved with a simplex method, you can query the simplex basis. Methods and routines are also available to retrieve other information about the optimization process (such as the iteration count).

Group optim.cplex.callable.advanced

The API of the advanced C routines of the CPLEX Callable Library.

Function Summary
CPXaddlazyconstraints
CPXaddusercuts
CPXbasicpresolve
CPXbinvacol
CPXbinvarow
CPXbinvcol
CPXbinvrow
CPXbranchcallbackbranchbds
CPXbranchcallbackbranchconstraints
CPXbranchcallbackbranchgeneral
CPXbtran
CPXcopybasednorms
CPXcopydnorms
CPXcopypnorms
CPXcopyprotected
CPXcrushform
CPXcrushpi
CPXcrushx
CPXcutcallbackadd
CPXcutcallbackaddlocal
CPXdjfrompi
CPXdualfarkas
CPXfreelazyconstraints
CPXfreepresolve
CPXfreeusercuts
CPXftran
CPXgetbasednorms
CPXgetbhead
CPXgetbranchcallbackfunc
CPXgetcallbackctype
CPXgetcallbackgloballb
CPXgetcallbackglobalub
CPXgetcallbackincumbent
CPXgetcallbackindicatorinfo
CPXgetcallbacklp
CPXgetcallbacknodeinfo
CPXgetcallbacknodeintfeas

CPXgetcallbacknodelb
CPXgetcallbacknodelp
CPXgetcallbacknodeobjval
CPXgetcallbacknodestat
CPXgetcallbacknodeub
CPXgetcallbacknodex
CPXgetcallbackorder
CPXgetcallbackpseudocosts
CPXgetcallbackseqinfo
CPXgetcallbacksosinfo
CPXgetcutcallbackfunc
CPXgetdeletenodecallbackfunc
CPXgetdnorms
CPXgetheuristiccallbackfunc
CPXgetijdiv
CPXgetijrow
CPXgetincumbentcallbackfunc
CPXgetnodecallbackfunc
CPXgetobjoffset
CPXgetpnorms
CPXgetprestat
CPXgetprotected
CPXgetray
CPXgetredlp
CPXgetsolvecallbackfunc
CPXkilldnorms
CPXkillpnorms
CPXmdleave
CPXpivot
CPXpivotin
CPXpivotout
CPXpreaddrows
CPXprechgobj
CPXpresolve
CPXqconstrslackfromx
CPXqpdjfrompi
CPXqpuncrushpi
CPXsetbranchcallbackfunc
CPXsetbranchnosolncallbackfunc
CPXsetcutcallbackfunc
CPXsetdeletenodecallbackfunc

CPXsetheuristiccallbackfunc
CPXsetincumbentcallbackfunc
CPXsetnodecallbackfunc
CPXsetsolvecallbackfunc
CPXslackfromx
CPXstrongbranch
CPXtightenbds
CPXuncrushform
CPXuncrushpi
CPXuncrushx
CPXunscaleprob

Warning

These advanced routines typically demand a profound understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

Group optim.cplex.callable.advanced.callbacks

The API of the advanced C callback routines of the CPLEX Callable Library.

Function Summary
CPXbranchcallbackbranchbds
CPXbranchcallbackbranchconstraints
CPXbranchcallbackbranchgeneral
CPXcutcallbackadd
CPXcutcallbackaddlocal
CPXgetbranchcallbackfunc
CPXgetcallbackctype
CPXgetcallbackgloballb
CPXgetcallbackglobalub
CPXgetcallbackincumbent
CPXgetcallbackindicatorinfo
CPXgetcallbacklp
CPXgetcallbacknodeinfo
CPXgetcallbacknodeintfeas
CPXgetcallbacknodelb
CPXgetcallbacknodelp
CPXgetcallbacknodeobjval
CPXgetcallbacknodestat
CPXgetcallbacknodeub
CPXgetcallbacknodex
CPXgetcallbackorder
CPXgetcallbackpseudocosts
CPXgetcallbackseqinfo
CPXgetcallbacksosinfo
CPXgetcutcallbackfunc
CPXgetdeletenodecallbackfunc
CPXgetheuristiccallbackfunc
CPXgetincumbentcallbackfunc
CPXgetnodecallbackfunc
CPXgetsolvecallbackfunc
CPXsetbranchcallbackfunc
CPXsetbranchnosolncallbackfunc
CPXsetcutcallbackfunc
CPXsetdeletenodecallbackfunc
CPXsetheuristiccallbackfunc
CPXsetincumbentcallbackfunc
CPXsetnodecallbackfunc

CPXsetsolvecallbackfunc

Warning

These advanced callback routines typically demand a profound understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

Advanced callback routines are **not** compatible with dynamic search.

Group optim.cplex.callable.analyzesolution

The routines in the CPLEX Callable Library to analyze solutions.

Function Summary
CPXboundsa
CPXclpwrite
CPXfeasopt
CPXfeasoptext
CPXgetconflict
CPXgetconflictext
CPXgetdblquality
CPXgetgrad
CPXgetintquality
CPXgetsolnpooldblquality
CPXgetsolnpoolintquality
CPXobjsa
CPXqpindefcertificate
CPXrefineconflict
CPXrefineconflictext
CPXrefinemipstartconflict
CPXrefinemipstartconflictext
CPXrhssa

Solution analysis routines give further information about a solution. As the solutions are computed with finite-precision arithmetic, there may be some numeric residuals; the quality routines give information about what these numeric residuals are. The sensitivity analysis routines give information about how the solution would change if some aspect of the problem is changed; these routines require a simplex basis, so they may be used only after a simplex optimization of an LP.

Group optim.cplex.callable.callbacks

The CPLEX Callable Library routines for managing callbacks.

Function Summary
CPXgetcallbackinfo
CPXgetinfocallbackfunc
CPXgetlpcallbackfunc
CPXgetmipcallbackfunc
CPXgetnetcallbackfunc
CPXgettuningcallbackfunc
CPXsetinfocallbackfunc
CPXsetlpcallbackfunc
CPXsetmipcallbackfunc
CPXsetnetcallbackfunc
CPXsettuningcallbackfunc

These callback routines, also known as interrupt routines, make it possible for you to define your own functions and for your application to call those functions to interrupt and resume optimization. You can also use callbacks to access progress information while the optimization is in process.

Group optim.cplex.callable.createdeletecopy

The routines in the CPLEX Callable Library to create and delete problems and to copy data.

Function Summary
CPXcloneprob
CPXcompletelp
CPXcopybase
CPXcopyctype
CPXcopylp
CPXcopylpwnames
CPXcopymipstart
CPXcopynettolp
CPXcopyobjname
CPXcopyorder
CPXcopyqpsep
CPXcopyquad
CPXcopysos
CPXcopystart
CPXcreateprob
CPXdelmipstarts
CPXdelsetmipstarts
CPXfreeprob
CPXNETaddarcs
CPXNETaddnodes
CPXNETcheckcopynet
CPXNETcopybase
CPXNETcopynet
CPXNETcreateprob
CPXNETextract
CPXNETfreeprob

These routines create, populate, or delete a problem object. You can also populate a problem object by means of the routines for modifying a problem or for reading data from a file.

Group optim.cplex.callable.debug

The CPLEX Callable Library routines for debugging data.

Function Summary
CPXcheckaddcols
CPXcheckaddrows
CPXcheckchgcoeflist
CPXcheckcopyctype
CPXcheckcopylp
CPXcheckcopylpwnames
CPXcheckcopyqpsep
CPXcheckcopyquad
CPXcheckcopysos
CPXcheckvals

These routines help you validate data in your problem to verify that you are solving the problem you intend.

Group optim.cplex.callable.manageparameters

The routines in the CPLEX Callable Library to manage parameters (that is, set parameters, get current values of parameters, and get information about parameters).

Function Summary
CPXgetchgparam
CPXgetdblparam
CPXgetintparam
CPXgetparamname
CPXgetparamnum
CPXgetparamtype
CPXgetstrparam
CPXinfodblparam
CPXinfointparam
CPXinfostrparam
CPXreadcopyparam
CPXsetdblparam
CPXsetdefaults
CPXsetintparam
CPXsetstrparam
CPXwriteparam

These routines are used to set parameters that control various aspects of CPLEX behavior and to find out current, default, and allowed values for parameters. For more information about parameters, see the *CPLEX Parameters Reference Manual*.

Group optim.cplex.callable.message

The CPLEX Callable Library routines for managing messages.

Function Summary
CPXaddchannel
CPXaddfpdest
CPXaddfuncdest
CPXdelchannel
CPXdelfpdest
CPXdelfuncdest
CPXdisconnectchannel
CPXflushchannel
CPXflushstdchannels
CPXgetchannels
CPXgeterrorstring
CPXgetlogfile
CPXmsg
CPXsetlogfile

These routines make it possible for your application to control which messages from CPLEX appear on screen, which are sent to files. They also provide support for you to create your own messages.

Group optim.cplex.callable.modifynetwork

The routines in the CPLEX Callable Library to modify a network.

Function Summary
CPXNETchgarcname
CPXNETchgarcnodes
CPXNETchgbds
CPXNETchgname
CPXNETchgnodename
CPXNETchgobj
CPXNETchgobjsen
CPXNETchgsupply
CPXNETdelarcs
CPXNETdelnodes
CPXNETdelset

After you have created a network problem object, use these routines to modify it.

Group optim.cplex.callable.modifyproblem

The routines in the CPLEX Callable Library to modify a problem created by `CPXcreateprob`.

Function Summary
CPXaddcols
CPXaddindconstr
CPXaddmipstarts
CPXaddqconstr
CPXaddrows
CPXaddsos
CPXchgbds
CPXchgcoef
CPXchgcoeflist
CPXchgcolname
CPXchgctype
CPXchgmipstart
CPXchgmipstarts
CPXchgname
CPXchgobj
CPXchgobjsen
CPXchgprobname
CPXchgprobtype
CPXchgprobtypesolnpool
CPXchgqpcoef
CPXchgrhs
CPXchgrngval
CPXchgrowname
CPXchgsense
CPXdelcols
CPXdelindconstrs
CPXdelmipstarts
CPXdelnames
CPXdelqconstrs
CPXdelrows
CPXdelsetcols
CPXdelsetmipstarts
CPXdelsetrows
CPXdelsetsos
CPXnewcols
CPXnewrows

Problem modification routines change a problem object after it has been created. The modifications that you can make are these:

- adding rows and columns to the constraint matrix,
- deleting rows and columns from the constraint matrix,
- changing the sense of the objective function,
- changing the value of coefficients in the constraint matrix,
- changing an objective or righthand side coefficient,
- changing the bounds on a variable,
- changing the sense of a constraint,
- changing names of rows or columns.

Group optim.cplex.callable.network

The network routines in the CPLEX Callable Library.

Function Summary
CPXNETaddarcs
CPXNETaddnodes
CPXNETbasewrite
CPXNETcheckcopynet
CPXNETchgarcname
CPXNETchgarcnodes
CPXNETchgbds
CPXNETchgname
CPXNETchgnodename
CPXNETchgobj
CPXNETchgobjsen
CPXNETchgsupply
CPXNETcopybase
CPXNETcopynet
CPXNETcreateprob
CPXNETdelarcs
CPXNETdelnodes
CPXNETdelset
CPXNETextract
CPXNETfreeprob
CPXNETgetarcindex
CPXNETgetarcname
CPXNETgetarcnodes
CPXNETgetbase
CPXNETgetdj
CPXNETgetitcnt
CPXNETgetlb
CPXNETgetnodearcs
CPXNETgetnodeindex
CPXNETgetnodename
CPXNETgetnumarcs
CPXNETgetnumnodes
CPXNETgetobj
CPXNETgetobjsen
CPXNETgetobjval
CPXNETgetphase1cnt
CPXNETgetpi

CPXNETgetprobname
CPXNETgetslack
CPXNETgetstat
CPXNETgetsupply
CPXNETgetub
CPXNETgetx
CPXNETprimopt
CPXNETreadcopybase
CPXNETreadcopyprob
CPXNETsolninfo
CPXNETsolution
CPXNETwriteprob

If part of your problem is structured as a network, then you may want to consider calling the CPLEX Network Optimizer. This optimizer may have a positive impact on performance. There are two alternative ways of calling the network optimizer:

- If your problem is an LP where a large part is a network structure, you may call the network optimizer for the populated LP object.
- If your entire problem consists of a network flow, you should consider creating a network object instead of an LP object. Then populate it, and solve it with the network optimizer. This alternative generally yields the best performance because it does not incur the overhead of LP data structures. This option is only available for the CPLEX Callable Library.

For more about formulating a problem in this way and applying the network optimizer in your application, see this topic in the *CPLEX User's Manual*.

Group optim.cplex.callable.optimizers

The routines in the CPLEX Callable Library to launch an optimizer.

Function Summary
CPXbaropt
CPXcleanup
CPXdualopt
CPXhybbaropt
CPXhybnetopt
CPXlpopt
CPXmipopt
CPXNETprimopt
CPXprimopt
CPXqpopt
CPXsiftopt

After you have specified the problem by populating a problem object, the problem can be optimized. A default optimizer is provided for each problem type. In most cases, the default optimizer will solve your problem well, but you may select a different optimizer that may suit your needs better for a particular model formulation.

Continuous LP and QP problem objects can be optimized with simplex or barrier optimizers. Continuous QCP problem objects and the special case of second order cone programming (SOCP) problems can be optimized with the barrier optimizer only.

For MIP problem objects, any appropriate continuous optimizer may be specified to solve the subproblems. You can also specify a different optimizer for solving the root LP subproblem and for the LP subproblems that occur at the nodes of the branch and cut tree.

For more information about parameters and their settings, see the *CPLEX Parameters Reference Manual*.

Group optim.cplex.callable.portability

The portability routines in the CPLEX Callable Library.

Function Summary
CPXfclose
CPXfopen
CPXfputs
CPXgettime
CPXmsgstr
CPXputenv
CPXstrcpy
CPXstrlen
CPXversion

Portability routines are needed for Windows platforms. They may also be used on UNIX platforms.

Group optim.cplex.callable.querygeneralproblem

The routines in the CPLEX Callable Library to query general problem data.

Function Summary
CPXgetcoef
CPXgetcolindex
CPXgetcolname
CPXgetcols
CPXgetlb
CPXgetnumcols
CPXgetnumnz
CPXgetnumrows
CPXgetobj
CPXgetobjname
CPXgetobjsen
CPXgetprobname
CPXgetprobtype
CPXgetrhs
CPXgetrngval
CPXgetrowindex
CPXgetrowname
CPXgetrows
CPXgetsense
CPXgetub

These routines to access information about a problem object after it has been created can be used at any time, even after problem modifications.

Group optim.cplex.callable.querymip

The routines in the CPLEX Callable Library to query MIP problem data.

Function Summary
CPXgetctype
CPXgetindconstr
CPXgetindconstrindex
CPXgetindconstrname
CPXgetnumbin
CPXgetnumindconstrs
CPXgetnumint
CPXgetnumsemicont
CPXgetnumsemiint
CPXgetnumsos
CPXgetorder
CPXgetsos
CPXgetsosindex
CPXgetsosname

These routines to access information about a MIP problem object after it has been created can be used at any time, even after problem modifications.

Group optim.cplex.callable.querynetwork

The routines in the CPLEX Callable Library to query network problem data.

Function Summary
CPXNETgetarcindex
CPXNETgetarcname
CPXNETgetarcnodes
CPXNETgetlb
CPXNETgetnodearcs
CPXNETgetnodeindex
CPXNETgetnodename
CPXNETgetnumarcs
CPXNETgetnumnodes
CPXNETgetobj
CPXNETgetobjsen
CPXNETgetprobname
CPXNETgetsupply
CPXNETgetub

These routines to access information about a network problem object after it has been created can be used at any time, even after problem modifications.

Group optim.cplex.callable.queryqcp

The routines in the CPLEX Callable Library to query QCP problem data (that is, problems with one or more quadratic constraints), including the special case of second order cone programming (SOCP) problems.

Function Summary
CPXgetnumqconstrs
CPXgetqconstr
CPXgetqconstrindex
CPXgetqconstrname

These routines to access information about a QCP problem object after it has been created can be used at any time, even after problem modifications. That is, these routines access information about a problem object with one or more quadratic constraints.

Group optim.cplex.callable.queryqp

The routines in the CPLEX Callable Library to query QP problem data (that is, problems with a quadratic objective function).

Function Summary
CPXgetnumqpnz
CPXgetnumquad
CPXgetqpcoef
CPXgetquad

These routines to access information about a QP problem object after it has been created can be used at any time, even after problem modifications. That is, these routines access information about a problem object with a quadratic objective function.

Group optim.cplex.callable.readfiles

The routines in the CPLEX Callable Library to read files.

Function Summary
CPXNETreadcopybase
CPXNETreadcopyprob
CPXreadcopybase
CPXreadcopymipstart
CPXreadcopymipstarts
CPXreadcopyorder
CPXreadcopyprob
CPXreadcopysol
CPXreadcopysolnpoolfilters

Use these routines to read data from system files. CPLEX can read problem files stored in a variety of formats. For more information about the file formats, see the *CPLEX File Formats Reference Manual*.

Group optim.cplex.callable.readnetworkfiles

The routines in the CPLEX Callable Library to read network files.

Function Summary
CPXNETreadcopybase
CPXNETreadcopyprob

Use these routines to read data from system files into a network problem object.

Group optim.cplex.callable.solutionpool

The routines in the CPLEX Callable Library for the solution pool.

Macro Summary
CPX_SOLNPOOL_DIV
CPX_SOLNPOOL_FIFO
CPX_SOLNPOOL_FILTER_DIVERSITY
CPX_SOLNPOOL_FILTER_RANGE
CPX_SOLNPOOL_OBJ

Function Summary
CPXaddsolnpooldivfilter
CPXaddsolnpoolrngfilter
CPXchgprobtypesolnpool
CPXdelsetsolnpoolfilters
CPXdelsetsolnpoolsolns
CPXdelsolnpoolfilters
CPXdelsolnpoolsolns
CPXfltwrite
CPXgetsolnpooldblquality
CPXgetsolnpooldivfilter
CPXgetsolnpoolfilterindex
CPXgetsolnpoolfiltername
CPXgetsolnpoolfiltertype
CPXgetsolnpoolintquality
CPXgetsolnpoolmeanobjval
CPXgetsolnpoolmipstart
CPXgetsolnpoolnumfilters
CPXgetsolnpoolnummipstarts
CPXgetsolnpoolnumreplaced
CPXgetsolnpoolnumsolns
CPXgetsolnpoolnumsolns
CPXgetsolnpoolobjval
CPXgetsolnpoolqconstrslack
CPXgetsolnpoolrngfilter
CPXgetsolnpoolslack
CPXgetsolnpoolsolnindex
CPXgetsolnpoolsolnname
CPXgetsolnpoolx
CPXmstwritesolnpool
CPXmstwritesolnpoolall

CPXpopulate
CPXreadcopysolnpoolfilters
CPXsolwritesolnpool
CPXsolwritesolnpoolall

These routines populate, filter, and manage the solution pool to accumulate or generate multiple solutions.

Group optim.cplex.callable.util

The general utilities in the CPLEX Callable Library.

Function Summary
CPXcloseCPLEX
CPXopenCPLEX
CPXsetterminate

These utilities initialize and close the CPLEX environment.

Group optim.cplex.callable.writefiles

The routines in the CPLEX Callable Library to write files.

Function Summary
CPXdperwrite
CPXdualwrite
CPXembwrite
CPXfltwrite
CPXmbasewrite
CPXmstwrite
CPXmstwritesolnpool
CPXmstwritesolnpoolall
CPXNETbasewrite
CPXNETwriteprob
CPXordwrite
CPXpperwrite
CPXpreslvwrite
CPXsolwrite
CPXsolwritesolnpool
CPXsolwritesolnpoolall
CPXwritemipstarts
CPXwriteprob

These routines write a problem object or, after the problem has been optimized, they write the optimal basis or solution report to a file.

Group optim.cplex.callable.writenetworkfiles

The routines in the CPLEX Callable Library to write network files.

Function Summary
CPXNETbasewrite
CPXNETwriteprob

These routines write a network problem object or, after the network problem has been optimized, they write the optimal basis or solution report to a file.

Group optim.cplex.errorcodes

The Callable Library macros that define error codes, their symbolic constants, their short message strings, and their explanations. There is a key to the symbols in the short message strings after the table.

Macro Summary	
CPXERR_ABORT_STRONGBRANCH	1263 Strong branching aborted
CPXERR_ADJ_SIGN_QUAD	1606 Lines %d,%d: Adjacent sign and quadratic character
CPXERR_ADJ_SIGN_SENSE	1604 Lines %d,%d: Adjacent sign and sense
CPXERR_ADJ_SIGNS	1602 Lines %d,%d: Adjacent signs
CPXERR_ALGNOTLICENSED	32024 Licensing problem: Optimization algorithm not licensed
CPXERR_ARC_INDEX_RANGE	1231 Arc index %d out of range
CPXERR_ARRAY_BAD_SOS_TYPE	3009 Illegal sostype entry %d
CPXERR_ARRAY_NOT_ASCENDING	1226 Array entry %d not ascending
CPXERR_ARRAY_TOO_LONG	1208 Array length too long
CPXERR_BAD_ARGUMENT	1003 Bad argument to Callable Library routine.
CPXERR_BAD_BOUND_SENSE	1622 Line %d: Invalid bound sense
CPXERR_BAD_BOUND_TYPE	1457 Line %d: Unrecognized bound type '%s'
CPXERR_BAD_CHAR	1537 Illegal character
CPXERR_BAD_CTYPE	3021 Illegal ctype entry %d
CPXERR_BAD_DIRECTION	3012 Line %d: Unrecognized direction '%c%c'
CPXERR_BAD_EXPO_RANGE	1435 Line %d: Exponent '%s' out of range
CPXERR_BAD_EXPONENT	1618 Line %d: Exponent '%s' not %s with number
CPXERR_BAD_FILETYPE	1424 Invalid filetype
CPXERR_BAD_ID	1617 Line %d: '%s' not valid identifier
CPXERR_BAD_INDCONSTR	1439 Line %d: Illegal indicator constraint
CPXERR_BAD_INDICATOR	1551 Line %d: Unrecognized basis marker '%s'
CPXERR_BAD_LAZY_UCUT	1438 Line %d: Illegal lazy constraint or user cut
CPXERR_BAD_LUB	1229 Illegal bound change specified by entry %d
CPXERR_BAD_METHOD	1292 Invalid choice of optimization method
CPXERR_BAD_NUMBER	1434 Line %d: Couldn't convert '%s' to a number
CPXERR_BAD_OBJ_SENSE	1487 Line %d: Unrecognized objective sense '%s'
CPXERR_BAD_PARAM_NAME	1028 Bad parameter name to CPLEX parameter routine
CPXERR_BAD_PARAM_NUM	1013 Bad parameter number to CPLEX parameter routine
CPXERR_BAD_PIVOT	1267 Illegal pivot
CPXERR_BAD_PRIORITY	3006 Negative priority entry %d
CPXERR_BAD_PROB_TYPE	1022 Unknown problem type. Problem not changed
CPXERR_BAD_ROW_ID	1532 Incorrect row identifier
CPXERR_BAD_SECTION_BOUNDS	1473 Line %d: Unrecognized section marker. Expecting RANGES, BOUNDS, QMATRIX, or ENDATA
CPXERR_BAD_SECTION_ENDATA	1462 Line %d: Unrecognized section marker. Expecting ENDATA

CPXERR_BAD_SECTION_QMATRIX	1475 Line %d: Unrecognized section marker. Expecting QMATRIX or ENDATA
CPXERR_BAD_SENSE	1215 Illegal sense entry %d
CPXERR_BAD_SOS_TYPE	1442 Line %d: Unrecognized SOS type: %c%c
CPXERR_BAD_STATUS	1253 Invalid status entry %d for basis specification
CPXERR_BADPRODUCT	32023 Licensing problem: License not valid for this product
CPXERR_BAS_FILE_SHORT	1550 Basis missing some basic variables
CPXERR_BAS_FILE_SIZE	1555 %d %s basic variable(s)
CPXERR_CALLBACK	1006 Error during callback
CPXERR_CANT_CLOSE_CHILD	1021 Cannot close a child environment
CPXERR_CHILD_OF_CHILD	1019 Cannot clone a cloned environment
CPXERR_COL_INDEX_RANGE	1201 Column index %d out of range
CPXERR_COL_REPEAT_PRINT	1478 %d Column repeats messages not printed
CPXERR_COL_REPEATS	1446 Column '%s' repeats
CPXERR_COL_ROW_REPEATS	1443 Column '%s' has repeated row '%s'
CPXERR_COL_UNKNOWN	1449 Line %d: '%s' is not a column name
CPXERR_CONFLICT_UNSTABLE	1720 Infeasibility not reproduced.
CPXERR_COUNT_OVERLAP	1228 Count entry %d specifies overlapping entries
CPXERR_COUNT_RANGE	1227 Count entry %d negative or larger than allowed
CPXERR_DBL_MAX	1233 Numeric entry %d is larger than allowed maximum of %g
CPXERR_DECOMPRESSION	1027 Decompression of unpresolved problem failed
CPXERR_DUP_ENTRY	1222 Duplicate entry or entries
CPXERR_EXTRA_BV_BOUND	1456 Line %d: 'BV' bound type illegal when prior bound given
CPXERR_EXTRA_FR_BOUND	1455 Line %d: 'FR' bound type illegal when prior bound given
CPXERR_EXTRA_FX_BOUND	1454 Line %d: 'FX' bound type illegal when prior bound given
CPXERR_EXTRA_INTEND	1481 Line %d: 'INTEND' found while not reading integers
CPXERR_EXTRA_INTORG	1480 Line %d: 'INTORG' found while reading integers
CPXERR_EXTRA_SOSEND	1483 Line %d: 'SOSEND' found while not reading a SOS
CPXERR_EXTRA_SOSORG	1482 Line %d: 'SOSORG' found while reading a SOS
CPXERR_FAIL_OPEN_READ	1423 Could not open file '%s' for reading
CPXERR_FAIL_OPEN_WRITE	1422 Could not open file '%s' for writing
CPXERR_FILE_ENTRIES	1553 Line %d: Wrong number of entries
CPXERR_FILE_FORMAT	1563 File '%s' has an incompatible format. Try setting reverse flag
CPXERR_FILTER_VARIABLE_TYPE	3414 Diversity filter has non-binary variable(s)
CPXERR_ILL_DEFINED_PWL	1213
CPXERR_ILOG_LICENSE	32201 ILM Error %d
CPXERR_IN_INFOCALLBACK	1804 Calling routines not allowed in informational callback
CPXERR_INDEX_NOT_BASIC	1251 Index must correspond to a basic variable

CPXERR_INDEX_RANGE	1200 Index is outside range of valid values
CPXERR_INDEX_RANGE_HIGH	1206 %s: 'end' value %d is greater than %d
CPXERR_INDEX_RANGE_LOW	1205 %s: 'begin' value %d is less than %d
CPXERR_INT_TOO_BIG	3018 Magnitude of variable %s: %g exceeds integer limit %d
CPXERR_INT_TOO_BIG_INPUT	1463 Line %d: Magnitude exceeds integer limit %d
CPXERR_INVALID_NUMBER	1650 Number not representable in exponential notation
CPXERR_LIMITS_TOO_BIG	1012 Problem size limits too large
CPXERR_LINE_TOO_LONG	1465 Line %d: Line longer than limit of %d characters
CPXERR_LO_BOUND_REPEATS	1459 Line %d: Repeated lower bound
CPXERR_LP_NOT_IN_ENVIRONMENT	1806 Problem is not member of this environment
CPXERR_MIPSEARCH_WITH_CALLBACKS	1805 MIP dynamic search incompatible with control callbacks
CPXERR_MISS_SOS_TYPE	3301 Line %d: Missing SOS type
CPXERR_MSG_NO_CHANNEL	1051 No channel pointer supplied to message routine
CPXERR_MSG_NO_FILEPTR	1052 No file pointer found for message routine
CPXERR_MSG_NO_FUNCTION	1053 No function pointer found for message routine
CPXERR_NAME_CREATION	1209 Unable to create default names
CPXERR_NAME_NOT_FOUND	1210 Name not found
CPXERR_NAME_TOO_LONG	1464 Line %d: Identifier/name too long to process
CPXERR_NAN	1225 Numeric entry %d is not a double precision number (NAN)
CPXERR_NEED_OPT_SOLN	1252 Optimal solution required
CPXERR_NEGATIVE_SURPLUS	1207 Insufficient array length
CPXERR_NET_DATA	1530 Inconsistent network file
CPXERR_NET_FILE_SHORT	1538 Unexpected end of network file
CPXERR_NO_BARRIER_SOLN	1223 No barrier solution exists
CPXERR_NO_BASIC_SOLN	1261 No basic solution exists
CPXERR_NO_BASIS	1262 No basis exists
CPXERR_NO_BOUND_SENSE	1621 Line %d: No bound sense
CPXERR_NO_BOUND_TYPE	1460 Line %d: Bound type missing
CPXERR_NO_COLUMNS_SECTION	1472 Line %d: No COLUMNS section
CPXERR_NO_CONFLICT	1719 No conflict is available.
CPXERR_NO_DUAL_SOLN	1232 No dual solution exists
CPXERR_NO_ENDATA	1552 ENDATA missing
CPXERR_NO_ENVIRONMENT	1002 No environment
CPXERR_NO_FILENAME	1421 File name not specified
CPXERR_NO_ID	1616 Line %d: Expected identifier, found '%c'
CPXERR_NO_ID_FIRST	1609 Line %d: Expected identifier first
CPXERR_NO_INT_X	3023 Integer feasible solution values are unavailable
CPXERR_NO_LU_FACTOR	1258 No LU factorization exists
CPXERR_NO_MEMORY	1001 Out of memory

CPXERR_NO_MIPSTART	3020 No MIP start exists
CPXERR_NO_NAME_SECTION	1441 Line %d: No NAME section
CPXERR_NO_NAMES	1219 No names exist
CPXERR_NO_NORMS	1264 No norms available
CPXERR_NO_NUMBER	1615 Line %d: Expected number, found '%c'
CPXERR_NO_NUMBER_BOUND	1623 Line %d: Missing bound number
CPXERR_NO_NUMBER_FIRST	1611 Line %d: Expected number first
CPXERR_NO_OBJ_SENSE	1436 Max or Min missing
CPXERR_NO_OBJECTIVE	1476 Line %d: No objective row found
CPXERR_NO_OP_OR_SENSE	1608 Line %d: Expected '+','-' or sense, found '%c'
CPXERR_NO_OPERATOR	1607 Line %d: Expected '+' or '-', found '%c'
CPXERR_NO_ORDER	3016 No priority order exists
CPXERR_NO_PROBLEM	1009 No problem exists
CPXERR_NO_QMATRIX_SECTION	1461 Line %d: No QMATRIX section
CPXERR_NO_QP_OPERATOR	1614 Line %d: Expected ^ or *
CPXERR_NO_QUAD_EXP	1612 Line %d: Expected quadratic exponent
CPXERR_NO_RHS_COEFF	1610 Line %d: Expected RHS coefficient
CPXERR_NO_RHS_IN_OBJ	1211 rhs has no coefficient in obj
CPXERR_NO_RNGVAL	1216 No range values
CPXERR_NO_ROW_NAME	1486 Line %d: No row name
CPXERR_NO_ROW_SENSE	1453 Line %d: No row sense
CPXERR_NO_ROWS_SECTION	1471 Line %d: No ROWS section
CPXERR_NO_SENSIT	1260 Sensitivity analysis not available for current status
CPXERR_NO_SOLN	1217 No solution exists
CPXERR_NO_SOLNPOOL	3024 No solution pool exists
CPXERR_NO_SOS	3015 No user-defined SOSs exist
CPXERR_NO_SOS_SEPARATOR	1627 Expected ':', found '%c'
CPXERR_NO_TREE	3412 Current problem has no tree
CPXERR_NO_VECTOR_SOLN	1556 Vector solution does not exist
CPXERR_NODE_INDEX_RANGE	1230 Node index %d out of range
CPXERR_NODE_ON_DISK	3504 No callback info on disk/compressed nodes
CPXERR_NOT_DUAL_UNBOUNDED	1265 Dual unbounded solution required
CPXERR_NOT_FIXED	1221 Only fixed variables are pivoted out
CPXERR_NOT_FOR_MIP	1017 Not available for mixed-integer problems
CPXERR_NOT_FOR_QCP	1031 Not available for QCP
CPXERR_NOT_FOR_QP	1018 Not available for quadratic programs
CPXERR_NOT_MILPCLASS	1024 Not a MILP or fixed MILP
CPXERR_NOT_MIN_COST_FLOW	1531 Not a min-cost flow problem
CPXERR_NOT_MIP	3003 Not a mixed-integer problem
CPXERR_NOT_MIQPCLASS	1029 Not a MIQP or fixed MIQP
CPXERR_NOT_ONE_PROBLEM	1023 Not a single problem

CPXERR_NOT_QP	5004 Not a quadratic program
CPXERR_NOT_SAV_FILE	1560 File '%s' is not a SAV file
CPXERR_NOT_UNBOUNDED	1254 Unbounded solution required
CPXERR_NULL_NAME	1224 Null pointer %d in name array
CPXERR_NULL_POINTER	1004 Null pointer for required data
CPXERR_ORDER_BAD_DIRECTION	3007 Illegal direction entry %d
CPXERR_PARAM_INCOMPATIBLE	1807 Incompatible parameters
CPXERR_PARAM_TOO_BIG	1015 Parameter value too big
CPXERR_PARAM_TOO_SMALL	1014 Parameter value too small
CPXERR_PRESLV_ABORT	1106 Aborted during presolve
CPXERR_PRESLV_BAD_PARAM	1122 Bad presolve parameter setting
CPXERR_PRESLV_BASIS_MEM	1107 Not enough memory to build basis for original LP
CPXERR_PRESLV_COPYORDER	1109 Can't copy priority order info from original MIP
CPXERR_PRESLV_COPYSOS	1108 Can't copy SOS info from original MIP
CPXERR_PRESLV_CRUSHFORM	1121 Can't crush solution form
CPXERR_PRESLV_DUAL	1119 The feature is not available for solving dual formulation
CPXERR_PRESLV_FAIL_BASIS	1114 Could not load unpresolved basis for original LP
CPXERR_PRESLV_INF	1117 Presolve determines problem is infeasible
CPXERR_PRESLV_INForUNBD	1101 Presolve determines problem is infeasible or unbounded
CPXERR_PRESLV_NO_BASIS	1115 Failed to find basis in presolved LP
CPXERR_PRESLV_NO_PROB	1103 No presolved problem created
CPXERR_PRESLV_SOLN_MIP	1110 Not enough memory to recover solution for original MIP
CPXERR_PRESLV_SOLN_QP	1111 Not enough memory to compute solution to original QP
CPXERR_PRESLV_START_LP	1112 Not enough memory to build start for original LP
CPXERR_PRESLV_TIME_LIM	1123 Time limit exceeded during presolve
CPXERR_PRESLV_UNBD	1118 Presolve determines problem is unbounded
CPXERR_PRESLV_UNCRUSHFORM	1120 Can't uncrush solution form
CPXERR_PRIIND	1257 Incorrect usage of pricing indicator
CPXERR_PRM_DATA	1660 Line %d: Not enough entries
CPXERR_PRM_HEADER	1661 Line %d: Missing or invalid header
CPXERR_PTHREAD_CREATE	3603 Could not create thread
CPXERR_PTHREAD_MUTEX_INIT	3601 Could not initialize mutex
CPXERR_Q_DIVISOR	1619 Line %d: Missing or incorrect divisor for Q terms
CPXERR_Q_DUP_ENTRY	5011 Duplicate entry for pair '%s' and '%s'
CPXERR_Q_NOT_INDEF	5014 Q is not indefinite
CPXERR_Q_NOT_POS_DEF	5002 Q in '%s' is not positive semi-definite
CPXERR_Q_NOT_SYMMETRIC	5012 Q is not symmetric
CPXERR_QCP_SENSE	6002 Illegal quadratic constraint sense
CPXERR_QCP_SENSE_FILE	1437 Line %d: Illegal quadratic constraint sense

CPXERR_QUAD_EXP_NOT_2	1613 Line %d: Quadratic exponent must be 2
CPXERR_QUAD_IN_ROW	1605 Line %d: Illegal quadratic term in a constraint
CPXERR_RANGE_SECTION_ORDER	1474 Line %d: 'RANGES' section out of order
CPXERR_RESTRICTED_VERSION	1016 Promotional version. Problem size limits exceeded
CPXERR_RHS_IN_OBJ	1603 Line %d: RHS sense in objective
CPXERR_RIM_REPEATS	1447 Line %d: %s '%s' repeats
CPXERR_RIM_ROW_REPEATS	1444 %s '%s' has repeated row '%s'
CPXERR_RIMNZ_REPEATS	1479 Line %d: %s %s repeats
CPXERR_ROW_INDEX_RANGE	1203 Row index %d out of range
CPXERR_ROW_REPEAT_PRINT	1477 %d Row repeats messages not printed
CPXERR_ROW_REPEATS	1445 Row '%s' repeats
CPXERR_ROW_UNKNOWN	1448 Line %d: '%s' is not a row name
CPXERR_SAV_FILE_DATA	1561 Not enough data in SAV file
CPXERR_SAV_FILE_WRITE	1562 Unable to write SAV file to disk
CPXERR_SBASE_ILLEGAL	1554 Superbases are not allowed
CPXERR_SBASE_INCOMPAT	1255 Incompatible with superbasis
CPXERR_SINGULAR	1256 Basis singular
CPXERR_STR_PARAM_TOO_LONG	1026 String parameter is too long
CPXERR_SUBPROB_SOLVE	3019 Failure to solve MIP subproblem
CPXERR_THREAD_FAILED	1234 Creation of parallel thread failed.
CPXERR_TILIM_CONDITION_NO	1268 Time limit reached in computing condition number
CPXERR_TILIM_STRONGBRANCH	1266 Time limit reached in strong branching
CPXERR_TOO_MANY_COEFFS	1433 Too many coefficients
CPXERR_TOO_MANY_COLS	1432 Too many columns
CPXERR_TOO_MANY_RIMNZ	1485 Too many rim nonzeros
CPXERR_TOO_MANY_RIMS	1484 Too many rim vectors
CPXERR_TOO_MANY_ROWS	1431 Too many rows
CPXERR_TOO_MANY_THREADS	1020 Thread limit exceeded
CPXERR_TREE_MEMORY_LIMIT	3413 Tree memory limit exceeded
CPXERR_UNIQUE_WEIGHTS	3010 Set does not have unique weights
CPXERR_UNSUPPORTED_CONSTRAINT_TYPE	1212 Unsupported constraint type was used.
CPXERR_UP_BOUND_REPEATS	1458 Line %d: Repeated upper bound
CPXERR_WORK_FILE_OPEN	1801 Could not open temporary file
CPXERR_WORK_FILE_READ	1802 Failure on temporary file read
CPXERR_WORK_FILE_WRITE	1803 Failure on temporary file write
CPXERR_XMLPARSE	1425 XML parsing error at line %d: %s

Each error code, such as 1616, is associated with a symbolic constant, such as `CPXERR_NO_ID`, and a short message string, such as Line %d: Expected identifier, found '%c'.

In the short message strings, the following symbols occur:

%d means a number, such as a line number

%s means a string, such as a file name, variable name, or other

%c means a character, such as a letter or arithmetic operator

Click the symbolic constant in the table to go to a longer explanation of an error code.

Group optim.cplex.solutionquality

The Callable Library macros that indicate the qualities of a solution, their symbolic constants, and their meaning. Methods for accessing solution quality are mentioned after the table.

Macro Summary	
CPX_DUAL_OBJ	Concert Technology enum: DualObj.
CPX_EXACT_KAPPA	Concert Technology enum: ExactKappa.
CPX_KAPPA	Concert Technology enum: Kappa.
CPX_MAX_COMP_SLACK	Concert Technology enum: MaxCompSlack.
CPX_MAX_DUAL_INFEAS	Concert Technology enum: MaxDualInfeas.
CPX_MAX_DUAL_RESIDUAL	Concert Technology enum: MaxDualResidual.
CPX_MAX_INDSLACK_INFEAS	Concert Technology enum: not applicable.
CPX_MAX_INT_INFEAS	Concert Technology enum: MaxIntInfeas.
CPX_MAX_PI	Concert Technology enum: MaxPi.
CPX_MAX_PRIMAL_INFEAS	Concert Technology enum: MaxPrimalInfeas.
CPX_MAX_PRIMAL_RESIDUAL	Concert Technology enum: MaxPrimalResidual.
CPX_MAX_QCPRIMAL_RESIDUAL	Concert Technology enum: MaxPrimalResidual.
CPX_MAX_QCSLACK	Concert Technology enum: not applicable.
CPX_MAX_QCSLACK_INFEAS	Concert Technology enum: not applicable.
CPX_MAX_RED_COST	Concert Technology enum: MaxRedCost.
CPX_MAX_SCALED_DUAL_INFEAS	Concert Technology enum: MaxScaledDualInfeas.
CPX_MAX_SCALED_DUAL_RESIDUAL	Concert Technology enum: MaxScaledDualResidual.
CPX_MAX_SCALED_PI	Concert Technology enum: MaxScaledPi.
CPX_MAX_SCALED_PRIMAL_INFEAS	Concert Technology enum: MaxScaledPrimalInfeas.
CPX_MAX_SCALED_PRIMAL_RESIDUAL	Concert Technology enum: MaxScaledPrimalResidual.
CPX_MAX_SCALED_RED_COST	Concert Technology enum: MaxScaledRedCost.
CPX_MAX_SCALED_SLACK	Concert Technology enum: MaxScaledSlack.
CPX_MAX_SCALED_X	Concert Technology enum: MaxScaledX.
CPX_MAX_SLACK	Concert Technology enum: MaxSlack.
CPX_MAX_X	Concert Technology enum: MaxX.
CPX_OBJ_GAP	Concert Technology enum: ObjGap.
CPX_PRIMAL_OBJ	Concert Technology enum: PrimalObj.
CPX_SUM_COMP_SLACK	Concert Technology enum: SumCompSlack.
CPX_SUM_DUAL_INFEAS	Concert Technology enum: SumDualInfeas.
CPX_SUM_DUAL_RESIDUAL	Concert Technology enum: SumDualResidual.
CPX_SUM_INDSLACK_INFEAS	Concert Technology enum: not applicable.
CPX_SUM_INT_INFEAS	Concert Technology enum: SumIntInfeas.
CPX_SUM_PI	Concert Technology enum: SumPi.
CPX_SUM_PRIMAL_INFEAS	Concert Technology enum: SumPrimalInfeas.
CPX_SUM_PRIMAL_RESIDUAL	Concert Technology enum: SumPrimalResidual.
CPX_SUM_QCPRIMAL_RESIDUAL	Concert Technology enum: SumPrimalResidual.

CPX_SUM_QCSLACK	Concert Technology enum: SumSlack.
CPX_SUM_QCSLACK_INFEAS	Concert Technology enum: not applicable.
CPX_SUM_RED_COST	Concert Technology enum: SumRedCost.
CPX_SUM_SCALED_DUAL_INFEAS	Concert Technology enum: SumScaledDualInfeas.
CPX_SUM_SCALED_DUAL_RESIDUAL	Concert Technology enum: SumScaledDualResidual.
CPX_SUM_SCALED_PI	Concert Technology enum: SumScaledPi.
CPX_SUM_SCALED_PRIMAL_INFEAS	Concert Technology enum: SumScaledPrimalInfeas.
CPX_SUM_SCALED_PRIMAL_RESIDUAL	Concert Technology enum: SumScaledPrimalResidual.
CPX_SUM_SCALED_RED_COST	Concert Technology enum: SumScaledRedCost.
CPX_SUM_SCALED_SLACK	Concert Technology enum: SumScaledSlack.
CPX_SUM_SCALED_X	Concert Technology enum: SumScaledX.
CPX_SUM_SLACK	Concert Technology enum: SumSlack.
CPX_SUM_X	Concert Technology enum: SumX.

This table lists quality values.

Values that are stored in a numeric variable or double variable are accessed by the Concert Technology method `getQuality` of the class `IloCplex` or by the Callable Library routine `CPXgetdblquality`.

Values that are stored in an integer variable are accessed by the method `getQuality` of the class `IloCplex` or by the routine `CPXgetintquality`.

Group optim.cplex.solutionstatus

The Callable Library macros that define solution status, their symbolic constants, their equivalent in Concert Technology enumerations, and their meaning. There is a note about unboundedness after the table.

Macro Summary	
CPX_STAT_ABORT_DUAL_OBJ_LIM	22 (Barrier only) enum: AbortDualObjLim
CPX_STAT_ABORT_IT_LIM	10 (Simplex or Barrier) enum: AbortItLim
CPX_STAT_ABORT_OBJ_LIM	12 (Simplex or Barrier) enum: AbortObjLim
CPX_STAT_ABORT_PRIM_OBJ_LIM	21 (Barrier only) enum: AbortPrimObjLim
CPX_STAT_ABORT_TIME_LIM	11 (Simplex or Barrier) enum: AbortTimeLim
CPX_STAT_ABORT_USER	13 (Simplex or Barrier) enum: AbortUser
CPX_STAT_CONFLICT_ABORT_CONTRADICTION	32 (conflict refiner) enum: ConflictAbortContradiction
CPX_STAT_CONFLICT_ABORT_IT_LIM	34 (conflict refiner) enum: ConflictAbortItLim
CPX_STAT_CONFLICT_ABORT_MEM_LIM	37 (conflict refiner) enum: ConflictAbortMemLim
CPX_STAT_CONFLICT_ABORT_NODE_LIM	35 (conflict refiner) enum: ConflictAbortNodeLim
CPX_STAT_CONFLICT_ABORT_OBJ_LIM	36 (conflict refiner) enum: ConflictAbortObjLim
CPX_STAT_CONFLICT_ABORT_TIME_LIM	33 (conflict refiner) enum: ConflictAbortTimeLim
CPX_STAT_CONFLICT_ABORT_USER	38 (conflict refiner) enum: ConflictAbortUser
CPX_STAT_CONFLICT_FEASIBLE	30 (conflict refiner) enum: ConflictFeasible
CPX_STAT_CONFLICT_MINIMAL	31 (conflict refiner) enum: ConflictMinimal
CPX_STAT_FEASIBLE	23 (Simplex or Barrier) enum: Feasible
CPX_STAT_FEASIBLE_RELAXED_INF	16 (Simplex or Barrier) enum: FeasibleRelaxedInf
CPX_STAT_FEASIBLE_RELAXED_QUAD	18 (Simplex or Barrier) enum: FeasibleRelaxedQuad
CPX_STAT_FEASIBLE_RELAXED_SUM	14 (Simplex or Barrier) enum: FeasibleRelaxedSum
CPX_STAT_INFEASIBLE	3 (Simplex or Barrier) enum: Infeasible
CPX_STAT_INForUNBD	4 (Simplex or Barrier) enum: InfOrUnbd
CPX_STAT_NUM_BEST	6 (Simplex or Barrier) enum: NumBest
CPX_STAT_OPTIMAL	1 (Simplex or Barrier) enum: Optimal
CPX_STAT_OPTIMAL_FACE_UNBOUNDED	20 (Barrier only) enum: OptimalFaceUnbounded
CPX_STAT_OPTIMAL_INFEAS	5 (Simplex or Barrier) enum: OptimalInfeas
CPX_STAT_OPTIMAL_RELAXED_INF	17 (Simplex or Barrier) enum: OptimalRelaxedInf
CPX_STAT_OPTIMAL_RELAXED_QUAD	19 (Simplex or Barrier) enum: OptimalRelaxedQuad
CPX_STAT_OPTIMAL_RELAXED_SUM	15 (Simplex or Barrier) enum: OptimalRelaxedSum
CPX_STAT_UNBOUNDED	2 (Simplex or Barrier) enum: Unbounded
CPXMIP_ABORT_FEAS	113 (MIP only) enum: AbortFeas
CPXMIP_ABORT_INFEAS	114 (MIP only) enum: AbortInfeas
CPXMIP_ABORT_RELAXED	126 (MIP only) enum: AbortRelaxed
CPXMIP_FAIL_FEAS	109 (MIP only) enum: FailFeas
CPXMIP_FAIL_FEAS_NO_TREE	116 (MIP only) enum: FailFeasNoTree
CPXMIP_FAIL_INFEAS	110 (MIP only) enum: FailInfeas
CPXMIP_FAIL_INFEAS_NO_TREE	117 (MIP only) enum: FailInfeasNoTree

CPXMIP_FEASIBLE	127 (MIP only) enum: Feasible
CPXMIP_FEASIBLE_RELAXED_INF	122 (MIP only) enum: FeasibleRelaxedInf
CPXMIP_FEASIBLE_RELAXED_QUAD	124 (MIP only) enum: FeasibleRelaxedQuad
CPXMIP_FEASIBLE_RELAXED_SUM	120 (MIP only) enum: FeasibleRelaxedSum
CPXMIP_INFEASIBLE	103 (MIP only) enum: Infeasible
CPXMIP_INForUNBD	119 (MIP only) enum: InfOrUnbd
CPXMIP_MEM_LIM_FEAS	111 (MIP only) enum: MemLimFeas
CPXMIP_MEM_LIM_INFEAS	112 (MIP only) enum: MemLimInfeas
CPXMIP_NODE_LIM_FEAS	105 (MIP only) enum: NodeLimFeas
CPXMIP_NODE_LIM_INFEAS	106 (MIP only) enum: NodeLimInfeas
CPXMIP_OPTIMAL	101 (MIP only) enum: Optimal
CPXMIP_OPTIMAL_INFEAS	115 (MIP only) enum: OptimalInfeas
CPXMIP_OPTIMAL_POPULATED	129 (MIP only) enum: OptimalPopulated
CPXMIP_OPTIMAL_POPULATED_TOL	130 (MIP only) enum: OptimalPopulatedTol
CPXMIP_OPTIMAL_RELAXED_INF	123 (MIP only) enum: OptimalRelaxedInf
CPXMIP_OPTIMAL_RELAXED_QUAD	125 (MIP only) enum: OptimalRelaxedQuad
CPXMIP_OPTIMAL_RELAXED_SUM	121 (MIP only) enum: OptimalRelaxedSum
CPXMIP_OPTIMAL_TOL	102 (MIP only) enum: OptimalTol
CPXMIP_POPULATESOL_LIM	128 (MIP only) enum: PopulateSolLim
CPXMIP_SOL_LIM	104 (MIP only) enum: SolLim
CPXMIP_TIME_LIM_FEAS	107 (MIP only) enum: TimeLimFeas
CPXMIP_TIME_LIM_INFEAS	108 (MIP only) enum: TimeLimInfeas
CPXMIP_UNBOUNDED	118 (MIP only) enum: Unbounded

This table lists the statuses for solutions to LP, QP, or MIP problems. These values are returned by the Callable Library routine `CPXgetstat` or by the Concert Technology methods `getCplexStatus` and `getCplexSubStatus` of the class `IloCplex`. If no solution exists, the return value is zero.

About Unboundedness

The treatment of models that are unbounded involves a few subtleties. Specifically, a declaration of unboundedness means that CPLEX has determined that the model has an unbounded ray. Given any feasible solution x with objective z, a multiple of the unbounded ray can be added to x to give a feasible solution with objective z-1 (or z+1 for maximization models). Thus, if a feasible solution exists, then the optimal objective is unbounded. Note that CPLEX has not necessarily concluded that a feasible solution exists. Users can call the routine `CPXsolninfo` to determine whether CPLEX has also concluded that the model has a feasible solution.

Global function CPXaddfuncdest

```
int CPXaddfuncdest(CPXCENVptr env, CPXCHANNELptr channel, void * handle,
void(CPXPUBLIC *msgfunction)(void *, const char *))
```

Definition file: cplex.h

The routine `CPXaddfuncdest` adds a function `msgfunction` to the message destination list for a channel. This routine allows users to trap messages instead of printing them. That is, when a message is sent to the channel, each destination that was added to the message destination list by `CPXaddfuncdest` calls its associated message.

To illustrate, consider an application in which a developer wishes to trap CPLEX error messages and display them in a dialog box that prompts the user for an action. Use `CPXaddfuncdest` to add the address of a function to the list of message destinations associated with the cpxerror channel. Then write the `msgfunction` routine. It must contain the code that controls the dialog box. When `CPXmsg` is called with `cpxerror` as its first argument, it calls the `msgfunction` routine, which can then display the error message.

Note

The argument `handle` is a generic pointer that can be used to hold information needed by the `msgfunction` routine to avoid making such information global to all routines.

Example

```
void msgfunction (void *handle, char *msg_string)
{
    FILE *fp;
    fp = (FILE *)handle;
    fprintf (fp, "%s", msg_string);
}
status = CPXaddfuncdest (env, mychannel, fileptr, msgfunction);
```

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`channel`

A pointer to the channel to which the function destination is to be added.

`handle`

A void pointer that can be used to pass arbitrary information into `msgfunction`.

`msgfunction`

A pointer to the function to be called when a message is sent to a channel.

See Also: CPXdelfuncdest

Returns:

The routine returns zero if successful and nonzero if an error occurs. Failure occurs when `msgfunction` is not in the message-destination list or the channel does not exist.

Global function CPXmstwrite

`int` **`CPXmstwrite`**`(CPXCENVptr env, CPXCLPptr lp, const char * filename_str)`

Definition file: cplex.h

This routine is **deprecated**. Use `CPXwritemipstarts` instead.

The routine `CPXmstwrite` writes the incumbent MIP start to a file in MST format.

The MST format is an XML format and is documented in the stylesheet `solution.xsl` and schema `solution.xsd` in the `include` directory of the CPLEX distribution. *CPLEX File Formats Reference Manual* also documents this format briefly.

See Also: CPXwritemipstarts

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the incumbent MIP start information should be written.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetmiprelgap

`int` **`CPXgetmiprelgap`**`(CPXCENVptr env, CPXCLPptr lp, double * gap_p)`

Definition file: cplex.h

The routine `CPXgetmiprelgap` accesses the relative objective gap for a MIP optimization.

For a **minimization** problem, this value is computed by

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

where `bestinteger` is the value returned by `CPXgetobjval` and `bestobjective` is the value returned by `CPXgetbestobjval`. For a **maximization** problem, the value is computed by:

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

Example

```
status = CPXgetmiprelgap (env, lp, &gap);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
gap_p	A pointer to the location where the relative mip gap is returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXkillpnorms

`void` **`CPXkillpnorms`**`(CPXLPptr lp)`

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXkillpnorms` deletes any primal steepest-edge norms that have been retained relative to an active basis. If the user believes that the values of these norms may be significantly in error, and the setting of the parameter `CPX_PARAM_PPRIIND` is `CPX_PPRIIND_STEEP`, calling `CPXkillpnorms` means that fresh primal steepest-edge norms will be computed on the next call to `CPXprimopt`.

Parameters:

lp A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.

Global function CPXdualwrite

```
int CPXdualwrite(CPXCENVptr env, CPXCLPptr lp, const char * filename_str, double *
objshift_p)
```

Definition file: cplex.h

The routine `CPXdualwrite` writes a dual formulation of the current CPLEX problem object. MPS format is used. This function can only be applied to a linear program; it generates an error for other problem types.

> **Note**
>
> Any fixed variables in the primal are removed before the dual problem is written to a file. Each fixed variable with a nonzero objective coefficient causes the objective value to shift. As a result, if fixed variables are present, the optimal objective obtained from solving the dual problem created using `CPXdualwrite` may not be the same as the optimal objective of the primal problem. The argument `objshift_p` can be used to reconcile this difference.

Example

```
status = CPXdualwrite (env, lp, "myfile.dua", &objshift);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the dual problem should be written.
objshift_p	A pointer to a variable of type `double` to hold the change in the objective function resulting from the removal of fixed variables in the primal.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcrushx

```
int CPXcrushx(CPXCENVptr env, CPXCLPptr lp, const double * x, double * prex)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXcrushx` crushes a solution for the original problem to a solution for the presolved problem.

Example

```
status = CPXcrushx (env, lp, origx, reducex);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
x	An array that contains primal solution (x) values for the original problem, as returned by routines such as `CPXgetx` or `CPXsolution`. The array must be of length at least the number of columns in the problem object.
prex	An array to receive the primal values corresponding to the presolved problem. The array must be of length at least the number of columns in the presolved problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

See `admipex6.c` in the *CPLEX User's Manual*.

Global function CPXgetslack

```
int CPXgetslack(CPXCENVptr env, CPXCLPptr lp, double * slack, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetslack` accesses the slack values for a range of linear constraints. The beginning and end of the range must be specified. Except for ranged rows, the slack values returned consist of the righthand side minus the row activity level. For ranged rows, the value returned is the row activity level minus the righthand side, or, equivalently, the value of the internal structural variable that CPLEX creates to represent ranged rows.

Example

```
status = CPXgetslack (env, lp, slack, 0, CPXgetnumrows(env,lp)-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
slack	An array to receive the values of the slack or surplus variables for each of the constraints. This array must be of length at least (`end - begin + 1`). If successful, `slack[0]` through `slack[end-begin]` contain the values of the slacks.
begin	An integer specifying the beginning of the range of slack values to be returned.
end	An integer specifying the end of the range of slack values to be returned.

Example

```
status = CPXgetslack (env, lp, slack, 0, CPXgetnumrows(env,lp)-1);
```

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsetdblparam

`int` **`CPXsetdblparam`**`(CPXENVptr env, int whichparam, double newvalue)`

Definition file: cplex.h

The routine `CPXsetdblparam` sets the value of a CPLEX parameter of type `double`.

The *CPLEX Parameters Reference Manual* provides a list of parameters with their types, options, and default values.

Example

```
status = CPXsetdblparam (env, CPX_PARAM_TILIM, 1000.0);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
whichparam	The symbolic constant (or reference number) of the parameter to change.
newvalue	The new value of the parameter.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsosindex

```
int CPXgetsosindex(CPXCENVptr env, CPXCLPptr lp, const char * lname_str, int *
index_p)
```

Definition file: cplex.h

The routine `CPXgetsosindex` searches for the index number of the specified special ordered set in a CPLEX problem object.

Example

```
status = CPXgetsosindex (env, lp, "set5", &setindex);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lname_str	A special ordered set name to search for.
index_p	A pointer to an integer to hold the index number of the special ordered set with name `lname_str`. If the routine is successful, `*index_p` contains the index number; otherwise, `*index_p` is undefined.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetnumsemiint

`int` **`CPXgetnumsemiint`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetnumsemiint` accesses the number of semi-integer variables in a CPLEX problem object.

Example

```
numsc = CPXgetnumsemiint (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If the problem object or environment does not exist, `CPXgetnumsemiint` returns the value 0 (zero); otherwise, it returns the number of semi-integer variables in the problem object.

Global function CPXcheckcopysos

`int` **`CPXcheckcopysos`**`(CPXCENVptr env, CPXCLPptr lp, int numsos, int numsosnz, const char * sostype, const int * sosbeg, const int * sosind, const double * soswt, char ** sosname)`

Definition file: cplex.h

The routine `CPXcheckcopysos` validates the arguments of the corresponding `CPXcopysos` routine. This data checking routine is found in source format in the file `check.c` which is provided with the standard CPLEX distribution. To call this routine, you must compile and link `check.c` with your program as well as the CPLEX Callable Library.

The `CPXcheckcopysos` routine has the same argument list as the `CPXcopysos` routine. The second argument, `lp`, is technically a pointer to a constant LP object of type `CPXCLPptr` rather than type `CPXLPptr`, as this routine will not modify the problem. For most user applications, this distinction is unimportant.

Example

```
status = CPXcheckcopysos (env, lp, numsos, numsosnz, sostype,
                          sosbeg, sosind, soswt, sosname);
```

Returns:

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.

Global function CPXgetcallbackincumbent

`int` **`CPXgetcallbackincumbent`**`(CPXCENVptr env, void * cbdata, int wherefrom, double * x, int begin, int end)`

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbackincumbent` retrieves the incumbent values during MIP optimization from within a user-written callback. The values are from the original problem if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF` or if the routine is called from an informational callback. Otherwise, they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

Example

```
status = CPXgetcallbackincumbent (env, cbdata, wherefrom,
                                  bestx, 0, cols-1);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting from where the user-written callback was called. The argument must be the value of `wherefrom` passed to the user-written callback.
x	An array to receive the values of the incumbent (best available) integer solution. This array must be of length at least (`end - begin + 1`). If successful, `x[0]` through `x[end-begin]` contain the incumbent values.
begin	An integer specifying the beginning of the range of incumbent values to be returned.
end	An integer specifying the end of the range of incumbent values to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcopyquad

```
int CPXcopyquad(CPXCENVptr env, CPXLPptr lp, const int * qmatbeg, const int *
qmatcnt, const int * qmatind, const double * qmatval)
```

Definition file: cplex.h

The routine `CPXcopyquad` is used to copy a quadratic objective matrix Q when Q is not diagonal. The arguments `qmatbeg`, `qmatcnt`, `qmatind`, and `qmatval` are used to specify the nonzero coefficients of the matrix Q. The meaning of these vectors is identical to the meaning of the corresponding vectors `matbeg`, `matcnt`, `matind` and `matval`, which are used to specify the structure of A in a call to `CPXcopylp`.

Q must be symmetric when copied by this function. Therefore, if the quadratic coefficient in algebraic form is *2x1x2*, then *x2* should be in the list for *x1*, and *x1* should be in the list for *x2*, and the coefficient would be 1.0 in each of those entries. See the corresponding example C program to review how the symmetry requirement is implemented.

Note

CPLEX evaluates the corresponding objective with a factor of `0.5` in front of the quadratic objective term.

When you build or modify your model with this routine, you can verify that the results are as you intended by calling `CPXcheckcopyquad` during application development.

How the arrays are accessed

Suppose that CPLEX wants to access the entries in a column `j`. These are assumed to be given by the array entries:

```
qmatval[qmatbeg[j]],..,qmatval[qmatbeg[j]+qmatcnt[j]-1]
```

The corresponding column/index entries are:

```
qmatind[qmatbeg[j]],..,qmatind[qmatbeg[j]+qmatcnt[j]-1
```

The entries in `qmatind[k]` are not required to be in column order. Duplicate entries in `qmatind` within a single column are not allowed. Note that any column `j` that has only a linear objective term has `qmatcnt[j] = 0` and no entries in `qmatind` and `qmatval`.

Example

```
status = CPXcopyquad (env, lp, qmatbeg, qmatcnt, qmatind,
                      qmatval);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
qmatbeg	An array that with `qmatcnt`, `qmatind`, and `qmatval` defines the quadratic coefficient matrix.
qmatcnt	An array that with `qmatbeg`, `qmatind`, and `qmatval` defines the quadratic coefficient matrix.
qmatind	An array that with `qmatbeg`, `qmatcnt`, and `qmatval` defines the quadratic coefficient matrix.
qmatval	An array that with `qmatbeg`, `qmatcnt`, and `qmatind` defines the quadratic coefficient matrix. The arrays `qmatbeg` and `qmatcnt` should be of length at least `CPXgetnumcols(env,lp)`. The arrays `qmatind` and `qmatval` should be of length at least `qmatbeg[numcols-1]+qmatcnt[numcols-1]`. CPLEX requires only the nonzero coefficients grouped by column in the array `qmatval`. The nonzero elements of every column must be stored in sequential locations in this array with `qmatbeg[j]` containing the index of the beginning of column `j` and `qmatcnt[j]` containing the number of entries in column `j`. Note that the components of

`qmatbeg` must be in ascending order. For each `k`, `qmatind[k]` indicates the column number of the corresponding coefficient, `qmatval[k]`. These arrays are accessed as explained above.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXNETgetphase1cnt

```
int CPXNETgetphase1cnt(CPXCENVptr env, CPXCNETptr net)
```

Definition file: cplex.h

The routine `CPXNETgetphase1cnt` returns the number of phase 1 network simplex iterations for the most recent call to `CPXNETprimopt`.

Example

```
phase1cnt = CPXNETgetphase1cnt (env, net);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

net A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

Returns:

Returns the total number of phase 1 network simplex iterations for the last call to `CPXNETprimopt`, for a `CPXNETptr` object. If `CPXNETprimopt` has not been called, zero is returned. If an error occurs, -1 is returned and an error message is issued.

Global function CPXbinvarow

```
int CPXbinvarow(CPXCENVptr env, CPXCLPptr lp, int i, double * z)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXbinvarow` computes the i-th row of **BinvA** where **Binv** represents the inverse of the matrix B and juxtaposition specifies matrix multiplication. In other words, it computes the i-th row of the tableau.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
i	An integer that specifies the index of the row to be computed.
z	An array containing the i-th row of **BinvA**. The array must be of length at least equal to the number of columns in the problem.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETchgobj

`int` **`CPXNETchgobj`**`(CPXCENVptr env, CPXNETptr net, int cnt, const int * indices, const double * obj)`

Definition file: cplex.h

The routine `CPXNETchgobj` is used to change the objective values for a set of arcs in the network stored in a network problem object.

Any solution information stored in the problem object is lost.

Example

```
status = CPXNETchgobj (env, net, cnt, indices, newobj);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
cnt	Number of arcs for which the objective values are to be changed.
indices	An array of indices that indicate the arcs for which the objective values are to be changed. This array must have a length of at least `cnt`. The indices must be in the range [`0, narcs-1`].
obj	An array of the new objective values for the arcs. This array must have a length of at least `cnt`.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXNETgetnumnodes

```
int CPXNETgetnumnodes(CPXCENVptr env, CPXCNETptr net)
```

Definition file: cplex.h

The routine `CPXNETgetnumnodes` is used to access the number of nodes in a network stored in a network problem object.

Example

```
cur_nnodes = CPXNETgetnumnodes (env, net);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

net A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

Returns:

The routine returns the number of network nodes stored in a network problem object. If an error occurs, 0 is returned and an error message is issued.

Global function CPXqconstrslackfromx

`int` **`CPXqconstrslackfromx`**`(CPXCENVptr env, CPXCLPptr lp, const double * x, double * qcslack)`

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXqconstrslackfromx` computes an array of slack values for quadratic constraints from primal solution values.

Example

```
status = CPXqconstrslackfromx (env, lp, x, qcslack);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
x	An array that contains primal solution (`x`) values for the problem, as returned by routines such as `CPXcrushx` and `CPXuncrushx`. The array must be of length at least the number of columns in the LP problem object.
qcslack	An array to receive the quadratic constraint slack values computed from the x values for the problem object. The array must be of length at least the number of quadratic constraints in the LP problem object.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetdblquality

`int` **`CPXgetdblquality`**`(CPXCENVptr env, CPXCLPptr lp, double * quality_p, int what)`

Definition file: cplex.h

The routine `CPXgetdblquality` accesses double-valued information about the quality of the current solution of a problem. A solution, though not necessarily a feasible or optimal one, must be available in the CPLEX problem object. The quality values are returned in the `double` variable pointed to by the argument `quality_p`.

The maximum bound infeasibility identifies the largest bound violation. Largest bound violation may help determine the cause of an infeasible problem. If the largest bound violation exceeds the feasibility tolerance by only a small amount, it may be possible to obtain a feasible solution to the problem by increasing the feasibility tolerance. If a problem is optimal, the largest bound violation gives insight into the smallest setting for the feasibility tolerance that would not cause the problem to terminate infeasibly.

Example

```
status = CPXgetdblquality (env, lp, &max_x, CPX_MAX_X);
```

Parameters:

env	A pointer to the CPLEX environment as returned by the `CPXopenCPLEX` routine.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
quality_p	A pointer to a `double` variable in which the requested quality value is to be stored. If an error occurs, the quality-value remains unchanged.
what	A symbolic constant specifying the quality value to be retrieved. The possible quality values for a solution are listed in the group optim.cplex.solutionquality in the *Callable Library Reference Manual*.

Returns:

The routine returns zero if successful and nonzero if an error occurs. If an error occurs, the quality-value remains unchanged.

Global function CPXsolninfo

```
int CPXsolninfo(CPXCENVptr env, CPXCLPptr lp, int * solnmethod_p, int * solntype_p,
int * pfeasind_p, int * dfeasind_p)
```

Definition file: cplex.h

The routine `CPXsolninfo` accesses solution information produced by the routines:

- `CPXlpopt`,
- `CPXprimopt`,
- `CPXdualopt`,
- `CPXbaropt`,
- `CPXhybbaropt`,
- `CPXhybnetopt`,
- `CPXqpopt`,
- `CPXfeasopt`, or
- `CPXmipopt`.

This information is maintained until the CPLEX problem object is freed by a call to `CPXfreeprob` or until the solution is rendered invalid because of a call to one of the problem modification routines.

The arguments to `CPXsolninfo` are pointers to locations where data are to be written. Such data can include the optimization method used to produce the current solution, the type of solution available, and what is known about the primal and dual feasibility of the current solution. If any piece of information represented by an argument to `CPXsolninfo` is not required, a NULL pointer can be passed for that argument.

Example

```
status = CPXsolninfo (env, lp, &solnmethod, &solntype,
                      &pfeasind, &dfeasind);
```

See also the topic *Interpreting Solution Quality* in the *CPLEX User's Manual* for information about how CPLEX determines primal or dual infeasibility.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
solnmethod_p	A pointer to an integer specifying the method used to produce the current solution. The specific values which `solnmethod_p` can take and their meanings are the same as the return values documented for `CPXgetmethod`.
solntype_p	A pointer to an integer variable specifying the type of solution currently available. Possible return values are `CPX_BASIC_SOLN`, `CPX_NONBASIC_SOLN`, `CPX_PRIMAL_SOLN`, and `CPX_NO_SOLN`, meaning the problem either has a simplex basis, has a primal and dual solution but no basis, has a primal solution but no corresponding dual solution, or has no solution, respectively.
pfeasind_p	A pointer to an integer variable specifying whether the current solution is known to be primal feasible. A false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude it was feasible when it terminated.
dfeasind_p	A pointer to an integer variable specifying whether the current solution is known to be dual feasible. A false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude it was feasible when it terminated.

Returns:

The routine returns zero if successful and it returns nonzero if an error occurs.

Global function CPXgetsolvecallbackfunc

```
void CPXgetsolvecallbackfunc(CPXCENVptr env, int(CPXPUBLIC
**solvecallback_p)(CALLBACK_SOLVE_ARGS), void ** cbhandle_p)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetsolvecallbackfunc` accesses the user-written callback to be called during MIP optimization to optimize the subproblem.

Example

```
CPXgetsolvecallbackfunc(env, &current_callback, &current_cbdata);
```

See also *Advanced MIP Control Interface* in the *CPLEX User's Manual*.

For documentation of callback arguments, see the routine `CPXsetsolvecallbackfunc`.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`solvecallback_p`

The address of the pointer to the current user-written solve callback. If no callback has been set, the pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the user's private pointer.

See Also: CPXgetcallbacknodelp, CPXsetsolvecallbackfunc

Returns:

This routine does not return a result.

Global function CPXmdleave

```
int CPXmdleave(CPXCENVptr env, CPXLPptr lp, const int * indices, int cnt, double *
downratio, double * upratio)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXmdleave` assumes that there is a resident optimal simplex basis, and a resident LU-factorization associated with this basis. It takes as input a list of basic variables as specified by `indices[]` and `cnt`, and returns values commonly known as Driebeek penalties in the two arrays `downratio[]` and `upratio[]`.

For a given j = indices[i], downratio[i] has the following meaning. Let *xj* be the name of the basic variable with index *j* and that the value of *xj* is *t*, and suppose that *xj* is fixed to some value *t' < t*. In a subsequent call to `CPXdualopt`, the leaving variable in the first iteration of this call is uniquely determined: It must be *xj*.

There are then two possibilities. Either an entering variable is determined, or it is concluded (in the first iteration) that the changed problem is dual unbounded (primal infeasible). In the latter case, `downratio[i]` is set equal to a large positive value (this number is system dependent, but is usually *1.0E+75*). In the former case, where *r* is the change in the objective function after this one iteration, `downratio[i]` is determined by */r/ = /t - t'/ * downratio[i]*.

The meaning of upratio[i] is analogous to that of downratio[i] except that *xj* is fixed to a value *t' > t*.

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by`CPXcreateprob`.
indices	An array of integers that must be of length at least `cnt`. The entries in `indices[]` must all be indices of current basic variables. Moreover, these indices must all be indices of original problem variables; that is, they must all take values smaller than the number of columns in the problem as returned by `CPXgetnumcols`. Negative indices and indices bigger than or equal to `CPXgetnumcols` result in an error.
cnt	An integer specifying the number of entries in `indices[]`. If `cnt` < 0, an error is returned.
downratio	An array of type `double` that must be of length at least `cnt`.
upratio	An array of type `double` that must be of length at least `cnt`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXreadcopyorder

```
int CPXreadcopyorder(CPXCENVptr env, CPXLPptr lp, const char * filename_str)
```

Definition file: cplex.h

The routine `CPXreadcopyorder` reads an ORD file and copies the priority order information into a CPLEX problem object. The parameter `CPX_PARAM_MIPORDIND` must be set to `CPX_ON` (its default value), in order for the priority order to be used for starting a subsequent optimization.

Example

```
status = CPXreadcopyorder (env, lp, "myprob.ord");
```

See Also: CPXordwrite

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	The name of the file from which the priority order should be read.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcopyobjname

`int` **`CPXcopyobjname`**`(CPXCENVptr env, CPXLPptr lp, const char * objname_str)`

Definition file: cplex.h

The routine `CPXcopyobjname` copies a name for the objective function into a CPLEX problem object. An argument to `CPXcopyobjname` defines the objective name.

Example

```
status = CPXcopyobjname (env, lp, "Cost");
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
objname_str	A pointer to a character string containing the objective name.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsolnpoolmipstart

```
int CPXgetsolnpoolmipstart(CPXCENVptr env, CPXCLPptr lp, int soln, int * cnt_p, int
* indices, double * value, int mipstartspace, int * surplus_p)
```

Definition file: cplex.h

This routine is **deprecated**. Use `CPXgetmipstarts` instead.

The routine `CPXgetsolnpoolmipstart` accesses MIP start information stored in the solution pool of a CPLEX problem object. Values are returned for all integer, binary, semi-continuous, and nonzero SOS variables.

Note

If the value of `mipstartspace` is 0 (zero), then the negative of the value of `*surplus_p` returned specifies the length needed for the arrays `indices` and `values`.

Example

```
status = CPXgetsolnpoolmipstart (env, lp, 5, &listsize, indices, values,
                                 numcols, &surplus);
```

See Also: CPXgetmipstarts

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
soln	An integer specifying the index of the solution pool member for which to return the MIP start. A value of -1 specifies that the current MIP start should be used instead of a solution pool member.
cnt_p	A pointer to an integer to contain the number of MIP start entries returned; that is, the true length of the arrays `indices` and `values`.
indices	An array to contain the indices of the variables in the MIP start. `indices[k]` is the index of the variable which is entry `k` in the MIP start information. Must be of length no less than `mipstartspace`.
value	An array to contain the MIP start values. The start value corresponding to `indices[k]` is returned in `values[k]`. Must be of length at least `mipstartspace`.
mipstartspace	An integer stating the length of the non-NULL array `indices` and `values`; `mipstartspace` may be 0 (zero).
surplus_p	A pointer to an integer to contain the difference between `mipstartspace` and the number of entries in each of the arrays `indices`, and `values`. A nonnegative value of `*surplus_p` specifies that the length of the arrays was sufficient. A negative value specifies that the length was insufficient and that the routine could not complete its task. In this case, the routine `CPXgetmipstart` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of `*surplus_p` specifies the amount of insufficient space in the arrays. The error `CPXERR_NO_MIPSTART` reports that no start information is available.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` reports that insufficient space was available in the arrays `indices` and `values` to hold the MIP start information.

Global function CPXdualfarkas

```
int CPXdualfarkas(CPXCENVptr env, CPXCLPptr lp, double * y, double * proof_p)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXdualfarkas` assumes that there is a resident solution as produced by a call to `CPXdualopt` and that the status of this solution as returned by `CPXgetstat` is `CPX_STAT_INFEASIBLE`.

The values returned in the array *y*[] have the following interpretation. For the *ith* constraint, if that constraint is a less-than-or-equal-to constraint, *y[i] <= 0* holds; if that constraint is a greater-than-or-equal-to constraint, *y[i] >= 0* holds. Thus, where `b` is the righthand-side vector for the given linear program, *A* is the constraint matrix, and *x* denotes the vector of variables, *y* may be used to derive the following valid inequality:

yTA x >= yTb

Here *y* is being interpreted as a column vector, and *yT* denotes the transpose of *y*.

The real point of computing *y* is the following. Suppose we define a vector *z* of dimension equal to the dimension of *x* and having the following value for entries

zj = uj where *yTAj > 0*, and

zj = lj where *yTAj < 0*,

where *Aj* denotes the column of *A* corresponding to *xj*, *uj* the given upper bound on *xj*, and *lj* is the specified lower bound. (*zj* is arbitrary if *yTAj = 0*.) Then *y* and *z* will satisfy

yTb - yTA z > 0.

This last inequality contradicts the validity of *yTA x >= yTb*, and hence shows that the given linear program is infeasible. The quantity `*proof_p` is set equal to *yTb - yTA z*. Thus, `*proof_p` in some sense denotes the degree of infeasibility.

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
y	An array of doubles of length at least equal to the number of rows in the problem.
proof_p	A pointer to a double. The argument `proof_p` is allowed to have the value NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetcolinfeas

```
int CPXgetcolinfeas(CPXCENVptr env, CPXCLPptr lp, const double * x, double *
infeasout, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetcolinfeas` computes the infeasibility of a given solution for a range of variables. The beginning and end of the range must be specified. This routine checks whether each variable takes a value within its bounds, but it does not check for integer feasibility in the case of integer variables. For each variable, the infeasibility value returned is 0 (zero) if the variable bounds are satisfied. Otherwise, if the infeasibility value is negative, it specifies the amount by which the lower bound (or semi-continuous lower bound in case of a semi-continuous or semi-integer variable) of the variable must be changed to make the queried solution valid. If the infeasibility value is positive, it specifies the amount by which the upper bound of the variable must be changed.

Example

```
status = CPXgetcolinfeas (env, lp, NULL, infeasout, 0, CPXgetnumcols(env,lp)-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
x	The solution whose infeasibility is to be computed. May be NULL, in which case the resident solution is used.
infeasout	An array to receive the infeasibility value for each of the variables. This array must be of length at least (`end - begin + 1`).
begin	An integer specifying the beginning of the range of variables whose infeasibility is to be returned.
end	An integer specifying the end of the range of variables whose infeasibility is to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXdelcols

```
int CPXdelcols(CPXCENVptr env, CPXLPptr lp, int begin, int end)
```

Definition file: cplex.h

The routine `CPXdelcols` deletes all the columns in a specified range. The range is specified using a lower and an upper index that represent the first and last column to be deleted, respectively. The indices of the columns following those deleted are decreased by the number of columns deleted.

Example

```
status = CPXdelcols (env, lp, 10, 20);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
begin	An integer that specifies the numeric index of the first column to be deleted.
end	An integer that specifies the numeric index of the last column to be deleted.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXslackfromx

```
int CPXslackfromx(CPXCENVptr env, CPXCLPptr lp, const double * x, double * slack)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXslackfromx` computes an array of slack values from primal solution values.

Example

```
status = CPXslackfromx (env, lp, x, slack);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
x	An array that contains primal solution (`x`) values for the problem, as returned by routines such as `CPXcrushx` and `CPXuncrushx`. The array must be of length at least the number of columns in the LP problem object.
slack	An array to receive the slack values computed from the x values for the problem object. The array must be of length at least the number of rows in the LP problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcheckcopyquad

`int` **`CPXcheckcopyquad`**`(CPXCENVptr env, CPXCLPptr lp, const int * qmatbeg, const int * qmatcnt, const int * qmatind, const double * qmatval)`

Definition file: cplex.h

The routine `CPXcheckcopyquad` validates the arguments of the corresponding routine `CPXcopyquad`. This data checking routine is found in source format in the file `check.c` provided with the standard CPLEX distribution. To call this routine, you must compile and link `check.c` with your program as well as the CPLEX Callable Library.

The `CPXcheckcopyquad` routine has the same argument list as the `CPXcopyquad` routine. The second argument, `lp`, is technically a pointer to a constant LP object of type `CPXCLPptr` rather than type `CPXLPptr`, as this routine will not modify the model. For most user applications, this distinction is unimportant.

Example

```
status = CPXcheckcopyquad (env, lp, qmatbeg, qmatcnt,
                           qmatind, qmatval);
```

Returns:

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.

Global function CPXbinvacol

```
int CPXbinvacol(CPXCENVptr env, CPXCLPptr lp, int j, double * x)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXbinvacol` computes the representation of the j-th column in terms of the basis. In other words, it solves Bx = Aj.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
j	An integer that specifies the index of the column to be computed.
x	An array containing the solution of Bx = Aj. The array must be of length at least equal to the number of rows in the problem.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcheckchgcoeflist

`int` **`CPXcheckchgcoeflist`**`(CPXCENVptr env, CPXCLPptr lp, int numcoefs, const int * rowlist, const int * collist, const double * vallist)`

Definition file: cplex.h

The routine `CPXcheckchgcoeflist` validates the arguments of the corresponding `CPXchgcoeflist` routine. This data checking routine is found in source format in the file `check.c` which is provided with the standard CPLEX distribution. To call this routine, you must compile and link `check.c` with your program as well as the CPLEX Callable Library.

The `CPXcheckchgcoeflist` routine has the same argument list as the `CPXchgcoeflist` routine. The second argument, `lp`, is technically a pointer to a constant LP object of type `CPXCLPptr` rather than type `CPXLPptr`, as this routine will not modify the problem. For most user applications, this distinction is unimportant.

Example

```
status = CPXcheckchgcoeflist (env, lp, numcoefs, rowlist,
                              collist, vallist);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
numcoefs	The number of coefficients to check, or, equivalently, the length of the arrays `rowlist`, `collist`, and `vallist`.
rowlist	An array of length `numcoefs` that with `collist` and `vallist` specifies the coefficients to check.
collist	An array of length `numcoefs` that with `rowlist` and `vallist` specifies the coefficients to check.
vallist	An array of length `numcoefs` that with `rowlist` and `collist` specifies the coefficients to change. The entries `rowlist[k]`, `collist[k]`, and `vallist[k]` specify that the matrix coefficient in row `rowlist[k]` and column `collist[k]` should be checked with respect to the value `vallist[k]`.

Returns:

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.

Global function CPXgetcallbacknodeub

```
int CPXgetcallbacknodeub(CPXCENVptr env, void * cbdata, int wherefrom, double * ub,
int begin, int end)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbacknodeub` retrieves the upper bound values for the subproblem at the current node during MIP optimization from within a user-written callback. The upper bounds are tightened after a new incumbent is found, so the values returned by `CPXgetcallbacknodex` may violate these bounds at nodes where new incumbents have been found. The values are from the original problem if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`; otherwise, they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

Example

```
status = CPXgetcallbacknodeub (env, cbdata, wherefrom,
                               ub, 0, cols-1);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting from where the user-written callback was called. The argument must be the value of `wherefrom` passed to the user-written callback.
ub	An array to receive the values of the upper bound values. This array must be of length at least (`end - begin + 1`). If successful, `ub[0]` through `ub[end-begin]` contain the upper bound values for the current subproblem.
begin	An integer specifying the beginning of the range of upper bound values to be returned.
end	An integer specifying the end of the range of upper bound values to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXnewcols

```
int CPXnewcols(CPXCENVptr env, CPXLPptr lp, int ccnt, const double * obj, const
double * lb, const double * ub, const char * xctype, char ** colname)
```

Definition file: cplex.h

The routine `CPXnewcols` adds empty columns to a specified CPLEX problem object. This routine may be called any time after a call to `CPXcreateprob`.

For each column, the user can specify the objective coefficient, the lower and upper bounds, the variable type, and name of the variable. The added columns are indexed to put them at the end of the problem. Thus, if `ccnt` columns are added to a problem object already having k columns, the new columns have indices `k, k+1`, ... `k+ccnt-1`. The constraint coefficients in the new columns are zero; the constraint coefficients can be changed with calls to `CPXchgcoef`, `CPXchgcoeflist`, or `CPXaddrows`.

The routine `CPXnewcols` is very similar to the routine `CPXnewrows`. It can be used to add variables to a problem object without specifying the matrix coefficients.

Types of new variables: values of ctype[j]

`CPX_CONTINUOUS`	`'C'`	continuous variable `j`
`CPX_BINARY`	`'B'`	binary variable `j`
`CPX_INTEGER`	`'I'`	general integer variable `j`
`CPX_SEMICONT`	`'S'`	semi-continuous variable `j`
`CPX_SEMIINT`	`'N'`	semi-integer variable `j`

Example

```
status = CPXnewcols (env, lp, ccnt, obj, lb, ub, NULL, NULL);
```

See also the example `lpex8.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
ccnt	An integer that specifies the number of new variables being added to the problem object.
obj	An array of length `ccnt` containing the objective function coefficients of the new variables. This array may be NULL, in which case the new objective coefficients are all set to 0 (zero).
lb	An array of length `ccnt` containing the lower bound on each of the new variables. Any lower bound that is set to a value less than or equal to that of the constant `-CPX_INFBOUND` is treated as negative infinity. `CPX_INFBOUND` is defined in the header file `cplex.h`. This array may be NULL, in which case the new lower bounds are all set to 0 (zero).
ub	An array of length `ccnt` containing the upper bound on each of the new variables. Any upper bound that is set to a value greater than or equal to that of the constant `CPX_INFBOUND` is treated as infinity. `CPX_INFBOUND` is defined in the header file `cplex.h`. This array may be NULL, in which case the new upper bounds are all set to `CPX_INFBOUND`.
xctype	An array of length `ccnt` containing the type of each of the new variables. Possible values appear in the table. This array may be NULL, in which case the new variables are created as continuous type. If this array is not NULL, then CPLEX interprets the problem as a MIP; in that case, the routine `CPXlpopt` will return the error `CPXERR_NOT_FOR_MIP`.

colname	An array of length `ccnt` containing pointers to character strings that specify the names of the new variables added to the problem object. May be NULL, in which case the new columns are assigned default names if the columns already resident in the problem object have names; otherwise, no names are associated with the variables. If column names are passed to `CPXnewcols` but existing variables have no names assigned, default names are created for the existing variables.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXprechgobj

`int` **`CPXprechgobj`**`(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices, const double * values)`

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXprechgobj` changes the objective function coefficients of an LP problem object and its associated presolved LP problem object. The CPLEX parameter `CPX_PARAM_REDUCE` must be set to `CPX_PREREDUCE_PRIMALONLY (1)` or `CPX_PREREDUCE_NOPRIMALORDUAL (0)` at the time of the presolve in order to change objective coefficients and preserve the presolved problem. This routine should be used in place of `CPXchgobj` when it is desired to preserve the presolved problem.

The arguments and operation of `CPXprechgobj` are the same as those of `CPXchgobj`. The objective coefficient changes are applied to both the original LP problem object and the associated presolved LP problem object.

Example

```
status = CPXprechgobj (env, lp, objcnt, objind, objval);
```

See also the example `adpreex1.c` in the standard distribution.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgettime

```
int CPXgettime(CPXCENVptr env, double * timestamp)
```

Definition file: cplex.h

This routine returns a time stamp.

To measure time spent between a starting point and ending point of an operation, take the result of this routine at the starting point; take the result of this routine at the end point; subtract the starting time stamp from the ending time stamp; the subtraction yields elapsed time in seconds.

Whether the elapsed time measures wall clock time (also known as real time) or CPU time depends on the setting of the clock type parameter `CPX_PARAM_CLOCKTYPE`.

The absolute value of the time stamp is not meaningful.

Global function CPXNETgetnumarcs

`int` **`CPXNETgetnumarcs`**`(CPXCENVptr env, CPXCNETptr net)`

Definition file: cplex.h

The routine `CPXNETgetnumarcs` is used to access the number of arcs in a network stored in a network problem object.

Example

```
cur_narcs = CPXNETgetnumarcs (env, net);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

net A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

Returns:

The routine returns the number of network arcs stored in a network problem object. If an error occurs, 0 is returned and an error message is issued.

Global function CPXgetnumindconstrs

`int` **`CPXgetnumindconstrs`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetnumindconstrs` accesses the number of indicator constraints in a CPLEX problem object.

Example

```
cur_numindconstrs = CPXgetnumindconstrs (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If the problem object or environment does not exist, `CPXgetnumindconstrs` returns the value 0 (zero); otherwise, it returns the number of indicator constraints.

Global function CPXcopyorder

```
int CPXcopyorder(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices, const
int * priority, const int * direction)
```

Definition file: cplex.h

The routine `CPXcopyorder` copies a priority order to a CPLEX problem object of type `CPXPROB_MILP`, `CPXPROB_MIQP`, or `CPXPROB_MIQCP`. A call to `CPXcopyorder` replaces any other information about priority order previously stored in that CPLEX problem object. During branching, integer variables with higher priorities are given preference over integer variables with lower priorities. Priorities must be nonnegative integers. A preferred branching direction may also be specified for each variable.

The CPLEX parameter `CPX_PARAM_MIPORDIND` must be set to `CPX_ON`, its default value, for the priority order to be used in a subsequent optimization.

Table 1: Settings for direction

`CPX_BRANCH_GLOBAL`	use global branching direction when setting the parameter `CPX_PARAM_BRDIR`
`CPX_BRANCH_DOWN`	branch down first on variable `indices[i]`
`CPX_BRANCH_UP`	branch up first on variable `indices[i]`

Example

```
status = CPXcopyorder (env, lp, cnt, indices, priority,
                       direction);
```

See Also: CPXreadcopyorder

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt	An integer giving the number of entries in the list.
indices	An array of length `cnt` containing the numeric indices of the columns corresponding to the integer variables that are assigned priorities.
priority	An array of length `cnt` containing the priorities assigned to the integer variables. The entry `priority[j]` is the priority assigned to variable `indices[j]`. May be NULL.
direction	An array of type `int` containing the branching direction assigned to the integer variables. The entry `direction[j]` is the direction assigned to variable `indices[j]`. May be NULL. Possible settings for `direction[j]` appear in Table 1.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETsolninfo

`int` **`CPXNETsolninfo`**`(CPXCENVptr env, CPXCNETptr net, int * pfeasind_p, int * dfeasind_p)`

Definition file: cplex.h

The routine `CPXNETsolninfo` is used to access solution information computed by the most recent call to `CPXNETprimopt`. The solution values are maintained in the object as long as no changes are applied to it with one of the routines `CPXNETchg...`, `CPXNETcopy...`, or `CPXNETadd...`.

The arguments to `CPXNETsolninfo` are pointers to locations where data are to be written. The returned values indicate what is known about the primal and dual feasibility of the current solution. If either piece of information represented by an argument to `CPXNETsolninfo` is not required, a NULL pointer can be passed for that argument.

Example

```
status = CPXNETsolninfo (env, lp, &pfeasind, &dfeasind);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
pfeasind_p	A pointer to an integer variables indicating whether the current solution is known to be primal feasible. Note that a false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude that it was feasible when it terminated.
dfeasind_p	A pointer to an integer variables indicating whether the current solution is known to be dual feasible. Note that a false return value does not necessarily mean that the solution is not feasible. It simply means that the relevant algorithm was not able to conclude that it was feasible when it terminated.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetsolnpooldivfilter

```
int CPXgetsolnpooldivfilter(CPXCENVptr env, CPXCLPptr lp, double * lowercutoff_p,
double * upper_cutoff_p, int * nzcnt_p, int * ind, double * val, double * refval,
int space, int * surplus_p, int which)
```

Definition file: cplex.h

Accesses a diversity filter of the solution pool.

This routine accesses a diversity filter, specified by the argument `which`, of the solution pool associated with the problem specified by the argument `lp`. Details about that filter are returned in the arguments of this routine.

Example

```
status = CPXgetsolnpooldivfilter (env, lp, &limlo, &limup,
                                  &num, ind, val, refval,
                                  cols, &surplus, i);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lowercutoff_p	Lower bound on the diversity measure of a diversity filter.
upper_cutoff_p	Upper bound on the diversity measure of a diversity filter.
nzcnt_p	Number of variables in the diversity measure.
ind	An array of indices of variables in the diversity measure. May be NULL if `space>` is 0.
val	An array of weights used in the diversity measure. May be NULL if `space>` is 0.
refval	List of reference values with which to compare the solution. May be NULL if `space>` is 0.
space	Integer specifying the length of the arrays `ind`,`val`, and `refval` (if `refval` is not NULL.
surplus_p	A pointer to an integer to contain the difference between `space` and the number of entries in each of the arrays `ind` and `val`. A nonnegative value of `surplus_p` means that the length of the arrays was sufficient. A negative value reports that the length was insufficient and consequently the routine could not complete its task. In this case, the routine `CPXgetsolnpooldivfilter` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of `surplus_p` specifies the amount of insufficient space in the arrays.
which	An integer specifying the index of the filter to access.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXaddcols

```
int CPXaddcols(CPXCENVptr env, CPXLPptr lp, int ccnt, int nzcnt, const double *
obj, const int * cmatbeg, const int * cmatind, const double * cmatval, const double
* lb, const double * ub, char ** colname)
```

Definition file: cplex.h

The routine CPXaddcols adds columns to a specified CPLEX problem object. This routine may be called any time after a problem object is created via CPXcreateprob.

The routine CPXaddcols is very similar to the routine CPXaddrows. The primary difference is that CPXaddcols cannot add coefficients in rows that do not already exist (that is, in rows with index greater than the number returned by CPXgetnumrows); whereas CPXaddrows can add coefficients in columns with index greater than the value returned by CPXgetnumcols, by the use of the ccnt argument. (See the discussion of the ccnt argument for CPXaddrows.) Thus, CPXaddcols has no variable rcnt and no array rowname.

The routine CPXnewrows can be used to add empty rows before adding new columns via CPXaddcols.

The nonzero elements of every column must be stored in sequential locations in the array cmatval from position cmatbeg[i] to cmatbeg[i+1] (or from cmatbeg[i] to nzcnt-1 if i=ccnt-1). Each entry, cmatind[i], specifies the row number of the corresponding coefficient, cmatval[i]. Unlike CPXcopylp, all columns must be contiguous, and cmatbeg[0] must be 0.

When you build or modify your problem with this routine, you can verify that the results are as you intended by calling CPXcheckaddcols during application development.

Example

```
status = CPXaddcols (env, lp, ccnt, nzcnt, obj, cmatbeg,
                     cmatind, cmatval, lb, ub, newcolname);
```

Parameters:

env	A pointer to the CPLEX environment as returned by the CPXopenCPLEX routine.
lp	A pointer to a CPLEX problem object as returned by CPXcreateprob.
ccnt	An integer that specifies the number of new columns being added to the constraint matrix.
nzcnt	An integer that specifies the number of nonzero constraint coefficients to be added to the constraint matrix.
obj	An array of length ccnt containing the objective function coefficients of the new variables. May be NULL, in which case, the objective coefficients of the new columns are set to 0.0.
cmatbeg	Array that specifies the nonzero elements of the columns being added.
cmatind	Array that specifies the nonzero elements of the columns being added.
cmatval	Array that specifies the nonzero elements of the columns being added. The format is similar to the format used to specify the constraint matrix in the routine CPXcopylp. (See description of matbeg, matcnt, matind, and matval in that routine).
lb	An array of length ccnt containing the lower bound on each of the new variables. Any lower bound that is set to a value less than or equal to that of the constant -CPX_INFBOUND is treated as negative infinity. CPX_INFBOUND is defined in the header file cplex.h. May be NULL, in which case the lower bounds of the new columns are set to 0.0.
ub	An array of length ccnt containing the upper bound on each of the new variables. Any upper bound that is set to a value greater than or equal to that of the constant CPX_INFBOUND is treated as infinity. CPX_INFBOUND is defined in the header file cplex.h. May be NULL, in which case the upper bounds of the new columns are set to CPX_INFBOUND (positive infinity).
colname	An array of length ccnt containing pointers to character strings that specify the names of the new variables added to the problem object. May be NULL, in which case the new columns are assigned default names if the columns already resident in the CPLEX problem object

have names; otherwise, no names are associated with the variables. If column names are passed to `CPXaddcols` but existing variables have no names assigned, default names are created for them.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXkilldnorms

`void` **`CPXkilldnorms`**`(CPXLPptr lp)`

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXkilldnorms` deletes any dual steepest-edge norms that have been retained relative to an active basis. If the user believes that the values of these norms may be significantly in error, and the setting of the parameter `CPX_PARAM_DPRIIND` is `CPX_DPRIIND_STEEP` or `CPX_DPRIIND_FULLSTEEP`, calling `CPXkilldnorms` means that fresh dual steepest-edge norms will be computed on the next call to `CPXdualopt`.

Parameters:

lp The pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.

Global function CPXdelsetsolnpoolsolns

`int` **`CPXdelsetsolnpoolsolns`**`(CPXCENVptr env, CPXLPptr lp, int * delstat)`

Definition file: cplex.h

The routine `CPXdelsetsolnpoolsolns` deletes solutions from the solution pool of the problem object specified by the argument `lp`. Unlike the routine `CPXdelsolnpoolsolns`, `CPXdelsetsolnpoolsolns` does not require the solutions to be in a contiguous range. After the deletion occurs, the remaining solutions are indexed consecutively starting at 0 (zero), and in the same order as before the deletion.

Note

The `delstat` array must have at least `CPXgetsolnpoolnumsolns(env,lp)` elements.

Example

```
status = CPXdelsetsolnpoolsolns (env, lp, delstat);
```

See Also: CPXdelsolnpoolsolns

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
delstat	An array specifying the solutions to be deleted. The routine `CPXdelsetsolnpoolsolns` deletes each solution `i` for which `delstat[i] = 1`. The deletion of solutions results in a renumbering of the remaining solutions. After termination, `delstat[i]` is either -1 for filters that have been deleted or the new index number that has been assigned to the remaining solutions.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETgetnodename

```
int CPXNETgetnodename(CPXCENVptr env, CPXCNETptr net, char ** nnames, char *
namestore, int namespc, int * surplus_p, int begin, int end)
```

Definition file: cplex.h

The routine `CPXNETgetnodename` is used to obtain the names of a range of nodes in a network stored in a network problem object. The beginning and end of the range, along with the length of the array in which the node names are to be returned, must be specified.

Example

```
status = CPXNETgetnodename (env, net, nnames, namestore, namespc,
                            &surplus, 0, nnodes-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
nnames	Where to copy pointers to node names stored in the `namestore` array. The length of this array must be at least (`end-begin+1`). The pointer to the name of node `i` is returned in `nnames[i-begin]`.
namestore	Array of characters to which the specified node names are to be copied. It may be NULL if `namespc` is 0.
namespc	Length of the `namestore` array.
surplus_p	Pointer to an integer in which the difference between `namespc` and the number of characters required to store the requested names is returned. A nonnegative value indicates that `namespc` was sufficient. A negative value indicates that it was insufficient. In that case, `CPXERR_NEGATIVE_SURPLUS` is returned and the negative value of `surplus_p` indicates the amount of insufficient space in the array `namestore`.
begin	Index of the first node for which a name is to be obtained.
end	Index of the last node for which a name is to be obtained.

Returns:

The routine returns zero on success and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` indicates that there was not enough space in the `namestore` array to hold the names.

Global function CPXsetheuristiccallbackfunc

```
int CPXsetheuristiccallbackfunc(CPXENVptr env, int(CPXPUBLIC
*heuristiccallback)(CALLBACK_HEURISTIC_ARGS), void * cbhandle)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXsetheuristiccallbackfunc` sets or modifies the user-written callback to be called by CPLEX during MIP optimization after the subproblem has been solved to optimality. That callback is not called when the subproblem is infeasible or cut off. The callback supplies CPLEX with heuristically-derived integer solutions.

If a linear program must be solved as part of a heuristic callback, make a copy of the node LP and solve the copy, not the CPLEX node LP.

Example

```
status = CPXsetheuristiccallbackfunc(env, myheuristicfunc, mydata);
```

See also the example `admipex2.c` in the standard distribution.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`heuristiccallback`

A pointer to a user-written heuristic callback. If this callback is set to NULL, no callback is called during optimization.

`cbhandle`

A pointer to the user's private data. This pointer is passed to the callback.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle,
              double     *objval_p,
              double     *x,
              int        *checkfeas_p,
              int        *useraction_p);
```

The call to the heuristic callback occurs after an optimal solution to the subproblem has been obtained. The user can provide that solution to start a heuristic for finding an integer solution. The integer solution provided to CPLEX replaces the incumbent if it has a better objective value. The basis that is saved as part of the incumbent is the optimal basis from the subproblem; it may not be a good basis for starting optimization of the fixed problem.

The integer solution returned to CPLEX is for the original problem if the parameter CPX_PARAM_MIPCBREDLP was set to CPX_OFF before the call to CPXmipopt that calls the callback. Otherwise, it is for the presolved problem.

Callback return value

The callback returns zero if successful and nonzero if an error occurs.

Callback arguments

env

A pointer to the CPLEX environment, as returned by CPXopenCPLEX.

cbdata

A pointer passed from the optimization routine to the user-written callback to identify the problem being optimized. The only purpose of the cbdata pointer is to pass it to the callback information routines.

wherefrom

An integer value reporting at which point in the optimization this function was called. It has the value CPX_CALLBACK_MIP_HEURISTIC for the heuristic callback.

cbhandle

A pointer to user private data.

objval_p

A pointer to a variable that on entry contains the optimal objective value of the subproblem and on return contains the objective value of the integer solution found, if any.

x

An array that on entry contains primal solution values for the subproblem and on return contains solution values for the integer solution found, if any. The values are from the original problem if the parameter CPX_PARAM_MIPCBREDLP is turned off (that is, set to CPX_OFF); otherwise, the values are from the presolved problem.

checkfeas_p

A pointer to an integer that specifies whether or not CPLEX should check the returned integer solution for integer feasibility. The solution is checked if checkfeas_p is nonzero. When the solution is checked and found to be integer infeasible, it is discarded, and optimization continues.

useraction_p

A pointer to an integer to contain the specifier of the action to be taken on completion of the user callback. The table summarizes the possible values.

Actions to be Taken after a User-Written Heuristic Callback

Value	Symbolic Constant	Action
0	CPX_CALLBACK_DEFAULT	No solution found
1	CPX_CALLBACK_FAIL	Exit optimization
2	CPX_CALLBACK_SET	Use user solution as reported in return values

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcutcallbackaddlocal

```
int CPXcutcallbackaddlocal(CPXCENVptr env, void * cbdata, int wherefrom, int nzcnt,
double rhs, int sense, const int * cutind, const double * cutval)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXcutcallbackaddlocal` adds local cuts during MIP branch and cut. A local cut is one that applies to the current node and the subtree rooted at this node. Global cuts, that is, cuts that apply throughout the branch-and-cut tree, are added with the routine `CPXcutcallbackadd`. This routine may be called only from within user-written cut callbacks; thus it may be called only when the value of its `wherefrom` argument is `CPX_CALLBACK_MIP_CUT`.

The cut may be for the original problem if the parameter `CPX_PARAM_MIPCBREDLP` was set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, the cut is used on the presolved problem.

Example

```
status = CPXcutcallbackaddlocal (env,
                                 cbdata,
                                 wherefrom,
                                 mynzcnt,
                                 myrhs,
                                 'L',
                                 mycutind,
                                 mycutval);
```

See Also: CPXcutcallbackadd, CPXgetcutcallbackfunc, CPXsetcutcallbackfunc

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value that reports where the user-written callback was called from. This argument must be the value of `wherefrom` passed to the user-written callback.
nzcnt	An integer value that specifies the number of coefficients in the cut, or equivalently, the length of the arrays `cutind` and `cutval`.
rhs	A double value that specifies the value of the righthand side of the cut.
sense	An integer value that specifies the sense of the cut.
cutind	An array containing the column indices of cut coefficients.
cutval	An array containing the values of cut coefficients.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetnumsos

`int` **`CPXgetnumsos`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetnumsos` accesses the number of special ordered sets (SOS) in a CPLEX problem object.

Example

```
numsos = CPXgetnumsos (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Example

```
numsos = CPXgetnumsos (env, lp);
```

Returns:

If the problem object or environment does not exist, or the problem is not a mixed integer problem, the routine returns the value 0; otherwise, it returns the number of special ordered sets (SOS) in the problem object.

Global function CPXgetcallbackctype

```
int CPXgetcallbackctype(CPXCENVptr env, void * cbdata, int wherefrom, char *
xctype, int begin, int end)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbackctype` retrieves the ctypes for the MIP problem from within a user-written callback during MIP optimization. The values are from the original problem if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`. Otherwise, they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

Example

```
status = CPXgetcallbackctype (env, cbdata, wherefrom,
                              prectype, 0, precols-1);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting from where the user-written callback was called. The argument must be the value of `wherefrom` passed to the user-written callback.
xctype	An array where the ctype values for the MIP problem will be returned. The array must be of length at least (`end - begin + 1`). If successful, `xctype[0]` through `xctype[end-begin]` contain the variable types.
begin	An integer specifying the beginning of the range of ctype values to be returned.
end	An integer specifying the end of the range of ctype values to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsolution

```
int CPXsolution(CPXCENVptr env, CPXCLPptr lp, int * lpstat_p, double * objval_p,
double * x, double * pi, double * slack, double * dj)
```

Definition file: cplex.h

The routine `CPXsolution` accesses the solution values produced by all the optimization routines **except** the routine `CPXNETprimopt`. The solution is maintained until the CPLEX problem object is freed via a call to `CPXfreeprob` or the solution is rendered invalid because of a call to one of the problem modification routines.

The arguments to `CPXsolution` are pointers to locations where data are to be written. Such data can include the status of the optimization, the value of the objective function, the values of the primal variables, the dual variables, the slacks and the reduced costs. All of that data exists after a successful call to one of the LP or QP optimizers. However, dual variables and reduced costs are **not** available after a successful call of the QCP or MIP optimizers. If any part of the solution represented by an argument to `CPXsolution` is not required, that argument can be passed with the value NULL in a call to `CPXsolution`. If only one part is required, it may be more convenient to use the CPLEX routine that accesses that part of the solution individually: `CPXgetstat`, `CPXgetobjval`, `CPXgetx`, `CPXgetpi`, `CPXgetslack`, or `CPXgetdj`.

For barrier, the solution values for `x`, `pi`, `slack`, and `dj` correspond to the last iterate of the primal-dual algorithm, independent of solution status.

If optimization stopped with an infeasible solution, take care to interpret the meaning of the values in the returned arrays as described in the Parameters section.

Example

```
status = CPXsolution (env, lp, &lpstat, &objval, x, pi,
                      slack, dj);
```

See also the example `lpex1.c` in the *CPLEX User's Manual* and in the standard distribution.

See Also: CPXsolninfo

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lpstat_p	A pointer to an integer specifying the result of the optimization. The specific values which `*lpstat_p` can take and their meanings are the same as the return values documented for `CPXgetstat` and are found in the group optim.cplex.statuscodes of this reference manual.
objval_p	A pointer to a double precision variable where the objective function value is to be stored.
x	An array to receive the values of the variables for the problem. The length of the array must be at least as great as the number of columns in the problem object. If the solution was computed using the dual simplex optimizer, and the solution is not feasible, `x` values are calculated relative to the phase I RHS used by `CPXdualopt`.
pi	An array to receive the values of the dual variables for each of the constraints. The length of the array must be at least as great as the number of rows in the problem object. If the solution was computed using the primal simplex optimizer, and the solution is not feasible, `pi` values are calculated relative to the phase I objective (the infeasibility function).
slack	An array to receive the values of the slack or surplus variables for each of the constraints. The length of the array must be at least as great as the number of rows in the problem object. If the solution was computed by the dual simplex optimizer, and the solution is not feasible, `slack` values are calculated relative to the phase I RHS used by `CPXdualopt`.
dj	An array to receive the values of the reduced costs for each of the variables. The length of the array must be at least as great as the number of columns in the problem object. If the solution was computed by the primal simplex optimizer, and the solution is not feasible, `dj` values are calculated relative to the phase I objective (the infeasibility function).

Returns:

This routine returns zero if a solution exists. If no solution exists, or some other failure occurs, `CPXsolution` returns nonzero.

Global function CPXsetterminate

```
int CPXsetterminate(CPXENVptr env, volatile int * terminate_p)
```

Definition file: cplex.h

This routine enables applications to terminate CPLEX gracefully.

Conventionally, your application should first call this routine to set a pointer to the termination signal. Then the application can set the termination signal to a nonzero value to tell CPLEX to abort. These conventions will terminate CPLEX even in a different thread. In other words, this routine makes it possible to handle signals such as control-C from a user interface. These conventions also enable termination within CPLEX callbacks.

Example

```
status = CPXsetterminate (env, &terminate);
```

To unset a termination signal set by this routine, call this routine again later with a NULL pointer as the value of the argument `terminate_p`.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
terminate_p	A pointer to the termination signal.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetobjname

```
int CPXgetobjname(CPXCENVptr env, CPXCLPptr lp, char * buf_str, int bufspace, int *
surplus_p)
```

Definition file: cplex.h

The routine `CPXgetobjname` accesses the name of the objective row of a CPLEX problem object.

Note

If the value of `bufspace` is 0, then the negative of the value of `surplus_p` returned specifies the total number of characters needed for the array `buf_str`.

Example

```
status = CPXgetobjname (env, lp, cur_objname, lenname,
                        &surplus);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
buf_str	A pointer to a buffer of size `bufspace`. May be NULL if `bufspace` is 0.
bufspace	An integer specifying the length of the array `buf_str`. May be 0.
surplus_p	A pointer to an integer to contain the difference between `bufspace` and the amount of memory required to store the objective row name. A nonnegative value of `surplus_p` specifies that the length of the array `buf_str` was sufficient. A negative value specifies that the length of the array was insufficient and that the routine could not complete its task. In this case, `CPXgetobjname` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `surplus_p` specifies the amount of insufficient space in the array `buf_str`.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` specifies that insufficient space was available in the `buf_str` array to hold the objective name.

Global function CPXpopulate

`int` **`CPXpopulate`**`(CPXCENVptr env, CPXLPptr lp)`

Definition file: cplex.h

The routine `CPXpopulate` generates multiple solutions to a mixed integer programming (MIP) problem.

The algorithm that populates the solution pool works in two phases.

In the first phase, it solves the problem to optimality (or some stopping criterion set by the user) while it sets up a branch and cut tree for the second phase.

In the second phase, it generates multiple solutions by using the information computed and stored in the first phase and by continuing to explore the tree.

The amount of preparation in the first phase and the intensity of exploration in the second phase are controlled by the solution pool intensity parameter `CPX_PARAM_SOLNPOOLINTENSITY`.

Optimality is not a stopping criterion for the populate procedure. Even if the optimality gap is zero, this routine will still try to find alternative solutions. The **stopping criteria** for `CPXpopulate` are these:

- Populate limit `CPX_PARAM_POPULATELIM`. This parameter controls how many solutions are generated before stopping. Its default value is 20.
- Time limit `CPX_PARAM_TILIM`, as in standard MIP optimization.
- Node limit `CPX_PARAM_NODELIM`, as in standard MIP optimization.
- In the absence of other stopping criteria, `CPXpopulate` stops when it cannot enumerate any more solutions. In particular, if the user specifies an objective tolerance with the relative or absolute solution pool gap parameters, `CPXpopulate` stops if it cannot enumerate any more solutions within the specified objective tolerance. However, there may exist additional solutions that are feasible, and if the user has specified an objective tolerance, those feasible solutions may also satisfy this additional criterion. (For example, there may be a great many solutions to a given problem with the same integer values but different values for continuous variables.) Depending on the setting of the solution pool intensity parameter `CPX_PARAM_SOLNPOOLINTENSITY`, `CPXpopulate` may or may not enumerate all possible solutions. Consequently, `CPXpopulate` may stop when it has enumerated only a subset of the solutions satisfying your criteria.

Successive calls to `CPXpopulate` create solutions that are stored in the solution pool. However, each call to `CPXpopulate` applies only to the subset of solutions created in the current call; the call does not affect the solutions already in the pool. In other words, solutions in the pool are persistent.

The user may call this routine independently of any MIP optimization of a problem (such as `CPXmipopt`). In that case, `CPXpopulate` carries out the first and second phase itself.

The user may also call `CPXpopulate` after `CPXmipopt`. The activity of `CPXmipopt` constitutes the first phase of the populate algorithm; `CPXpopulate` then re-uses the information computed and stored by `CPXmipopt` and thus carries out only the second phase.

`CPXpopulate` does not try to generate multiple solutions for unbounded MIP problems. As soon as the proof of unboundedness is obtained, `CPXpopulate` stops.

Example

```
status = CPXpopulate (env, lp);
```

For more detail about populate, see also the chapter titled *Solution Pool: Generating and Keeping Multiple Solutions* in the *CPLEX User's Manual*.

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp A pointer to the CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetmipstart

```
int CPXgetmipstart(CPXCENVptr env, CPXCLPptr lp, int * cnt_p, int * indices, double
* value, int mipstartspace, int * surplus_p)
```

Definition file: cplex.h

This routine is **deprecated**. Use `CPXgetmipstarts` instead.

The routine `CPXgetmipstart` accesses information about the incumbent MIP start stored in a CPLEX problem object. Values are returned for all integer, binary, semi-continuous, and nonzero SOS variables.

> **Note**
>
> If the value of `mipstartspace` is 0 (zero), then the negative of the value of `surplus_p` returned specifies the length needed for the arrays `indices` and `values`.

Example

```
status = CPXgetmipstart (env, lp, &listsize, indices, values,
                         numcols, &surplus);
```

See Also: CPXgetmipstarts

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt_p	A pointer to an integer to contain the number of MIP start entries returned; that is, the true length of the arrays `indices` and `values`.
indices	An array to contain the indices of the variables in the incumbent MIP start. `indices[k]` is the index of the variable which is entry `k` in the MIP start information. Must be of length no less than `mipstartspace`.
value	An array to contain the incumbent MIP start values. The start value corresponding to `indices[k]` is returned in `values[k]`. Must be of length at least `mipstartspace`.
mipstartspace	An integer stating the length of the non-NULL array `indices` and `values`; `mipstartspace` may be 0 (zero).
surplus_p	A pointer to an integer to contain the difference between `mipstartspace` and the number of entries in each of the arrays `indices`, and `values`. A nonnegative value of `*surplus_p` specifies that the length of the arrays was sufficient. A negative value specifies that the length was insufficient and that the routine could not complete its task. In this case, the routine `CPXgetmipstart` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of `*surplus_p` specifies the amount of insufficient space in the arrays. The error `CPXERR_NO_MIPSTART` reports that no start information is available.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` reports that insufficient space was available in the arrays `indices` and `values` to hold the incumbent MIP start information.

Global function CPXNETcopybase

```
int CPXNETcopybase(CPXCENVptr env, CPXNETptr net, const int * astat, const int *
nstat)
```

Definition file: cplex.h

The routine `CPXNETcopybase` can be used to set the network basis for a network problem object. It is not necessary to load a basis prior to optimizing a problem, but a very good starting basis may increase the speed of optimization significantly. A copied basis does not need to be feasible to be used by the network optimizer.

Any solution information stored in the problem object is lost.

Example

```
status = CPXNETcopybase (env, net, arc_stat, node_stat);
```

Table 1: Status of arcs in astat

`CPX_BASIC`	if the arc is to be basic
`CPX_AT_LOWER`	if the arc is to be nonbasic and its flow is on the lower bound
`CPX_AT_UPPER`	if the arc is to be nonbasic and its flow is on the upper bound
`CPX_FREE_SUPER`	if the arc is to be nonbasic but is free. In this case its flow is set to 0

Table 2: Status of artificial arcs in nstat

`CPX_BASIC`	if the arc is to be basic
`CPX_AT_LOWER`	if the arc is to be nonbasic and its flow is set to 0

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
astat	Array of status values for network arcs. Each arc needs to be assigned one of the values in Table 1.
nstat	Array of status values for artificial arcs from each node to the root node. Each artificial arc needs to be assigned one of the values in Table 2. At least one of the artificial arcs must be assigned the status `CPX_BASIC` for a network basis.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXrefineconflictext

```
int CPXrefineconflictext(CPXCENVptr env, CPXLPptr lp, int grpcnt, int concnt, const
double * grppref, const int * grpbeg, const int * grpind, const char * grptype)
```

Definition file: cplex.h

The routine `CPXrefineconflictext` extends `CPXrefineconflict` to problems with indicator constraints, quadratic constraints, or special ordered sets (SOSs) and to situations where groups of constraints should be considered as a single constraint. The routine `CPXrefineconflictext` identifies a minimal conflict for the infeasibility of the current problem or a subset of constraints of the current problem. Since the conflict is minimal, removal of any group of constraints that is a member of the conflict will remove that particular source of infeasibility. However, there may be other conflicts in the problem; consequently, that repair of one conflict does not guarantee feasibility of the solution of the remaining problem.

Constraints are considered in groups in this routine. If any constraint in a group participates in the conflict, the entire group is determined to do so. No further detail about the constraints within that group is returned. A group may consist of a single constraint.

A group may be assigned a preference; that is, a value specifying how much the user wants the group to be part of a conflict. A group with a higher preference is more likely to be included in the conflict. However, no guarantee is made when a minimal conflict is returned that other conflicts containing groups with a greater preference do not exist.

To retrieve information about the conflict computed by `CPXrefineconflictext`, call the routine `CPXgetconflictext`.

Table 1: Possible values for elements of grptype

`CPX_CON_LOWER_BOUND`	1	variable lower bound
`CPX_CON_UPPER_BOUND`	2	variable upper bound
`CPX_CON_LINEAR`	3	linear constraint
`CPX_CON_QUADRATIC`	4	quadratic constraint
`CPX_CON_SOS`	5	special ordered set
`CPX_CON_INDICATOR`	6	indicator constraint

The parameters `CPX_PARAM_CUTUP`, `CPX_PARAM_CUTLO`, `CPX_PARAM_OBJULIM`, `CPX_PARAM_OBJLLIM` do not influence this routine. If you want to study infeasibilities introduced by those parameters, consider adding an objective function constraint to your model to enforce their effect before you invoke this routine.

Example

```
status = CPXrefineconflictext (env, lp, ngrp, ngrp, pri, beg, ind, type);
```

See Also: CPXgetconflictext, CPXrefineconflict, CPXclpwrite

Parameters:

env	A pointer to the CPLEX environment as returned by the routine `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
grpcnt	The number of constraint groups to be considered.
concnt	An integer specifying the total number of elements passed in the arrays `grpind` and `grptype`, or, equivalently, the end of the last group in `grpind`.
grppref	An array of preferences for the groups. The value `grppref[i]` specifies the preference for the group designated by the index `i`. A negative value specifies that the corresponding group should not be

considered in the computation of a conflict. In other words, such groups are not considered part of the problem. Groups with a preference of 0 (zero) are always considered to be part of the conflict. No further checking is performed on such groups.

grpbeg An array of integers specifying where the constraint indices for each group begin in the array `grpind`. Its length must be at least `grpcnt`.

grpind An array of integers containing the indices for the constraints in each group. For each of the various types of constraints listed in the table, the constraint indices range from 0 (zero) to the number of constraints of that type minus one. Group `i` contains the constraints with the indices `grpind[grpbeg[i]], ..., grpind[grpbeg[i+1]-1]` for `i` less than `grpcnt-1`, and `grpind[grpbeg[i]], ..., grpind[concnt-1]` for `i == grpcnt-1`. Its length must be at least `concnt`. A constraint must not be referenced more than once in this array. For any constraint in the problem that is not a member of a group and thus does not appear in this array, the constraint is assigned a default preference of 0 (zero). Thus such constraints are included in the conflict without any analysis.

grptype An array of characters containing the constraint types for the constraints as they appear in groups. The types of the constraints in group `i` are specified in `grptype[grpbeg[i]], ..., grptype[grpbeg[i+1]-1]` for `i` less than `grpcnt-1` and `grptype[grpbeg[i]], ..., grptype[concnt-1]` for `i == grpcnt-1`. Its length must be at least `concnt`, and every constraint must appear at most once in this array. Possible values appear in Table 1.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetbhead

```
int CPXgetbhead(CPXCENVptr env, CPXCLPptr lp, int * head, double * x)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetbhead` returns the basis header; it gives the negative value minus one of all row indices of slacks.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
head	An array. The array contains the indices of the variables in the resident basis, where basic slacks are specified by the negative of the corresponding row index minus 1 (one); that is, `-rowindex - 1`. The array must be of length at least equal to the number of rows in the LP problem object.
x	An array. This array contains the values of the basic variables in the order specified by `head[]`. The array must be of length at least equal to number of rows in the LP problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXboundsa

```
int CPXboundsa(CPXCENVptr env, CPXCLPptr lp, int begin, int end, double * lblower,
double * lbupper, double * ublower, double * ubupper)
```

Definition file: cplex.h

The routine `CPXboundsa` accesses upper and lower sensitivity ranges for lower and upper variable bounds for a specified range of variable indices. The beginning and end of the range must be specified. Information for variable `j`, where `begin` <= `j` <= `end`, is returned in position `(j-begin)` of the arrays `lblower`, `lbupper`, `ublower`, and `ubupper`.

When the routine returns, the element `lblower[j-begin]` and `lbupper[j-begin]` will contain the lowest and highest value the lower bound of variable `j` can assume without affecting the optimality of the solution. Likewise, `ublower[j-begin]` and `ubupper[j-begin]` will contain the lowest and highest value the upper bound of variable `j` can assume without affecting the optimality of the solution.

Note

If you want sensitivity ranges only for lower bound, then both `lblower` and `lbupper` should be non NULL, and `ublower` and `ubupper` can be NULL.

Example

```
status = CPXboundsa (env, lp, 0, CPXgetnumcols(env,lp)-1,
                     lblower, lbupper, ublower, ubupper);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
begin	An integer specifying the beginning of the range of ranges to be returned.
end	An integer specifying the end of the range of ranges to be returned.
lblower	An array where the lower bound lower range values are to be returned. The length of this array must be at least `(end - begin + 1)`. May be NULL.
lbupper	An array where the lower bound upper range values are to be returned. The length of this array must be at least `(end - begin + 1)`. May be NULL.
ublower	An array where the upper bound lower range values are to be returned. The length of this array must be at least `(end - begin + 1)`. May be NULL.
ubupper	An array where the upper bound upper range values are to be returned. The length of this array must be at least (`end - begin + 1)`. May be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs. This routine fails if no basis exists.

Global function CPXgetsolnpoolx

```
int CPXgetsolnpoolx(CPXCENVptr env, CPXCLPptr lp, int soln, double * x, int begin,
int end)
```

Definition file: cplex.h

The routine `CPXgetsolnpoolx` accesses the solution values for a range of problem variables for a member of the solution pool. The beginning and end of the range must be specified.

Example

```
status = CPXgetsolnpoolx (env, lp, 5, x, 0, CPXgetnumcols(env, lp)-1);
```

See also the example `populate.c` in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
soln	An integer specifying the index of the solution pool member for which to return primal values. A value of -1 specifies that the incumbent should be used instead of a solution pool member.
x	An array to receive the values of a member of the solution pool for the problem. This array must be of length at least (`end - begin + 1`). If successful, `x[0]` through `x[end-begin]` contains the solution values.
begin	An integer specifying the beginning of the range of variable values to be returned.
end	An integer specifying the end of the range of variable values to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcloneprob

```
CPXLPptr CPXcloneprob(CPXCENVptr env, CPXCLPptr lp, int * status_p)
```

Definition file: cplex.h

The routine `CPXcloneprob` can be used to create a new CPLEX problem object and copy all the problem data from an existing problem object to it. Solution and starting information is not copied.

Example

```
copy = CPXcloneprob (env, lp, &status);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object of which a copy is to be created.
status_p	A pointer to an integer used to return any error code produced by this routine.

Returns:

If successful, `CPXcloneprob` returns a pointer that can be passed to other CPLEX routines to identify the problem object that has been created, and the argument `*status_p` is zero. If not successful, a NULL pointer is returned, and an error status is returned in the argument `*status_p`.

Global function CPXgetxqxax

```
int CPXgetxqxax(CPXCENVptr env, CPXCLPptr lp, double * xqxax, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetxqxax` is used to access quadratic constraint activity levels for a range of quadratic constraints in a quadratically constrained program (QCP). The beginning and end of the range must be specified.

Quadratic constraint activity is the sum of the linear and quadratic terms of the constraint evaluated with the values of the structural variables in the problem.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
xqxax	An array to receive the values of the quadratic constraint activity levels for each of the constraints in the specified range. The array must be of length at least (`end-begin+1`). If successful, `x[0]` through `x[end-begin]` contain the quadratic constraint activities.
begin	An integer indicating the beginning of the range of quadratic constraint activities to be returned.
end	An integer indicating the end of the range of quadratic constraint activities to be returned.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXdelindconstrs

`int` **`CPXdelindconstrs`**`(CPXCENVptr env, CPXLPptr lp, int begin, int end)`

Definition file: cplex.h

The routine `CPXdelindconstrs` deletes a range of indicator constraints. The range is specified by a lower index that represent the first indicator constraint to be deleted and an upper index that represents the last indicator constraint to be deleted. The indices of the constraints following those deleted constraints are automatically decreased by the number of deleted constraints.

Example

```
status = CPXdelindconstrs (env, lp, 10, 20);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
begin	An integer that specifies the numeric index of the first indicator constraint to be deleted.
end	An integer that specifies the numeric index of the last indicator constraint to be deleted.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetnumrows

`int` **`CPXgetnumrows`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetnumrows` accesses the number of rows in the constraint matrix, not including the objective function, quadratic constraints, or the bounds constraints on the variables.

Example

```
cur_numrows = CPXgetnumrows (env, lp);
```

See also the example `lpex1.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If the CPLEX problem object or environment does not exist, `CPXgetnumrows` returns the value 0 (zero); otherwise, it returns the number of rows.

Global function CPXNETgetstat

`int` **`CPXNETgetstat`**`(CPXCENVptr env, CPXCNETptr net)`

Definition file: cplex.h

The routine `CPXNETgetstat` returns the solution status for a network problem object.

Example

```
netstatus = CPXNETgetstat (env, net);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

Returns:

If no solution is available for the network problem object, CPXNETgetstat returns 0 (zero). When a solution exists, the possible return values are:

`CPX_STAT_OPTIMAL`	Optimal solution found.
`CPX_STAT_UNBOUNDED`	Problem has an unbounded ray.
`CPX_STAT_INFEASIBLE`	Problem is infeasible.
`CPX_STAT_INForUNB`	Problem is infeasible or unbounded.
`CPX_STAT_ABORT_IT_LIM`	Aborted due to iteration limit.
`CPX_STAT_ABORT_TIME_LIM`	Aborted due to time limit.
`CPX_STAT_ABORT_USER`	Aborted on user request.

Global function CPXdelmipstarts

`int` **`CPXdelmipstarts`**`(CPXCENVptr env, CPXLPptr lp, int begin, int end)`

Definition file: cplex.h

The routine `CPXdelmipstarts` deletes a range MIP starts. The range is specified using a beginning and ending index that represent the first and last MIP start to delete. The indices of the MIP starts following those deleted are automatically decreased by the number of deleted MIP starts.

Example

```
status = CPXdelmipstarts (env, lp, 10, 20);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
begin	An integer that specifies the numeric index of the first MIP start to be deleted.
end	An integer that specifies the numeric index of the last MIP start to be deleted.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetmipitcnt

`int` **`CPXgetmipitcnt`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetmipitcnt` accesses the cumulative number of simplex iterations used to solve a mixed integer problem.

Example

```
itcnt = CPXgetmipitcnt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Example

```
itcnt = CPXgetmipitcnt (env, lp);
```

Returns:

If a solution exists, `CPXgetmipitcnt` returns the total iteration count. If no solution, problem, or environment exists, `CPXgetmipitcnt` returns the value 0.

Global function CPXsolwritesolnpoolall

```
int CPXsolwritesolnpoolall(CPXCENVptr env, CPXCLPptr lp, const char * filename_str)
```

Definition file: cplex.h

The routine `CPXsolwritesolnpoolall` writes all the solutions in the solution pool to a file for the selected CPLEX problem object. The routine writes files in SOL format, which is an XML format.

The SOL format is documented in the stylesheet `solution.xsl` and schema `solution.xsd` in the `include` directory of the CPLEX distribution. *CPLEX File Formats Reference Manual* also documents this format briefly.

Example

```
status = CPXsolwritesolnpoolall (env, lp, "myfile.sol");
```

See Also: CPXsolwrite, CPXsolwritesolnpool

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the solutions should be written.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetnumqpnz

`int` **`CPXgetnumqpnz`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetnumqpnz` returns the number of nonzeros in the Q matrix of a problem object.

Example

```
numqpnz = CPXgetnumqpnz (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If successful, the routine returns the number of nonzeros in the Q matrix. If an error occurs, zero is returned.

Global function CPXgetub

```
int CPXgetub(CPXCENVptr env, CPXCLPptr lp, double * ub, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetub` accesses a range of upper bounds on the variables of a CPLEX problem object. The beginning and end of the range must be specified.

Unbounded Variables

If a variable lacks an upper bound, then `CPXgetub` returns a value greater than or equal to `CPX_INFBOUND`.

Example

```
status = CPXgetub (env, lp, ub, 0, cur_numcols-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
ub	An array where the specified upper bounds on the variables are to be returned. This array must be of length at least (`end - begin + 1`). The upper bound of variable `j` is returned in `ub[j-begin]`.
begin	An integer specifying the beginning of the range of upper bounds to be returned.
end	An integer specifying the end of the range of upper bounds to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsiftopt

```
int CPXsiftopt(CPXCENVptr env, CPXLPptr lp)
```

Definition file: cplex.h

This routine looks at a subset of the columns of a problem and uses either the barrier or simplex optimizer to solve this reduced model. That is, it solves a model consisting of a selected subset of columns. After solving this reduced model, it performs a pricing step. The pricing step serves two purposes:

- to select additional columns to enter the reduced model;
- to select existing columns to discard from the reduced model.

The routine repeats this procedure iteratively until it finds an optimal solution of the original problem.

This routine is useful for linear programming models (LPs) with a great many variables (columns) and relatively few constraints (rows). In colloquial terms, these problems are known as long, skinny models. This routine is **not** applicable to a model with quadratic terms in the objective function (QP) nor to models with quadratic terms among the constraints (QCP).

Example

```
status = CPXsiftopt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXfreeprob

```
int CPXfreeprob(CPXCENVptr env, CPXLPptr * lp_p)
```

Definition file: cplex.h

The routine `CPXfreeprob` removes the specified CPLEX problem object from the CPLEX environment and frees the associated memory used internally by CPLEX. The routine is used when the user has no need for further access to the specified problem data.

Example

```
status = CPXfreeprob (env, &lp);
```

See also the example `lpex1.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp_p	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Example

```
status = CPXfreeprob (env, &lp);
```

See also the example `lpex1.c` in the *CPLEX User's Manual* and in the standard distribution.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsiftitcnt

`int` **`CPXgetsiftitcnt`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetsiftitcnt` accesses the total number of sifting iterations to solve an LP problem.

Example

```
itcnt = CPXgetsiftitcnt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object as returned by `CPXcreateprob`.

Returns:

The routine returns the total iteration count if a solution exists. It returns zero if no solution exists or any other type of error occurs.

Global function CPXcopyctype

```
int CPXcopyctype(CPXCENVptr env, CPXLPptr lp, const char * xctype)
```

Definition file: cplex.h

The routine `CPXcopyctype` can be used to copy variable type information into a given problem. Variable types specify whether a variable is continuous, integer, binary, semi-continuous, or semi-integer. Adding `ctype` information automatically changes the problem type from continuous to mixed integer (from `CPXPROB_LP` to `CPXPROB_MILP`, from `CPXPROB_QP` to `CPXPROB_MIQP`, and from `CPXPROB_QCP` to `CPXPROB_MIQCP`), even if the provided `ctype` data specifies that all variables are continuous.

This routine allows the types of all the variables to be set in one function call. When `CPXcopyctype` is called, any current solution information is freed.

Note

Defining a variable `j` to be binary by setting the corresponding `ctype[j]='B'` does not change the bounds associated with that variable. A later call to `CPXmipopt` will change the bounds to 0 (zero) and 1 (one) and issue a warning.

Table 1: Possible values for elements of xctype

`CPX_CONTINUOUS`	`'C'`	continuous variable
`CPX_BINARY`	`'B'`	binary variable
`CPX_INTEGER`	`'I'`	general integer variable
`CPX_SEMICONT`	`'S'`	semi-continuous variable
`CPX_SEMIINT`	`'N'`	semi-integer variable

When you build or modify your problem with this routine, you can verify that the results are as you intended by calling `CPXcheckcopyctype` during application development.

Example

```
status = CPXcopyctype (env, lp, ctype);
```

See also the example `mipex1.c` distributed with the product.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
xctype	An array of length `CPXgetnumcols(env,lp)` containing the type of each column in the constraint matrix. Possible values appear in Table 1.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETgetarcnodes

```
int CPXNETgetarcnodes(CPXCENVptr env, CPXCNETptr net, int * fromnode, int * tonode,
int begin, int end)
```

Definition file: cplex.h

The routine `CPXNETgetarcnodes` is used to access the from-nodes and to-nodes for a range of arcs in the network stored in a network problem object.

Example

```
status = CPXNETgetarcnodes (env, net, fromnode, tonode,
                            0, cur_narcs-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
fromnode	Array in which to write the from-node indices of the requested arcs. If NULL is passed, no from-node indices are retrieved. Otherwise, the size of the array must be (`end-begin+1`).
tonode	Array in which to write the to-node indices of the requested arcs. If NULL is passed, no to-node indices are retrieved. Otherwise, the size of the array must be (`end-begin+1`).
begin	Index of the first arc to get nodes for.
end	Index of the last arc to get nodes for.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXobjsa

```
int CPXobjsa(CPXCENVptr env, CPXCLPptr lp, int begin, int end, double * lower,
double * upper)
```

Definition file: cplex.h

The routine `CPXobjsa` accesses upper and lower sensitivity ranges for objective function coefficients for a specified range of variable indices. The beginning and end of the range of variable indices must be specified.

Note

Information for variable `j`, where `begin` <= `j` <= `end`, is returned in position `(j-begin)` of the arrays `lower` and `upper`. The items `lower[j-begin]` and `upper[j-begin]` contain the lowest and highest value the objective function coefficient for variable `j` can assume without affecting the optimality of the solution.

Example

```
status = CPXobjsa (env, lp, 0, CPXgetnumcols(env,lp)-1,
                   lower, upper);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
begin	An integer specifying the beginning of the range of ranges to be returned.
end	An integer specifying the end of the range of ranges to be returned.
lower	An array where the objective function lower range values are to be returned. This array must be of length at least `(end - begin + 1)`.
upper	An array where the objective function upper range values are to be returned. This array must be of length at least `(end - begin + 1)`.

Returns:

The routine returns zero if successful and nonzero if an error occurs. This routine fails if no optimal basis exists.

Global function CPXgetnummipstarts

```
int CPXgetnummipstarts(CPXCENVptr env, CPXCLPptr lp)
```

Definition file: cplex.h

The routine `CPXgetnummipstarts` accesses the number of MIP starts in the CPLEX problem object.

Example

```
numsolns = CPXgetnummipstarts (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If the CPLEX problem object or environment does not exist, `CPXgetnummipstarts` returns the value 0 (zero); otherwise, it returns the number of solutions.

Global function CPXgetdj

```
int CPXgetdj(CPXCENVptr env, CPXCLPptr lp, double * dj, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetdj` accesses the reduced costs for a range of the variables of a linear or quadratic program. The beginning and end of the range must be specified.

Example

```
status = CPXgetdj (env, lp, dj, 0, CPXgetnumcols(env,lp)-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
dj	An array to receive the values of the reduced costs for each of the variables. This array must be of length at least (`end - begin + 1`). If successful, `dj[0]` through `dj[end-begin]` contain the values of the reduced costs.
begin	An integer specifying the beginning of the range of reduced-cost values to be returned.
end	An integer specifying the end of the range of reduced-costs values to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETfreeprob

```
int CPXNETfreeprob(CPXENVptr env, CPXNETptr * net_p)
```

Definition file: cplex.h

The routine `CPXNETfreeprob` deletes the network problem object pointed to by `net_p`. This also deletes all network problem data and solution data stored in the network problem object.

Example

```
CPXNETfreeprob (env, &net);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net_p	CPLEX network problem object to be deleted.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetparamname

```
int CPXgetparamname(CPXCENVptr env, int whichparam, char * name_str)
```

Definition file: cplex.h

The routine `CPXgetparamname` returns the name of a CPLEX parameter, given the symbolic constant (or reference number) for it.

The *CPLEX Parameters Reference Manual* provides a list of parameters with their types, options, and default values.

Example

```
status = CPXgetparamname (env, CPX_PARAM_ADVIND, param_string);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
whichparam	An integer specifying the symbolic constant (or reference number) of the desired parameter.
name_str	A character array (that is, a pointer to a buffer) of length at least `CPX_STR_PARAM_MAX` to hold the name of the selected parameter.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcopymipstart

```
int CPXcopymipstart(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices,
const double * values)
```

Definition file: cplex.h

This routine is **deprecated**. See `CPXaddmipstarts` instead to add multiple MIP starts to a CPLEX problem object.

The routine `CPXcopymipstart` copies MIP start values to a CPLEX problem object of type `CPXPROB_MILP`, `CPXPROB_MIQP`, or `CPXPROB_MIQCP`.

MIP start values may be specified for any subset of the integer or continuous variables in the problem. When optimization begins or resumes, CPLEX attempts to find a feasible MIP solution that is compatible with the set of values specified in the MIP start. When a partial MIP start is provided, CPLEX tries to extend it to a complete solution by solving a MIP over the variables whose values are **not** specified in the MIP start. The parameter `CPX_PARAM_SUBMIPNODELIM` controls the amount of effort CPLEX expends in trying to solve this secondary MIP. If CPLEX is able to find a complete feasible solution, that solution becomes the incumbent. If the specified MIP start values are infeasible, these values are retained for use in a subsequent repair heuristic. See the description of the parameter `CPX_PARAM_REPAIRTRIES` for more information about this repair heuristic.

This routine replaces any existing MIP start information in the problem. Use the routine `CPXchgmipstart` to modify or extend an existing MIP start.

Example

```
status = CPXcopymipstart (env, lp, cnt, indices, values);
```

The parameter `CPX_PARAM_ADVIND` must be set to 1 (one), its default value, or 2 (two) in order for the MIP start to be used.

See Also: CPXreadcopyorder, CPXreadcopymipstart, CPXchgmipstart, CPXaddmipstarts, CPXreadcopymipstart, CPXchgmipstarts

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt	An integer giving the number of entries in the list.
indices	An array of length `cnt` containing the numeric indices of the columns corresponding to the variables which are assigned starting values.
values	An array of length `cnt` containing the values to be used for the starting integer solution. The entry `values[j]` is the value assigned to the variable `indices[j]`. An entry `values[j]` greater than or equal to `CPX_INFBOUND` specifies that no value is set for the variable `indices[j]`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcopyprotected

```
int CPXcopyprotected(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXcopyprotected` specifies a set of variables that should not be substituted out of the problem. If presolve can fix a variable to a value, it is removed, even if it is specified in the protected list.

Example

```
status = CPXcopyprotected (env, lp, cnt, indices);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
cnt	The number of variables to be protected.
indices	An array of length `cnt` containing the column indices of variables to be protected from being substituted out.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETgetdj

`int` **`CPXNETgetdj`**`(CPXCENVptr env, CPXCNETptr net, double * dj, int begin, int end)`

Definition file: cplex.h

The routine `CPXNETgetdj` is used to access reduced costs for a range of arcs of the network stored in a network problem object.

For this function to succeed, a solution must exist for the problem object. If the solution is not feasible (`CPXNETsolninfo` returns `0` in argument `pfeasind_p`), the reduced costs are computed with respect to an objective function that penalizes infeasibilities.

Example

```
status = CPXNETgetdj (env, net, dj, 10, 20);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
dj	Array in which to write requested reduced costs. If NULL is passed, no reduced cost values are returned. Otherwise, `dj` must point to an array of size at least (`end-begin+1`).
begin	Index of the first arc for which a reduced cost value is to be obtained.
end	Index of the last arc for which a reduced cost value is to be obtained.

Example

```
status = CPXNETgetdj (env, net, dj, 10, 20);
```

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXdjfrompi

```
int CPXdjfrompi(CPXCENVptr env, CPXCLPptr lp, const double * pi, double * dj)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXdjfrompi` computes an array of reduced costs from an array of dual values. This routine is for linear programs. Use `CPXqpdjfrompi` for quadratic programs.

Example

```
status = CPXdjfrompi (env, lp, pi, dj);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object as returned by `CPXcreateprob`.
pi	An array that contains dual solution (`pi`) values for the problem, as returned by routines such as CPXuncrushpi `and CPXcrushpi. The array must be of length at least the number of rows in the problem object.`
dj	An array to receive the reduced cost values computed from the `pi` values for the problem object. The array must be of length at least the number of columns in the problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXchgrhs

```
int CPXchgrhs(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices, const
double * values)
```

Definition file: cplex.h

The routine `CPXchgrhs` changes the righthand side coefficients of a set of linear constraints in the CPLEX problem object.

Example

```
status = CPXchgrhs (env, lp, cnt, indices, values);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt	An integer that specifies the total number of righthand side coefficients to be changed, and thus specifies the length of the arrays `indices` and `values`.
indices	An array of length `cnt` containing the numeric indices of the rows corresponding to the linear constraints for which righthand side coefficients are to be changed.
values	An array of length `cnt` containing the new values of the righthand side coefficients of the linear constraints present in `indices`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXdelsolnpoolsolns

`int` **`CPXdelsolnpoolsolns`**`(CPXCENVptr env, CPXLPptr lp, int begin, int end)`

Definition file: cplex.h

The routine `CPXdelsolnpoolsolns` deletes a range of solutions from the solution pool. The range is specified using a lower and upper index that represent the first and last solution to be deleted, respectively. The indices of the solutions following those deleted are decreased by the number of deleted solutions.

Example

```
status = CPXdelsolnpoolsolns (env, lp, 10, 20);
```

See Also: CPXdelsetsolnpoolsolns

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
begin	An integer that specifies the numeric index of the first solution to be deleted.
end	An integer that specifies the numeric index of the last solution to be deleted.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXbranchcallbackbranchconstraints

```
int CPXbranchcallbackbranchconstraints(CPXCENVptr env, void * cbdata, int
wherefrom, double nodeest, int rcnt, int nzcnt, const double * rhs, const char *
sense, const int * rmatbeg, const int * rmatind, const double * rmatval, void *
userhandle, int * seqnum_p)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXbranchcallbackbranchconstraints` specifies the branches to be taken from the current node when the branch is specified by adding one or more constraints to the node problem. It may be called only from within a user-written branch callback function.

Constraints are in terms of the original problem if the parameter `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, constraints are in terms of the presolved problem.

Table 1: Values of sense[i]

L	less than or equal to constraint
E	equal to constraint
G	greater than or equal to constraint

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	A pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value that reports where the user-written callback was called from. This argument must be the value of `wherefrom` passed to the user-written callback.
nodeest	A double that specifies the value of the node estimate for the node to be created with this branch. The node estimate is used to select nodes from the branch-and-cut tree with certain values of the node selection parameter `CPX_PARAM_NODESEL`.
rcnt	An integer that specifies the number of constraints for the branch.
nzcnt	An integer that specifies the number of nonzero constraint coefficients for the branch. This specifies the length of the arrays `rmatind` and `rmatval`.
rhs	An array of length `rcnt` containing the righthand side term for each constraint for the branch.
sense	An array of length `rcnt` containing the sense of each constraint to be added for the branch. Values of the sense appear in Table 1.
rmatbeg	An array that with `rmatind` and `rmatval` defines the constraints for the branch.
rmatind	An array that with `rmatbeg` and `rmatval` defines the constraints for the branch.
rmatval	An array that with `rmatbeg` and `rmatind` defines the constraints for the branch. The format is similar to the format used to describe the constraint matrix in the routine `CPXaddrows`. Every row must be stored in sequential locations in this array from position `rmatbeg[i]` to `rmatbeg[i+1]-1` (or from `rmatbeg[i]` to `nzcnt -1` if `i=rcnt-1`). Each entry, `rmatind[i]`, specifies the column index of the corresponding coefficient, `rmatval[i]`. All rows must be contiguous, and `rmatbeg[0]` must be `0`.
userhandle	A pointer to user private data that should be associated with the node created by this branch. May

be NULL.

seqnum_p A pointer to an integer that, on return, will contain the sequence number that CPLEX has assigned to the node created from this branch. The sequence number may be used to select this node in later calls to the node callback.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXrefinemipstartconflictext

```
int CPXrefinemipstartconflictext(CPXCENVptr env, CPXLPptr lp, int mipstartindex,
int grpcnt, int concnt, const double * grppref, const int * grpbeg, const int *
grpind, const char * grptype)
```

Definition file: cplex.h

The routine `CPXrefinemipstartconflictext` refines a conflict of a MIP start that may include quadratic constraints, indicator constraints, or special ordered sets, as well as linear contraints and bounds. In other words, this routine extends `CPXrefinemipstartconflict` to MIP starts with indicator constraints, quadratic constraints, or special ordered sets (SOSs) and to situations where groups of constraints should be considered as a single constraint. That is, this routine identifies a minimal conflict for the infeasibility of the MIP start or a subset of its constraints.

Like the routine `CPXrefineconflictext`, this routine considers constraints in groups. If any constraint in a group participates in the conflict, the entire group is determined to do so. No further detail about the constraints within that group is returned. A group may consist of a single constraint.

A group may be assigned a preference; that is, a value specifying how much the user wants the group to be part of a conflict. A group with a higher preference is more likely to be included in the conflict. However, no guarantee is made when a minimal conflict is returned that other conflicts containing groups with a greater preference do not exist.

To retrieve information about the conflict computed by `CPXrefinemipstartconflictext`, call the routine `CPXgetconflictext`.

To write the conflict to a file, use the routine `CPXclpwrite`.

This conflict is a submodel of the original model with the property that CPLEX cannot generate a solution from the chosen MIP start using the given level of effort and that removal of any constraint or bound in the conflict invalidates that property.

The parameters `CPX_PARAM_CUTUP`, `CPX_PARAM_CUTLO`, `CPX_PARAM_OBJULIM`, `CPX_PARAM_OBJLLIM` do not influence this routine. If you want to study infeasibilities introduced by those parameters, consider adding an objective function constraint to your model to enforce their effect before you invoke this routine.

When the MIP start was added to the current model, an effort level may have been associated with it to specify to CPLEX how much effort to expend in transforming the MIP start into a feasible solution. This routine respects effort levels **except** level 1 (one): check feasibility. It does not check feasibility.

Table 1: Possible values for elements of grptype

`CPX_CON_LOWER_BOUND`	`1`	variable lower bound
`CPX_CON_UPPER_BOUND`	`2`	variable upper bound
`CPX_CON_LINEAR`	`3`	linear constraint
`CPX_CON_QUADRATIC`	`4`	quadratic constraint
`CPX_CON_SOS`	`5`	special ordered set
`CPX_CON_INDICATOR`	`6`	indicator constraint

Parameters:

env	A pointer to the CPLEX environment as returned by the routine `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
grpcnt	The number of constraint groups to be considered.
concnt	

An integer specifying the total number of elements passed in the arrays `grpind` and `grptype`, or, equivalently, the end of the last group in `grpind`.

grppref An array of preferences for the groups. The value `grppref[i]` specifies the preference for the group designated by the index `i`. A negative value specifies that the corresponding group should not be considered in the computation of a conflict. In other words, such groups are not considered part of the MIP start. Groups with a preference of 0 (zero) are always considered to be part of the conflict. No further checking is performed on such groups.

grpbeg An array of integers specifying where the constraint indices for each group begin in the array `grpind`. Its length must be at least `grpcnt`.

grpind An array of integers containing the indices for the constraints in each group. For each of the various types of constraints listed in the table, the constraint indices range from 0 (zero) to the number of constraints of that type minus one. Group `i` contains the constraints with the indices `grpind[grpbeg[i]], ..., grpind[grpbeg[i+1]-1]` for `i` less than `grpcnt-1`, and `grpind[grpbeg[i]], ..., grpind[concnt-1]` for `i == grpcnt-1`. Its length must be at least `concnt`. A constraint must not be referenced more than once in this array. For any constraint in the problem that is not a member of a group and thus does not appear in this array, the constraint is assigned a default preference of 0 (zero). Thus such constraints are included in the conflict without any analysis.

grptype An array of characters containing the constraint types for the constraints as they appear in groups. The types of the constraints in group `i` are specified in `grptype[grpbeg[i]], ..., grptype[grpbeg[i+1]-1]` for `i` less than `grpcnt-1` and `grptype[grpbeg[i]], ..., grptype[concnt-1]` for `i == grpcnt-1`. Its length must be at least `concnt`, and every constraint must appear at most once in this array. Possible values appear in the table.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetbestobjval

```
int CPXgetbestobjval(CPXCENVptr env, CPXCLPptr lp, double * objval_p)
```

Definition file: cplex.h

The routine `CPXgetbestobjval` accesses the currently best known bound of all the remaining open nodes in a branch-and-cut tree.

It is computed for a minimization problem as the minimum objective function value of all remaining unexplored nodes. Similarly, it is computed for a maximization problem as the maximum objective function value of all remaining unexplored nodes.

For a regular MIP optimization using `CPXmipopt`, this value is also the best known bound on the optimal solution value of the MIP problem. In fact, when a problem has been solved to optimality, this value matches the optimal solution value.

However, for `CPXpopulate` the value can also exceed the optimal solution value if CPLEX has already solved the model to optimality but continues to search for additional solutions.

Example

```
status = CPXgetbestobjval (env, lp, &objval);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
objval_p	A pointer to the location where the best node objective value is returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetqconstrinfeas

```
int CPXgetqconstrinfeas(CPXCENVptr env, CPXCLPptr lp, const double * x, double *
infeasout, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetqconstrinfeas` computes the infeasibility of a given solution for a range of quadratic constraints. The beginning and end of the range must be specified. For each constraint, the infeasibility value returned is 0 (zero) if the constraint is satisfied. Otherwise, the infeasibility value returned is the amount by which the righthand side of the constraint must be changed to make the queried solution valid. It is positive for a less-than-or-equal-to constraint and negative for a greater-than-or-equal-to constraint.

Example

```
status = CPXgetqconstrinfeas (env, lp, NULL, infeasout, 0, CPXgetnumqconstrs(env,lp)-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
x	The solution whose infeasibility is to be computed. May be NULL in which case the resident solution is used.
infeasout	An array to receive the infeasibility value for each of the quadratic constraints. This array must be of length at least (`end - begin + 1`).
begin	An integer indicating the beginning of the range of quadratic constraints whose infeasibility is to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetmipstartindex

`int` **`CPXgetmipstartindex`**`(CPXCENVptr env, CPXCLPptr lp, const char * lname_str, int * index_p)`

Definition file: cplex.h

The routine `CPXgetmipstartindex` searches for the index number of the specified MIP start in a CPLEX problem object.

Example

```
status = CPXgetmipstartindex (env, lp, "mipstart6", &rowindex);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lname_str	A MIP start name to search for.
index_p	A pointer to an integer to hold the index number of the MIP start with name `lname_str`. If the routine is successful, `*index_p` contains the index number; otherwise, `*index_p` is undefined.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXfreelazyconstraints

```
int CPXfreelazyconstraints(CPXCENVptr env, CPXLPptr lp)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXfreelazyconstraints` clears the list of lazy constraints that have been previously specified through calls to `CPXaddlazyconstraints`.

Example

```
status = CPXfreelazyconstraints (env, lp);
```

Parameters:

env A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

lp A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXchgobj

```
int CPXchgobj(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices, const
double * values)
```

Definition file: cplex.h

The routine `CPXchgobj` changes the linear objective coefficients of a set of variables in a CPLEX problem object.

Example

```
status = CPXchgobj (env, lp, cnt, indices, values);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt	An integer that specifies the total number of objective coefficients to be changed, and thus specifies the length of the arrays `indices` and `values`.
indices	An array of length `cnt` containing the numeric indices of the columns corresponding to the variables for which objective coefficients are to be changed.
values	An array of length `cnt` containing the new values of the objective coefficients of the variables specified in `indices`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXpivotout

```
int CPXpivotout(CPXCENVptr env, CPXLPptr lp, const int * clist, int clen)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXpivotout` pivots a list of fixed variables out of the resident basis. Variables are fixed when the absolute difference between the lower and upper bounds is at most 1.0e-10.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
clist	An array of length `clen`, containing the column indices of the variables to be pivoted out of the basis. If any of these variables is not fixed, `CPXpivotout` returns an error code.
clen	An integer that specifies the number of entries in the array `clist[]`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetnumcuts

`int` **`CPXgetnumcuts`**`(CPXCENVptr env, CPXCLPptr lp, int cuttype, int * num_p)`

Definition file: cplex.h

The routine `CPXgetnumcuts` accesses the number of cuts of the specified type in use at the end of the previous optimization.

Example

```
status = CPXgetnumcuts (env, lp, CPX_CUT_COVER, &numcovers);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cuttype	An integer specifying the type of cut for which to return the number.
num_p	An pointer to an integer to contain the number of cuts.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcopypartialbase

```
int CPXcopypartialbase(CPXCENVptr env, CPXLPptr lp, int ccnt, const int * cindices,
const int * cstat, int rcnt, const int * rindices, const int * rstat)
```

Definition file: cplex.h

The routine `CPXcopypartialbase` copies a partial basis into an LP problem object. Basis status values do not need to be specified for every variable or slack, surplus, or artificial variable. If the status of a variable is not specified, it is made nonbasic at its lower bound if the lower bound is finite; otherwise, it is made nonbasic at its upper bound if the upper bound is finite; otherwise, it is made nonbasic at 0.0 (zero). If the status of a slack, surplus, or artificial variable is not specified, it is made basic.

Table 1: Values of cstat[i]

`CPX_AT_LOWER`	`0`	variable at lower bound
`CPX_BASIC`	`1`	variable is basic
`CPX_AT_UPPER`	`2`	variable at upper bound
`CPX_FREE_SUPER`	`3`	variable free and nonbasic

Table 2: Status of rows other than ranged rows in rstat[i]

`CPX_AT_LOWER`	`0`	associated slack variable is nonbasic at value 0.0 (zero)
`CPX_BASIC`	`1`	associated slack, surplus, or artificial variable is basic

Table 3: Status of ranged rows in rstat[i]

`CPX_AT_LOWER`	`0`	associated slack variable nonbasic at its lower bound
`CPX_BASIC`	`1`	associated slack variable basic
`CPX_AT_UPPER`	`2`	associated slack variable nonbasic at its upper bound

Example

```
status = CPXcopypartialbase (env, lp, ccnt, colind, colstat,
                             rcnt, rowind, rowstat);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
ccnt	An integer that specifies the number of variable or column status values specified, and is the length of the `cindices` and `cstat` arrays.
cindices	An array of length `ccnt` that contains the indices of the variables for which status values are being specified.
cstat	An array of length `ccnt` where the *ith* entry contains the status for variable `cindices[i]`.
rcnt	An integer that specifies the number of slack, surplus, or artificial status values specified, and is the length of the `rindices` and `rstat` arrays.
rindices	An array of length `rcnt` that contains the indices of the slack, surplus, or artificial variables for which status values are being specified.
rstat	An array of length `rcnt` where the i-th entry contains the status for slack, surplus, or artificial `rindices[i]`. For rows other than ranged rows, the array element `rstat[i]` has the meaning summarized in Table 2. For ranged rows, the array element `rstat[i]` has the meaning summarized

in Table 3.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsetlogfile

```
int CPXsetlogfile(CPXENVptr env, CPXFILEptr lfile)
```

Definition file: cplex.h

The routine `CPXsetlogfile` modifies the log file to which messages from all four CPLEX-defined channels are written.

> **Note**
>
> A call to `CPXsetlogfile` is equivalent to directing output from the `cpxresults`, `cpxwarning`, `cpxerror` and `cpxlog` message channels to a single file.

Example

```
status = CPXsetlogfile (env, logfile);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lfile	A CPXFILEptr to the log file. This routine sets `lfile` to be the file pointer for the current log file. A NULL pointer may be passed if no log file is desired. NULL is the default value. Before calling this routine, obtain this pointer with a call to `CPXfopen`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetprobtype

`int` **`CPXgetprobtype`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetprobtype` accesses the problem type that is currently stored in a CPLEX problem object.

Example

```
probtype = CPXgetprobtype (env, lp);
```

Return values

Value	Symbolic Constant	Meaning
-1	-	Error: no problem or environment.
0	`CPXPROB_LP`	Linear program; no quadratic data or `ctype` information stored.
1	`CPXPROB_MILP`	Problem with `ctype` information.
3	`CPXPROB_FIXEDMILP`	Problem with `ctype` information, integer variables fixed.
5	`CPXPROB_QP`	Problem with quadratic data stored.
7	`CPXPROB_MIQP`	Problem with quadratic data and `ctype` information.
8	`CPXPROB_FIXEDMIQP`	Problem with quadratic data and `ctype` information, integer variables fixed.
`10`	`CPXPROB_QCP`	Problem with quadratic constraints.
`11`	`CPXPROB_MIQCP`	Problem with quadratic constraints and ctype information.

See Also: CPXchgprobtype

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The values returned by `CPXgetprobtype` appear in the table.

Global function CPXsetinfocallbackfunc

```
int CPXsetinfocallbackfunc(CPXENVptr env, int(CPXPUBLIC *callback)(CPXCENVptr, void
*, int, void *), void * cbhandle)
```

Definition file: cplex.h

The routine `CPXsetinfocallbackfunc` sets the user-written callback routine that CPLEX calls regularly during the optimization of a mixed integer program and during certain cut generation routines.

This routine enables the user to create a separate callback function to be called during the solution of mixed integer programming problems (MIPs). Unlike any other callback routines, this user-written callback function only retrieves information about MIP search. It does not control the search, though it allows the search to terminate. The user-written callback function is allowed to call only two other routines: `CPXgetcallbackinfo` and `CPXgetcallbackincumbent`.

The prototype for the callback function is identical to that of `CPXsetmipcallbackfunc`.

Example

```
status = CPXsetinfocallbackfunc (env, mycallback, NULL);
```

Parameters

`env`

A pointer to the CPLEX environment, as returned by one of the `CPXopenCPLEX` routines.

`callback`

A pointer to a user-written callback function. Setting `callback` to NULL will prevent any callback function from being called during optimization. The call to `callback` will occur after every node during optimization and during certain cut generation routines. This function must be written by the user. Its prototype is explained in the Callback description.

`cbhandle`

A pointer to user private data. This pointer will be passed to the callback function.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle);
```

This is the user-written callback routine.

Callback return value

A nonzero return value terminates the optimization. That is, if the user-written callback function returns nonzero, it signals that CPLEX should terminate optimization.

Callback arguments

`env`

A pointer to the CPLEX environment that was passed into the associated optimization routine.

`cbdata`

A pointer passed from the optimization routine to the user-written callback function that identifies the problem being optimized. The only purpose for the `cbdata` pointer is to pass it to the routine `CPXgetcallbackinfo`.

`wherefrom`

An integer value reporting from which optimization algorithm the user-written callback function was called. Possible values and their meaning appear in the table.

Value	Symbolic Constant	Meaning
101	CPX_CALLBACK_MIP	From mipopt
107	CPX_CALLBACK_MIP_PROBE	From probing or clique merging
108	CPX_CALLBACK_MIP_FRACCUT	From Gomory fractional cuts
109	CPX_CALLBACK_MIP_DISJCUT	From disjunctive cuts
110	CPX_CALLBACK_MIP_FLOWMIR	From Mixed Integer Rounding (MIR) cuts

`cbhandle`

A pointer to user private data as passed to `CPXsetinfocallbackfunc`.

See Also: CPXgetcallbackinfo, CPXsetmipcallbackfunc

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetincumbentcallbackfunc

```
void CPXgetincumbentcallbackfunc(CPXCENVptr env, int(CPXPUBLIC
**incumbentcallback_p)(CALLBACK_INCUMBENT_ARGS), void ** cbhandle_p)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetincumbentcallbackfunc` accesses the user-written callback to be called by CPLEX during MIP optimization after an integer solution has been found but before this solution replaces the incumbent. This callback can be used to discard solutions that do not meet criteria beyond that of the mixed integer programming formulation.

Example

```
CPXgetincumbentcallbackfunc(env, &current_incumbentcallback, &current_handle);
```

See also *Advanced MIP Control Interface* in the *CPLEX User's Manual*.

For documentation of callback arguments, see the routine `CPXsetincumbentcallbackfunc`.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`incumbentcallback_p`

The address of the pointer to the current user-written incumbent callback. If no callback has been set, the pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the user's private pointer.

See Also: CPXsetincumbentcallbackfunc

Returns:

This routine does not return a result.

Global function CPXNETgetitcnt

`int` **`CPXNETgetitcnt`**`(CPXCENVptr env, CPXCNETptr net)`

Definition file: cplex.h

The routine `CPXNETgetitcnt` accesses the total number of network simplex iterations for the most recent call to `CPXNETprimopt`, for a network problem object.

Example

```
itcnt = CPXNETgetitcnt (env, net);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

net A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

Returns:

Returns the total number of network simplex iterations for the last call to `CPXNETprimopt`, for a network problem object. If `CPXNETprimopt` has not been called, zero is returned. If an error occurs, -1 is returned and an error message is issued.

Global function CPXchgrowname

```
int CPXchgrowname(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices, char
** newname)
```

Definition file: cplex.h

This routine changes the names of linear constraints in a CPLEX problem object. If this routine is performed on a problem object with no constraint names, default names are created before the change is made.

Example

```
status = CPXchgrowname (env, lp, cnt, indices, values);
```

See Also: CPXdelnames

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt	An integer that specifies the total number of linear constraint names to be changed, and thus specifies the length of the arrays `indices` and `newname`.
indices	An array of length `cnt` containing the numeric indices of the linear constraints for which the names are to be changed.
newname	An array of length `cnt` containing the strings of the new names for the linear constraints specified in `indices`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsolnpoolfiltertype

`int` **`CPXgetsolnpoolfiltertype`**`(CPXCENVptr env, CPXCLPptr lp, int * ftype_p, int which)`

Definition file: cplex.h

Accesses the type of a filter of the solution pool.

This routine accesses the type of the filter, specified by the argument `which`, of the solution pool associated with the LP problem specified by the argument `lp`.

Example

```
status = CPXgetsolnpoolfiltertype (env, lp, &ftype, i);
```

The argument `ftype_p` specifies the type of filter: either a diversity filter or a range filter. Table 1 summarizes the possible values of this argument.

Table 1: Possible types of filters

Symbolic name	Integer value	Meaning
CPX_SOLNPOOL_FILTER_DIVERSITY	1	diversity filter
CPX_SOLNPOOL_FILTER_RANGE	2	range filter

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
ftype_p	The filter type: either diversity or range filter.
which	The index of the filter.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcheckcopyqpsep

```
int CPXcheckcopyqpsep(CPXCENVptr env, CPXCLPptr lp, const double * qsepvec)
```

Definition file: cplex.h

The routine `CPXcheckcopyqpsep` validates the argument of the corresponding routine `CPXcopyqpsep`. This data checking routine is found in source format in the file `check.c` provided with the standard CPLEX distribution. To call this routine, you must compile and link `check.c` with your program as well as the CPLEX Callable Library.

The routine `CPXcheckcopyqpsep` has the same argument list as `CPXcopyqpsep`. The second argument, `lp`, is technically a pointer to a constant LP object of type `CPXCLPptr` rather than type `CPXLPptr`, as this routine will not modify the model. For most user applications, this distinction is unimportant.

Example

```
status = CPXcheckcopyqpsep (env, lp, qsepvec);
```

Returns:

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.

Global function CPXcheckaddcols

`int` **`CPXcheckaddcols`**`(CPXCENVptr env, CPXCLPptr lp, int ccnt, int nzcnt, const double * obj, const int * cmatbeg, const int * cmatind, const double * cmatval, const double * lb, const double * ub, char ** colname)`

Definition file: cplex.h

The routine `CPXcheckaddcols` validates the arguments of the corresponding `CPXaddcols` routine. This data checking routine is found in source format in the file `check.c` which is provided with the standard CPLEX distribution. To call this routine, you must compile and link `check.c` with your program as well as the CPLEX Callable Library.

The `CPXcheckaddcols` routine has the same argument list as the `CPXaddcols` routine. The second argument, `lp`, is technically a pointer to a constant LP object of type `CPXCLPptr` rather than type `CPXLPptr`, as this routine will not modify the problem. For most user applications, this distinction is unimportant.

Example

```
status = CPXcheckaddcols (env, lp, ccnt, nzcnt, obj, cmatbeg,
                          cmatind, cmatval, lb, ub, newcolname);
```

Returns:

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.

Global function CPXgetx

```
int CPXgetx(CPXCENVptr env, CPXCLPptr lp, double * x, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetx` accesses the solution values for a range of problem variables. The beginning and end of the range must be specified.

Example

```
status = CPXgetx (env, lp, x, 0, CPXgetnumcols(env, lp)-1);
```

See also the example `lpex2.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
x	An array to receive the values of the primal variables for the problem. This array must be of length at least (`end - begin + 1`). If successful, `x[0]` through `x[end-begin]` contains the solution values.
begin	An integer specifying the beginning of the range of variable values to be returned.
end	An integer specifying the end of the range of variable values to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXflushstdchannels

```
int CPXflushstdchannels(CPXCENVptr env)
```

Definition file: cplex.h

The routine `CPXflushstdchannels` flushes the output buffers of the four standard channels `cpxresults`, `cpxwarning`, `cpxerror`, and `cpxlog`. Use this routine where it is important to see all of the output created by CPLEX either on the screen or in a disk file without calling `CPXflushchannel` for each of the four channels.

Example

```
status = CPXflushstdchannels (env);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETcopynet

```
int CPXNETcopynet(CPXCENVptr env, CPXNETptr net, int objsen, int nnodes, const
double * supply, char ** nnames, int narcs, const int * fromnode, const int *
tonode, const double * low, const double * up, const double * obj, char ** anames)
```

Definition file: cplex.h

The routine `CPXNETcopynet` copies a network to a network object, overriding any other network saved in the object. The network to be copied is specified by providing the:

- the objective sense
- number of nodes
- supply values for each node
- names for each node
- number of arcs
- indices of the from-nodes (or, equivalently, the tail nodes) for each arc
- indices of the to-nodes (or, equivalently, the head nodes) for each arc
- lower and upper bounds on flow through each arc
- cost for flow through each arc
- names of each arc.

The arcs are numbered according to the order given in the `fromnode` and `tonode` arrays. Some of the parameters are optional and replaced by default values if NULL is passed for them.

Example

```
status = CPXNETcopynet (env, net, CPX_MAX, nnodes, supply, NULL,
                        narcs, fromnode, tonode, NULL, NULL, obj,
                        NULL);
```

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`net`

A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

`objsen`

Optimization sense of the network to be copied. It may take values `CPX_MAX` for a maximization problem or `CPX_MIN` for a minimization problem.

`nnodes`

Number of nodes to be copied to the network object.

`supply`

Supply values for the nodes. If NULL is passed all supply values default to 0 (zero). Otherwise, the size of the array must be at least `nnodes`.

`nnames`

Pointer to an array of names for the nodes. If NULL is passed, no names are assigned to the nodes. Otherwise, the size of the array must be at least `nnodes` and every name in the array must be a string terminating in 0 (zero).

`narcs`

Number of arcs to be copied to the network object.

`fromnode`

The array of indices in each arc's from-node. The indices must be in the range `[0, nnodes-1]`. The size of the array must be at least `narcs`.

`tonode`

The array of indices in each arc's to-node. The indices must be in the range `[0, nnodes-1]`. The size of the array must be at least `narcs`.

`low`

Pointer to an array of lower bounds on the flow through arcs. If NULL is passed, all lower bounds default to 0 (zero). Otherwise, the size of the array must be at least `narcs`. Values less than or equal to `-CPX_INFBOUND` are considered `-infinity`.

`up`

Pointer to an array of upper bounds on the flow through arcs. If NULL is passed, all lower bounds default to `CPX_INFBOUND`. Otherwise, the size of the array must be at least `narcs`. Values greater than or equal to `CPX_INFBOUND` are considered infinity.

`obj`

Pointer to an array of objective values for flow through arcs. If NULL is passed, all objective values default to 0 (zero). Otherwise, the size of the array must be at least `narcs`.

`anames`

Pointer to an array of names for the arcs. If NULL is passed, no names are assigned to the nodes. Otherwise, the size of the array must be at least `narcs`, and every name in the array must be a string terminating in 0 (zero).

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXNETchgnodename

```
int CPXNETchgnodename(CPXCENVptr env, CPXNETptr net, int cnt, const int * indices,
char ** newname)
```

Definition file: cplex.h

The routine `CPXNETchgnodename` changes the names of a set of nodes in the network stored in a network problem object.

Example

```
status = CPXNETchgnodename (env, net, 10, indices, newname);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
cnt	An integer that indicates the total number of node names to be changed. Thus `cnt` specifies the length of the arrays `indices` and `name`.
indices	An array of length `cnt` containing the numeric indices of the nodes for which the names are to be changed.
newname	An array of length `cnt` containing the new names for the nodes specified in `indices`.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXchgbds

```
int CPXchgbds(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices, const char
* lu, const double * bd)
```

Definition file: cplex.h

The routine `CPXchgbds` changes the lower or upper bounds on a set of variables of a problem. Several bounds can be changed at once, with each bound specified by the index of the variable with which it is associated. The value of a variable can be fixed at one value by setting the upper and lower bounds to the same value.

Unbounded Variables

If a variable lacks a lower bound, then `CPXgetlb` returns a value less than or equal to `-CPX_INFBOUND`.

If a variable lacks an upper bound, then `CPXgetub` returns a value greater than or equal to `CPX_INFBOUND`.

These conventions about unbounded variables should be taken into account when you change bounds with `CPXchgbds`.

Example

```
status = CPXchgbds (env, lp, cnt, indices, lu, bd);
```

Values of lu denoting lower or upper bound in indices[j]

`lu[j]`	`= 'L'`	`bd[j]` is a lower bound
`lu[j]`	`= 'U'`	`bd[j]` is an upper bound
`lu[j]`	`= 'B'`	`bd[j]` is the lower and upper bound

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt	An integer that specifies the total number of bounds to be changed, and thus specifies the length of the arrays `indices`, `lu`, and `bd`.
indices	An array of length `cnt` containing the numeric indices of the columns corresponding to the variables for which bounds are to be changed.
lu	An array of length `cnt` containing characters that tell whether the corresponding entry in the array `bd` specifies the lower or upper bound on column `indices[j]`. Possible values appear in the table.
bd	An array of length `cnt` containing the new values of the lower or upper bounds of the variables present in `indices`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXreadcopymipstarts

`int` **`CPXreadcopymipstarts`**`(CPXCENVptr env, CPXLPptr lp, const char * filename_str)`

Definition file: cplex.h

The routine `CPXreadcopymipstarts` reads a `MST` file and copies the information of all the MIP starts contained in this file into a CPLEX problem object. The parameter `CPX_PARAM_ADVIND` must be set to 1 (one), its default value, or 2 (two) in order for the MIP starts to be used.

Example

```
status = CPXreadcopymipstart(env, lp, "myprob.mst");
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A string containing the name of the `MST` file.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetnumquad

```
int CPXgetnumquad(CPXCENVptr env, CPXCLPptr lp)
```

Definition file: cplex.h

The routine `CPXgetnumquad` returns the number of variables that have quadratic objective coefficients in a CPLEX problem object.

Example

```
numquad = CPXgetnumquad (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If successful, the routine returns the number of variables having quadratic coefficients. If an error occurs, 0 is returned.

Global function CPXaddchannel

`CPXCHANNELptr` **`CPXaddchannel`**`(CPXENVptr env)`

Definition file: cplex.h

The routine `CPXaddchannel` instantiates a new channel object.

Example

```
mychannel = CPXaddchannel (env);
```

See also `lpex5.c` in the *CPLEX User's Manual*.

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

Returns:

If successful, `CPXaddchannel` returns a pointer to the new channel object; otherwise, it returns NULL.

Global function CPXNETgetarcindex

```
int CPXNETgetarcindex(CPXCENVptr env, CPXCNETptr net, const char * lname_str, int *
index_p)
```

Definition file: cplex.h

The routine `CPXNETgetarcindex` returns the index of the specified arc (in the network stored in a network problem object) in the integer pointed to by `index_p`.

Example

```
status = CPXNETgetarcindex (env, net, "from_a_to_b", &index);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
lname_str	Name of the arc to look for.
index_p	A pointer to an integer to hold the arc index. If the routine is successful, `*index_p` contains the index number; otherwise, `*index_p` is undefined.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetnumcols

```
int CPXgetnumcols(CPXCENVptr env, CPXCLPptr lp)
```

Definition file: cplex.h

The routine `CPXgetnumcols` accesses the number of columns in the constraint matrix, or equivalently, the number of variables in the CPLEX problem object.

Example

```
cur_numcols = CPXgetnumcols (env, lp);
```

See also the example `lpex1.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If the problem object or environment does not exist, `CPXgetnumcols` returns the value 0 (zero); otherwise, it returns the number of columns (variables).

Global function CPXgetpi

```
int CPXgetpi(CPXCENVptr env, CPXCLPptr lp, double * pi, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetpi` accesses the dual values for a range of the constraints of a linear or quadratic program. The beginning and end of the range must be specified.

Example

```
status = CPXgetpi (env, lp, pi, 0, CPXgetnumrows(env,lp)-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
pi	An array to receive the values of the dual variables for each of the constraints. This array must be of length at least (`end - begin + 1`). If successful, `pi[0]` through `pi[end-begin]` contain the dual values.
begin	An integer specifying the beginning of the range of dual values to be returned.
end	An integer specifying the end of the range of dual values to be returned.

Example

```
status = CPXgetpi (env, lp, pi, 0, CPXgetnumrows(env,lp)-1);
```

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXreadcopyprob

`int` **`CPXreadcopyprob`**`(CPXCENVptr env, CPXLPptr lp, const char * filename_str, const char * filetype_str)`

Definition file: cplex.h

The routine `CPXreadcopyprob` reads an MPS, LP, or SAV file into an existing CPLEX problem object. Any existing data associated with the problem object is destroyed. The problem can then be optimized by any one of the optimization routines. To determine the contents of the data, use CPLEX query routines.

The type of the file may be specified with the `filetype` argument. When the `filetype` argument is NULL, the file name is checked for one of these suffixes: `.lp`, `.mps`, or `.sav`. CPLEX will also look for the following additional optional suffixes: `.Z`, `.gz`, or `.bz2`.

If the file name matches one of these patterns, `filetype` is set accordingly. If `filetype` is NULL and none of these strings is found at the end of the file name, or if the specified type is not recognized, CPLEX attempts automatically to detect the type of the file by examining the first few bytes.

If the file name ends in `.gz`, `.bz2`, or `.z`, the file is read as a compressed file on platforms where the corresponding file-compression application has been installed properly. Thus, a file name ending in `.sav` is read as a SAV format file, while a file name ending in `.sav.gz` is read as a compressed SAV format file.

Microsoft Windows does not support reading compressed files with this API.

Values of filetype_str

`SAV`	Use SAV format
`MPS`	Use MPS format
`LP`	Use LP format

Example

```
status = CPXreadcopyprob (env, lp, "myprob.mps", NULL);
```

See also the example `lpex2.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	The name of the file from which the problem should be read.
filetype_str	A case-insensitive string containing the type of the file (one of the strings in the table). May be NULL, in which case the file type is inferred from the last characters of the file name.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetmipstartname

```
int CPXgetmipstartname(CPXCENVptr env, CPXCLPptr lp, char ** name, char * store,
int storesz, int * surplus_p, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetmipstartname` accesses a range of names of MIP starts in a CPLEX problem object. The beginning and end of the range, along with the length of the array in which the names of the MIP starts are to be returned, must be specified.

Note

If the value of `storesz` is 0 (zero), then the negative of the value of `surplus_p` returned specifies the total number of characters needed for the array `name`.

Example

```
status = CPXgetmipstartname (env, lp, name, store,
                         storesz, &surplus, 0,
                         cur_nummipstarts-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
name	An array of pointers to the MIP start names stored in the array `name`. This array must be of length at least (`end - begin + 1`). The pointer to the name of MIP start `i` is returned in `name[i-begin]`.
store	An array of characters where the specified MIP start names are to be returned. May be NULL if `storesz` is 0 (zero).
storesz	An integer specifying the length of the array `store`. May be 0 (zero).
surplus_p	A pointer to an integer to contain the difference between `storesz` and the total amount of memory required to store the requested names. A nonnegative value of `surplus_p` specifies that `storesz` was sufficient. A negative value specifies that it was insufficient and that the routine could not complete its task. In that case, `CPXgetmipstartname` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `surplus_p` specifies the amount of insufficient space in the array `store`.
begin	An integer specifying the beginning of the range of MIP start names to be returned.
end	An integer specifying the end of the range of MIP start names to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` specifies that insufficient space was available in the `store` array to hold the names.

Global function CPXNETgetnodeindex

```
int CPXNETgetnodeindex(CPXCENVptr env, CPXCNETptr net, const char * lname_str, int
* index_p)
```

Definition file: cplex.h

The routine `CPXNETgetnodeindex` returns the index of the specified node (in the network stored in a network problem object) in the integer pointed to by `index_p`.

Example

```
status = CPXNETgetnodeindex (env, net, "root", &index);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
lname_str	Name of the node to look for.
index_p	A pointer to an integer to hold the node index. If the routine is successful, `*index_p` contains the index number; otherwise, `*index_p` is undefined.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXNETreadcopybase

```
int CPXNETreadcopybase(CPXCENVptr env, CPXNETptr net, const char * filename_str)
```

Definition file: cplex.h

The routine `CPXNETreadcopybase` reads a basis file in `BAS` format and copies the basis to a network problem object. If no arc or node names are available for the problem object when reading the basis file, default names are assumed. Any basis that may have been created or saved in the problem object is replaced.

Example

```
status = CPXNETreadcopybase (env, net, "netbasis.bas");
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
filename_str	Name of the basis file to read.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetbranchcallbackfunc

```
void CPXgetbranchcallbackfunc(CPXCENVptr env, int(CPXPUBLIC
**branchcallback_p)(CALLBACK_BRANCH_ARGS), void ** cbhandle_p)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetbranchcallbackfunc` accesses the user-written callback routine to be called during MIP optimization after a branch has been selected but before the branch is carried out. CPLEX uses the callback routine to change its branch selection.

Example

```
CPXgetbranchcallbackfunc(env, &current_callback,
                         &current_handle);
```

See also *Advanced MIP Control Interface* in the *CPLEX User's Manual*.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`branchcallback_p`

The address of the pointer to the current user-written branch callback. If no callback has been set, the returned pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the user's private pointer.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle,
              int        type,
              int        sos,
              int        nodecnt,
              int        bdcnt,
              double     *nodeest,
              int        *nodebeg,
              int        *indices,
              char       *lu,
              int        *bd,
              int        *useraction_p);
```

The call to the branch callback occurs after a branch has been selected but before the branch is carried out. This function is written by the user. On entry to the callback, the CPLEX-selected branch is defined in the arguments. The arguments to the callback specify a list of changes to make to the bounds of variables when child nodes are created. One, two, or zero child nodes can be created, so one, two, or zero lists of changes are specified in the arguments. The first branch specified is considered first. The callback is called with zero lists of bound changes

when the solution at the node is integer feasible.

Custom branching strategies can be implemented by calling the CPLEX function `CPXbranchcallbackbranchbds` and setting the `useraction` variable to `CPX_CALLBACK_SET`. Then CPLEX will carry out these branches instead of the CPLEX-selected branches.

Branch variables are in terms of the original problem if the parameter `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, branch variables are in terms of the presolved problem.

Callback return value

The callback returns zero if successful and nonzero if an error occurs.

Callback arguments

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`cbdata`

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

`wherefrom`

An integer value reporting where in the optimization this function was called. It will have the value `CPX_CALLBACK_MIP_BRANCH`.

`cbhandle`

A pointer to user-private data.

`int type`

An integer that specifies the type of branch. This table summarizes possible values.

Branch Types Returned from a User-Written Branch Callback

Symbolic Constant	Value	Branch
`CPX_TYPE_VAR`	`0`	variable branch
`CPX_TYPE_SOS1`	`1`	SOS1 branch
`CPX_TYPE_SOS2`	`2`	SOS2 branch
`CPX_TYPE_USER`	`X`	user-defined

`sos`

An integer that specifies the special ordered set (SOS) used for this branch. A value of `-1` specifies that this branch is not an SOS-type branch.

`nodecnt`

An integer that specifies the number of nodes CPLEX will create from this branch. Possible values are:

- `0` (zero), or
- `1`, or
- `2`.

If the argument is `0`, the node will be fathomed unless user-specified branches are made; that is, no child nodes are created and the node itself is discarded.

`bdcnt`

An integer that specifies the number of bound changes defined in the arrays `indices`, `lu`, and `bd` that define the CPLEX-selected branch.

`nodeest`

An array with `nodecnt` entries that contains estimates of the integer objective-function value that will be attained from the created node.

`nodebeg`

An array with `nodecnt` entries. The i-th entry is the index into the arrays `indices`, `lu`, and `bd` of the first bound changed for the ith node.

`indices`

Together with `lu` and `bd`, this array defines the bound changes for each of the created nodes. The entry `indices[i]` is the index for the variable.

`lu`

Together with `indices` and `bd`, this array defines the bound changes for each of the created nodes. The entry `lu[i]` is one of the three possible values specifying which bound to change:

- `L` for lower bound, or
- `U` for upper bound, or
- `B` for both bounds.

`bd`

Together with `indices` and `lu`, this array defines the bound changes for each of the created nodes. The entry `bd[i]` specifies the new value of the bound.

`useraction_p`

A pointer to an integer specifying the action for CPLEX to take at the completion of the user callback. The table summarizes the possible actions.

Actions to be Taken After a User-Written Branch Callback

Value	Symbolic Constant	Action
`0`	`CPX_CALLBACK_DEFAULT`	Use CPLEX-selected branch
`1`	`CPX_CALLBACK_FAIL`	Exit optimization
`2`	`CPX_CALLBACK_SET`	Use user-selected branch, as defined by calls to `CPXbranchcallbackbranchbds`
`3`	`CPX_CALLBACK_NO_SPACE`	Allocate more space and call callback again

See Also: CPXsetbranchcallbackfunc

Returns:

This routine does not return a result.

Global function CPXaddfpdest

`int` **`CPXaddfpdest`**`(CPXCENVptr env, CPXCHANNELptr channel, CPXFILEptr fileptr)`

Definition file: cplex.h

The routine `CPXaddfpdest` adds a file to the list of message destinations for a channel. The destination list for all CPLEX-defined channels is initially empty.

Example

```
CPXaddfpdest (env, mychannel, fileptr);
```

See `lpex5.c` in the *CPLEX User's Manual*.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
channel	A pointer to the channel for which destinations are to be added.
fileptr	A pointer to the file to be added to the destination list. Before calling this routine, obtain this pointer with a call to `CPXfopen`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetnodeleftcnt

`int` **`CPXgetnodeleftcnt`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetnodeleftcnt` accesses the number of unexplored nodes left in the branch-and-cut tree.

Example

```
nodes_left = CPXgetnodeleftcnt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If no solution, problem, or environment exists, `CPXgetnodeleftcnt` returns 0 (zero); otherwise, `CPXgetnodeleftcnt` returns the number of unexplored nodes left in the branch-and-cut tree.

Global function CPXfltwrite

`int` **`CPXfltwrite`**`(CPXCENVptr env, CPXCLPptr lp, const char * filename_str)`

Definition file: cplex.h

The routine `CPXfltwrite` writes filters from the selected problem object to a file in FLT format. This format is documented in the *CPLEX File Formats Reference Manual*.

See Also: CPXreadcopysolnpoolfilters

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the filters should be written.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetcallbacknodeobjval

`int` **`CPXgetcallbacknodeobjval`**`(CPXCENVptr env, void * cbdata, int wherefrom, double * objval_p)`

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbacknodeobjval` retrieves the objective value for the subproblem at the current node during MIP optimization from within a user-written callback.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`, or
- `CPX_CALLBACK_MIP_CUT`.

Example

```
status = CPXgetcallbacknodeobjval (env, cbdata, wherefrom,
                                   &objval);
```

See also `admipex1.c` and `admipex3.c` in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting from where the user-written callback was called. The argument must be the value of `wherefrom` passed to the user-written callback.
objval_p	A pointer to a variable of type `double` where the objective value of the node subproblem is to be stored.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetrowinfeas

```
int CPXgetrowinfeas(CPXCENVptr env, CPXCLPptr lp, const double * x, double *
infeasout, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetrowinfeas` computes the infeasibility of a given solution for a range of linear constraints.

The beginning and end of the range must be specified.

For each constraint, the infeasibility value returned is 0 (zero) if the constraint is satisfied. Otherwise, except for ranged rows, the infeasibility value returned is the amount that makes the queried solution valid when added to the righthand side of the constraint. It is positive for a less-than-or-equal-to constraint, negative for a greater-than-or-equal-to constraint.

For an equality constraint, it is positive when the row activity exceeds the right hand side, and negative when the row activity is less than the right hand side.

For ranged rows, if the infeasibility value is negative, it specifies the amount by which the lower bound of the range must be changed; if it is positive, it specifies the amount by which the upper bound of the range must be changed.

Example

```
status = CPXgetrowinfeas (env, lp, NULL, infeasout, 0, CPXgetnumrows(env,lp)-1);
```

return The routine returns zero if successful and nonzero if an error occurs.
Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
x	The solution whose infeasibility is to be computed. May be NULL, in which case the resident solution is used.
infeasout	An array to receive the infeasibility value for each of the constraints. This array must be of length at least (`end - begin + 1`).
begin	An integer specifying the beginning of the range of linear constraints whose infeasibility is to be returned.
end	An integer specifying the end of the range of linear constraints whose infeasibility is to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXfopen

```
CPXFILEptr CPXfopen(const char * filename_str, const char * type_str)
```

Definition file: cplex.h

The routine `CPXfopen` opens files to be used in conjunction with the routines `CPXaddfpdest`, `CPXdelfpdest` and `CPXsetlogfile`. It has the same arguments as the standard C library function `fopen`.

Example

```
fp = CPXfopen ("mylog.log", "w");
```

See also `lpex5.c` in the *CPLEX User's Manual*.

Parameters:

filename_str	A pointer to a character string that contains the name of the file to be opened.
type_str	A pointer to a character string, containing characters according to the syntax of the standard C function `fopen`.

Returns:

The routine returns a pointer to an object representing an open file, or NULL if the file could not be opened. A CPXFILEptr is analogous to FILE *type in C language.

Global function CPXfreeusercuts

`int` **`CPXfreeusercuts`**`(CPXCENVptr env, CPXLPptr lp)`

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXfreeusercuts` clears the list of user cuts that have been previously specified through calls to `CPXaddusercuts`.

Example

```
status = CPXfreeusercuts (env, lp);
```

Parameters:

env A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETgetbase

`int` **`CPXNETgetbase`**`(CPXCENVptr env, CPXCNETptr net, int * astat, int * nstat)`

Definition file: cplex.h

The routine `CPXNETgetbase` is used to access the network basis for a network problem object. Either of the arguments `astat` or `nstat` may be NULL.

For this function to succeed, a solution must exist for the problem object.

Table 1: Status codes of network arcs

`CPX_BASIC`	If the arc is basic.
`CPX_AT_LOWER`	If the arc is nonbasic and its flow is on the lower bound.
`CPX_AT_UPPER`	If the arc is nonbasic and its flow is on the upper bound.
`CPX_FREE_SUPER`	If the arc is nonbasic but is free.In this case its flow is 0.

Table 2: Status of artificial arcs

`CPX_BASIC`	If the arc is basic.
`CPX_AT_LOWER`	If the arc is nonbasic and its flow is on the lower bound.

Example

```
status = CPXNETgetbase (env, net, astat, nstat);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

net A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

astat An array in which the statuses for network arcs are to be written. After termination, `astat[i]` contains the status assigned to arc `i` of the network stored in `net`. The status may be one of the values in Table 1. If NULL is passed, no arc statuses are copied. Otherwise, `astat` must be an array of a size that is at least `CPXNETgetnumarcs`.

nstat An array in which the statuses for artificial arcs from each node to the root node are to be written. After termination, `nstat[i]` contains the status assigned to the artificial arc from node `i` to the root node of the network stored in `net`. The status may be one of values in Table 2. If NULL is passed, no node statuses are copied. Otherwise, `nstat` must be an array of a size that is at least `CPXNETgetnumnodes`.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXdelqconstrs

`int` **`CPXdelqconstrs`**`(CPXCENVptr env, CPXLPptr lp, int begin, int end)`

Definition file: cplex.h

The routine `CPXdelqconstrs` deletes a range of quadratic constraints. The range is specified by a lower and upper index that represent the first and last quadratic constraints to be deleted, respectively. The indices of the constraints following those deleted are decreased by the number of deleted constraints.

Example

```
status = CPXdelqconstrs (env, lp, 10, 20);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
begin	An integer that indicates the numeric index of the first quadratic constraint to be deleted.
end	An integer that indicates the numeric index of the last quadratic constraint to be deleted.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXdelsetsolnpoolfilters

`int` **`CPXdelsetsolnpoolfilters`**`(CPXCENVptr env, CPXLPptr lp, int * delstat)`

Definition file: cplex.h

The routine `CPXdelsetsolnpoolfilters` deletes filters from the problem object specified by the argument `lp`. Unlike the routine `CPXdelsolnpoolfilters`, `CPXdelsetsolnpoolfilters` does not require the filters to be in a contiguous range. After the deletion occurs, the remaining filters are indexed consecutively starting at 0, and in the same order as before the deletion.

Note

The `delstat` array must have at least `CPXgetsolnpoolnumfilters(env,lp)` elements.

Example

```
status = CPXdelsetsolnpoolfilters (env, lp, delstat);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
delstat	An array specifying the filters to be deleted. The routine `CPXdelsetfilters` deletes each filter `i` for which `delstat[i] = 1`. The deletion of filters results in a renumbering of the remaining filters. After termination, `delstat[i]` is either -1 for filters that have been deleted or the new index number that has been assigned to the remaining filters.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETgetslack

`int` **`CPXNETgetslack`**`(CPXCENVptr env, CPXCNETptr net, double * slack, int begin, int end)`

Definition file: cplex.h

The routine `CPXNETgetslack` is used to access slack values or, equivalently, violations of supplies/demands for a range of nodes in the network stored in a network problem object.

For this function to succeed, a solution must exist for the problem object.

Example

```
status = CPXNETgetslack (env, net, slack, 10, 20);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
slack	Array in which to write solution slack variables for requested nodes. If NULL is passed, no data is returned. Otherwise, `slack` must point to an array of size at least (`end-begin+1`).
begin	Index of the first node for which a slack value is to be obtained.
end	Index of the last node for which a slack value is to be obtained.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetray

```
int CPXgetray(CPXCENVptr env, CPXCLPptr lp, double * z)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetray` finds an unbounded direction (also known as a ray) for a linear program where the CPLEX simplex optimizer concludes that the LP is unbounded (solution status `CPX_STAT_UNBOUNDED`). An error is returned, `CPXERR_NOT_UNBOUNDED`, if this case does not hold.

As an illustration, consider a linear program of the form:

```
Minimize        c'x
Subject to      Ax = b
                x >= 0
```

where ' specifies the transpose.

If the CPLEX simplex algorithm completes optimization with a solution status of `CPX_STAT_UNBOUNDED`, the vector `z` returned by `CPXgetray` would satisfy the following:

```
c'z < 0
Az = 0
z >= 0
```

if computations could be carried out in exact arithmetic.

Example

```
status = CPXgetray (env, lp, z);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX LP problem object, as returned by `CPXcreateprob`.
z	The array where the unbounded direction is returned. This array must be at least as large as the number of columns in the problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetcrossppushcnt

```
int CPXgetcrossppushcnt(CPXCENVptr env, CPXCLPptr lp)
```

Definition file: cplex.h

The routine `CPXgetcrossppushcnt` accesses the number of primal push iterations in the crossover method. A push occurs when a nonbasic variable switches bounds and does not enter the basis.

Example

```
itcnt = CPXgetcrossppushcnt (env, lp);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns the primal push iteration count if a solution exists. If no solution exists, it returns zero.

Global function CPXrhssa

```
int CPXrhssa(CPXCENVptr env, CPXCLPptr lp, int begin, int end, double * lower,
double * upper)
```

Definition file: cplex.h

The routine `CPXrhssa` accesses upper and lower sensitivity ranges for righthand side values of a range of constraints. The beginning and end of the range of constraints must be specified.

Note

Information for constraint `j`, where `begin` <= `j` <= `end`, is returned in position `(j-begin)` of the arrays `lower` and `upper`. The items `lower[j-begin]` and `upper[j-begin]` contain the lowest and highest value the righthand side value of constraint `j` can assume without affecting the optimality of the solution.

Example

```
status = CPXrhssa (env, lp, 0, CPXgetnumrows(env,lp)-1,
                   lower, upper);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
begin	An integer specifying the beginning of the range of ranges to be returned.
end	An integer specifying the end of the range of ranges to be returned.
lower	An array where the righthand side lower range values are to be returned. This array must be of length at least `(end - begin + 1)`.
upper	An array where the righthand side upper range values are to be returned. This array must be of length at least `(end - begin + 1)`.

Returns:

The routine returns zero if successful and nonzero if an error occurs. This routine fails if no optimal basis exists.

Global function CPXgetintquality

`int` **`CPXgetintquality`**`(CPXCENVptr env, CPXCLPptr lp, int * quality_p, int what)`

Definition file: cplex.h

The routine `CPXgetintquality` accesses integer-valued information about the quality of the current solution of a problem. A solution, though not necessarily a feasible or optimal one, must be available in the CPLEX problem object. The quality values are returned in the `int` variable pointed to by the argument `quality_p`.

Example

```
status = CPXgetintquality (env, lp, &max_x_ind, CPX_MAX_X);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
quality_p	A pointer to an integer variable in which the requested quality value is to be stored.
what	A symbolic constant specifying the quality value to be retrieved. The possible quality values for a solution are listed in the group optim.cplex.solutionquality in the *Callable Library Reference Manual*.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXaddrows

```
int CPXaddrows(CPXCENVptr env, CPXLPptr lp, int ccnt, int rcnt, int nzcnt, const
double * rhs, const char * sense, const int * rmatbeg, const int * rmatind, const
double * rmatval, char ** colname, char ** rowname)
```

Definition file: cplex.h

The routine `CPXaddrows` adds constraints to a specified CPLEX problem object. This routine may be called any time after a call to `CPXcreateprob`.

When you add a ranged row, `CPXaddrows` sets the corresponding range value to 0 (zero). Use the routine `CPXchgrngval` to change the range value.

Values of sense

`sense[i]`	`= 'L'`	<= constraint
`sense[i]`	`= 'E'`	= constraint
`sense[i]`	`= 'G'`	>= constraint
`sense[i]`	`= 'R'`	ranged constraint

When you build or modify your problem with this routine, you can verify that the results are as you intended by calling `CPXcheckaddrows` during application development.

Note

The use of `CPXaddrows` as a way to add new columns is discouraged in favor of a direct call to `CPXnewcols` before calling `CPXaddrows`.

Example

```
status = CPXaddrows (env, lp, ccnt, rcnt, nzcnt, rhs,
                     sense, rmatbeg, rmatind, rmatval,
                     newcolname, newrowname);
```

See also the example `lpex3.c` in the *CPLEX User's Manual* and in the standard distribution.

For more about the conventions for representing a matrix as compact arrays, see the discussion of matbeg, matind, and matval in the routine `CPXcopylp`.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
ccnt	An integer that specifies the number of new columns in the constraints being added to the constraint matrix. When new columns are added, they are given an objective coefficient of zero, a lower bound of zero, and an upper bound of `CPX_INFBOUND`.
rcnt	An integer that specifies the number of new rows to be added to the constraint matrix.
nzcnt	An integer that specifies the number of nonzero constraint coefficients to be added to the constraint matrix. This specifies the length of the arrays `rmatind` and `rmatval`.
rhs	An array of length `rcnt` containing the righthand side term for each constraint to be added to the CPLEX problem object. May be NULL, in which case the new righthand side values are set to 0.0.
sense	An array of length `rcnt` containing the sense of each constraint to be added to the CPLEX problem object. May be NULL, in which case the new constraints are created as equality constraints. Possible values of this argument appear in the table.
rmatbeg	An array used with `rmatind` and `rmatval` to define the rows to be added.

rmatind An array used with `rmatbeg` and `rmatval` to define the rows to be added.

rmatval An array used with `rmatbeg` and `rmatind` to define the rows to be added. The format is similar to the format used to describe the constraint matrix in the routine `CPXcopylp` (see description of `matbeg`, `matcnt`, `matind`, and `matval` in that routine), but the nonzero coefficients are grouped by row instead of column in the array `rmatval`. The nonzero elements of every row must be stored in sequential locations in this array from position `rmatbeg[i]` to `rmatbeg[i+1]-1` (or from `rmatbeg[i]` to `nzcnt -1` if `i=rcnt-1`). Each entry, `rmatind[i]`, specifies the column index of the corresponding coefficient, `rmatval[i]`. Unlike `CPXcopylp`, all rows must be contiguous, and `rmatbeg[0]` must be 0 (zero).

colname An array of length `ccnt` containing pointers to character strings that represent the names of the new columns added to the CPLEX problem object, or equivalently, the new variable names. May be NULL, in which case the new columns are assigned default names if the columns already resident in the CPLEX problem object have names; otherwise, no names are associated with the variables. If column names are passed to `CPXaddrows` but existing variables have no names assigned, default names are created for them.

rowname An array containing pointers to character strings that represent the names of the new rows, or equivalently, the constraint names. May be NULL, in which case the new rows are assigned default names if the rows already resident in the CPLEX problem object have names; otherwise, no names are associated with the constraints. If row names are passed to `CPXaddrows` but existing constraints have no names assigned, default names are created for them.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetcallbackpseudocosts

```
int CPXgetcallbackpseudocosts(CPXCENVptr env, void * cbdata, int wherefrom, double
* uppc, double * downpc, int begin, int end)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbackpseudocosts` retrieves the pseudo-cost values during MIP optimization from within a user-written callback. The values are from the original problem if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`. Otherwise, they are from the presolved problem.

Note

When pseudo-costs are retrieved for the original problem variables, pseudo-costs are zero for variables that have been removed from the problem, since they are never used for branching.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

Example

```
status = CPXgetcallbackpseudocosts (env, cbdata, wherefrom,
                                    upcost, downcost,
                                    j, k);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of cbdata passed to the user-written callback.
wherefrom	An integer value reporting from where the user-written callback was called. The argument must be the value of `wherefrom` passed to the user-written callback.
uppc	An array to receive the values of up pseudo-costs. This array must be of length at least (`end - begin + 1`). If successful, `uppc[0]` through `uppc[end-begin]` will contain the up pseudo-costs. May be NULL.
downpc	An array to receive the values of the down pseudo-costs. This array must be of length at least (`end - begin + 1`). If successful, `downpc[0]` through `downpc[end-begin]` will contain the down pseudo-costs. May be NULL.
begin	An integer specifying the beginning of the range of pseudo-costs to be returned.
end	An integer specifying the end of the range of pseudo-costs to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXdelnames

```
int CPXdelnames(CPXCENVptr env, CPXLPptr lp)
```

Definition file: cplex.h

The routine `CPXdelnames` removes all names that have been previously assigned to rows and columns. The memory that was used by those names is released.

Names can be assigned to rows and columns in a variety of ways, and this routine allows them to be removed. For example, if the problem is read from a file in LP or MPS format, names are also read from the file. Names can be assigned by the user by calling one of the routines `CPXchgrowname`, `CPXchgcolname`, or `CPXchgname`.

Example

```
CPXdelnames (env, lp);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXdperwrite

```
int CPXdperwrite(CPXCENVptr env, CPXLPptr lp, const char * filename_str, double
epsilon)
```

Definition file: cplex.h

When solving degenerate linear programs with the dual simplex method, CPLEX may initiate a perturbation of the objective function of the problem in order to improve performance. The routine `CPXdperwrite` writes a similarly perturbed problem to a binary SAV format file.

Example

```
status = CPXdperwrite (env, lp, "myprob.dpe", epsilon);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the perturbed LP problem should be written.
epsilon	The perturbation constant.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetobj

```
int CPXgetobj(CPXCENVptr env, CPXCLPptr lp, double * obj, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetobj` accesses a range of objective function coefficients of a CPLEX problem object. The beginning and end of the range must be specified.

Example

```
status = CPXgetobj (env, lp, obj, 0, cur_numcols-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
obj	An array where the specified objective coefficients are to be returned. This array must be of length at least (`end - begin + 1`). The objective function coefficient of variable `j` is returned in `obj[j - begin]`.
begin	An integer specifying the beginning of the range of objective function coefficients to be returned.
end	An integer specifying the end of the range of objective function coefficients to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcopyqpsep

`int` **`CPXcopyqpsep`**`(CPXCENVptr env, CPXLPptr lp, const double * qsepvec)`

Definition file: cplex.h

The routine `CPXcopyqpsep` is used to copy the quadratic objective matrix Q for a separable QP problem. A separable QP problem is one where the coefficients of Q have no nonzero off-diagonal elements.

> **Note**
>
> CPLEX evaluates the corresponding objective with a factor of `0.5` in front of the quadratic objective term.

When you build or modify your model with this routine, you can verify that the results are as you intended by calling `CPXcheckcopyqpsep` during application development.

Example

```
status = CPXcopyqpsep (env, lp, qsepvec);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
qsepvec	An array of length `CPXgetnumcols(env,lp)`.`qsepvec[0], qsepvec[1]`,..., `qsepvec[numcols-1]` should contain the quadratic coefficients of the separable quadratic objective.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXNETchgsupply

`int` **`CPXNETchgsupply`**`(CPXCENVptr env, CPXNETptr net, int cnt, const int * indices, const double * supply)`

Definition file: cplex.h

The routine `CPXNETchgsupply` is used to change supply values for a set of nodes in the network stored in a network problem object.

Any solution information stored in the problem object is lost.

Example

```
status = CPXNETchgsupply (env, net, cnt, indices, supply);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
cnt	An integer indicating the number of nodes for which the supply values are to be changed.
indices	An array of indices that indicate the nodes for which the supply values are to be changed. This array must have a length of at least `cnt`. The indices must be in the range [`0, nnodes-1`].
supply	An array that contains the new supply values. This array must have a length of at least `cnt`.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXsetdeletenodecallbackfunc

```
int CPXsetdeletenodecallbackfunc(CPXENVptr env, void(CPXPUBLIC
*deletecallback)(CALLBACK_DELETENODE_ARGS), void * cbhandle)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXsetdeletenodecallbackfunc` sets and modifies the user-written callback to be called during MIP optimization when a node is to be deleted. Nodes are deleted in these circumstances:

- when a branch is carried out from that node, or
- when the node relaxation is infeasible, or
- when the node relaxation objective value is worse than the cutoff.

Example

```
status = CPXsetdeletenodecallbackfunc (env,
                                       mybranchfunc,
                                       mydata);
```

See also the example `admipex1.c` in the standard distribution.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`deletecallback`

A pointer to a user-written branch callback. If the callback is set to NULL, no callback is called during optimization.

`cbhandle`

A pointer to user private data. This pointer is passed to the callback.

Callback description

```
int callback (CPXCENVptr env,
              void      *cbdata,
              int       wherefrom,
              void      *cbhandle,
              int       seqnum,
              void      *handle);
```

The call to the delete node callback routine occurs during MIP optimization when a node is to be deleted.

The main purpose of the callback is to provide an opportunity to free any user data associated with the node, thus preventing memory leaks.

Callback return value

The callback returns zero if successful and nonzero if an error occurs.

Callback arguments

`env`

A pointer to the CPLEX environment, as returned by one of the `CPXopenCPLEX` routines.

`cbdata`

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

`wherefrom`

An integer value reporting where in the optimization this function was called. It will have the value `CPX_CALLBACK_MIP_DELETENODE`.

`cbhandle`

A pointer to user private data.

`seqnum`

The sequence number of the node that is being deleted.

`handle`

A pointer to the user private data that was assigned to the node when it was created with one of the callback branching routines:

- `CPXbranchcallbackbranchbds`, or
- `CPXbranchcallbackbranchconstraints`, or
- `CPXbranchcallbackbranchgeneral`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETwriteprob

```
int CPXNETwriteprob(CPXCENVptr env, CPXCNETptr net, const char * filename_str,
const char * format_str)
```

Definition file: cplex.h

The routine `CPXNETwriteprob` writes the network stored in a network problem object to a file. This can be done in CPLEX (`.net`) or DIMACS (`.min`) network file format or as the LP representation of the network in any of the LP formats (`.lp`, `.mps`, or `.sav`).

If the file name ends with `.gz`, a compressed file is written.

File extensions for network files

net	for CPLEX network format
min	for DIMACS network format
lp	for LP format of LP formulation
mps	for MPS format of LP formulation
sav	for SAV format of LP formulation

Example

```
status = CPXNETwriteprob (env, net, "network.net", NULL);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
filename_str	Name of the network file to write, where the file extension specifies the file format unless overridden by the `format` argument. If the file name ends with `.gz` a compressed file is written in accordance with the selected file type.
format_str	File format to generate. Possible values appear in the table. If NULL is passed, the format is inferred from the file name.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetcutcallbackfunc

```
void CPXgetcutcallbackfunc(CPXCENVptr env, int(CPXPUBLIC
**cutcallback_p)(CALLBACK_CUT_ARGS), void ** cbhandle_p)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcutcallbackfunc` accesses the user-written callback for adding cuts. The user-written callback is called by CPLEX during MIP branch and cut for every node that has an LP optimal solution with objective value below the cutoff and that is integer infeasible. CPLEX also calls the callback when comparing an integer feasible solution, including one provided by a MIP start before any nodes exist, against lazy constraints. The callback routine adds globally valid cuts to the LP subproblem.

Example

```
CPXgetcutcallbackfunc(env, &current_cutfunc, &current_data);
```

See also *Advanced MIP Control Interface* in the *CPLEX User's Manual*.

For documentation of callback arguments, see the routine `CPXsetcutcallbackfunc`.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`cutcallback_p`

The address of the pointer to the current user-written cut callback. If no callback has been set, the pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the user's private pointer.

See Also: CPXcutcallbackadd, CPXsetcutcallbackfunc

Returns:

This routine does not return a result.

Global function CPXgetnumint

`int` **`CPXgetnumint`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetnumint` accesses the number of general integer variables in a CPLEX problem object.

Example

```
numint = CPXgetnumint (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Example

```
numint = CPXgetnumint (env, lp);
```

Returns:

If the problem object or environment does not exist, CPXgetnumint returns zero. Otherwise, it returns the number of general integer variables in the problem object.

Global function CPXwritemipstarts

`int` **`CPXwritemipstarts`**`(CPXCENVptr env, CPXCLPptr lp, const char * filename_str, int begin, int end)`

Definition file: cplex.h

The routine `CPXwritemipstarts` writes a range of MIP starts of a CPLEX problem object to a file in MST format.

The MST format is an XML format and is documented in the stylesheet `solution.xsl` and schema `solution.xsd` in the `include` directory of the CPLEX distribution. *CPLEX File Formats Reference Manual* also documents this format briefly.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the MIP start information should be written.
begin	An integer specifying the beginning of the range of MIP starts to be written.
end	An integer specifying the end of the range of MIP starts to be written.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetnumnz

`int` **`CPXgetnumnz`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetnumnz` accesses the number of nonzero elements in the constraint matrix of a CPLEX problem object, not including the objective function, quadratic constraints, or the bounds constraints on the variables.

Example

```
cur_numnz = CPXgetnumnz (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If the problem object or environment does not exist, `CPXgetnumnz` returns the value 0 (zero); otherwise, it returns the number of nonzero elements.

Global function CPXsetnodecallbackfunc

```
int CPXsetnodecallbackfunc(CPXENVptr env, int(CPXPUBLIC
*nodecallback)(CALLBACK_NODE_ARGS), void * cbhandle)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXsetnodecallbackfunc` sets and modifies the user-written callback to be called during MIP optimization after CPLEX has selected a node to explore, but before this exploration is carried out. The callback routine can change the node selected by CPLEX to a node selected by the user.

Example

```
status = CPXgetnodecallbackfunc(env, mynodefunc, mydata);
```

See also the example `admipex1.c` in the standard distribution.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`nodecallback`

A pointer to the current user-written node callback. If no callback has been set, the pointer evaluates to NULL.

`cbhandle`

A pointer to user private data. This pointer is passed to the user-written node callback.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle,
              int        *nodeindex_p,
              int        *useraction_p);
```

CPLEX calls the node callback after selecting the next node to explore. The user can choose another node by setting the argument values of the callback.

Callback return value

The callback returns zero if successful and nonzero if an error occurs.

Callback arguments

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`cbdata`

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

`wherefrom`

An integer value reporting where in the optimization this function was called. It has the value `CPX_CALLBACK_MIP_NODE`.

`cbhandle`

A pointer to user private data.

`nodeindex_p`

A pointer to an integer that specifies the node number of the user-selected node. The node selected by CPLEX is node number 0 (zero). Other nodes are numbered relative to their position in the tree, and this number changes with each tree operation. The unchanging identifier for a node is its sequence number. To access the sequence number of a node, use the routine `CPXgetcallbacknodeinfo`. An error results if a user attempts to select a node that has been moved to a node file. (See the *CPLEX User's Manual* for more information about node files.)

`useraction_p`

A pointer to an integer specifying the action to be taken on completion of the user callback. The table summarizes the possible actions.

Actions to be Taken after a User-Written Node Callback

Value	Symbolic Constant	Action
0	`CPX_CALLBACK_DEFAULT`	Use CPLEX-selected node
1	`CPX_CALLBACK_FAIL`	Exit optimization
2	`CPX_CALLBACK_SET`	Use user-selected node as defined in returned values

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsolnpoolnumfilters

`int` **`CPXgetsolnpoolnumfilters`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetsolnpoolnumfilters` accesses the number of filters in the solution pool.

Example

```
numfilters = CPXgetsolnpoolnumfilters (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If the CPLEX problem object or environment does not exist, `CPXgetsolnpoolnumfilters` returns the value 0 (zero); otherwise, it returns the number of filters.

Global function CPXgetcallbacknodeinfo

```
int CPXgetcallbacknodeinfo(CPXCENVptr env, void * cbdata, int wherefrom, int
nodeindex, int whichinfo, void * result_p)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbacknodeinfo` is called from within user-written callbacks during a MIP optimization and accesses information about nodes. The node information is from the original problem if the parameter `CPX_PARAM_MIPCBREDLP` is turned off (set to `CPX_OFF`). Otherwise, the information is from the presolved problem.

The primary purpose of this routine is to examine nodes in order to select one from which to proceed. In this case, the `wherefrom` argument is `CPX_CALLBACK_MIP_NODE`, and a node with any `nodeindex` value can be queried. A secondary purpose of this routine is to obtain information about the current node. When the `wherefrom` argument is any one of the following values, only the current node can be queried.

- `CPX_CALLBACK_MIP_CUT`
- `CPX_CALLBACK_MIP_INCUMBENT`
- `CPX_CALLBACK_MIP_HEURISTIC`
- `CPX_CALLBACK_MIP_SOLVE`
- `CPX_CALLBACK_MIP_BRANCH`

To query the current node, specify a `nodeindex` value of 0. Other values of the `wherefrom` argument are invalid for this routine. An invalid `nodeindex` value or `wherefrom` argument value will result in an error return value.

Note

The values returned for `CPX_CALLBACK_INFO_NODE_SIINF` and `CPX_CALLBACK_INFO_NODE_NIINF` for the current node are the values that applied to the node when it was stored and thus before the branch was solved. As a result, these values should not be used to assess the feasibility of the node. Instead, use the routine `CPXgetcallbacknodeintfeas` to check the feasiblity of a node.

This routine cannot retrieve information about nodes that have been moved to node files. For more information about node files, see the *CPLEX User's Manual*. If the argument `nodeindex` refers to a node in a node file, `CPXgetcallbacknodeinfo` returns the value `CPXERR_NODE_ON_DISK`. Nodes still in memory have the lowest index numbers so a user can loop through the nodes until `CPXgetcallbacknodeinfo` returns an error, and then exit the loop.

Example

```
status = CPXgetcallbacknodeinfo(env,
                                cbdata,
                                wherefrom,
                                0,
                                CPX_CALLBACK_INFO_NODE_NIINF,
                                &numiinf);
```

Table 1: Information Requested for a User-Written Node Callback

Symbolic Constant	C Type	Meaning

CPX_CALLBACK_INFO_NODE_SIINF	double	sum of integer infeasibilities
CPX_CALLBACK_INFO_NODE_NIINF	int	number of integer infeasibilities
CPX_CALLBACK_INFO_NODE_ESTIMATE	double	estimated integer objective
CPX_CALLBACK_INFO_NODE_DEPTH	int	depth of node in branch-and-cut tree
CPX_CALLBACK_INFO_NODE_OBJVAL	double	objective value of LP subproblem
CPX_CALLBACK_INFO_NODE_TYPE	char	type of branch at this node; see Table 2
CPX_CALLBACK_INFO_NODE_VAR	int	for nodes of type CPX_TYPE_VAR, the branching variable for this node; for other types, -1 is returned
CPX_CALLBACK_INFO_NODE_SOS	int	for nodes of type CPX_TYPE_SOS1 or CPX_TYPE_SOS2 the number of the SOS used in branching; -1 otherwise
CPX_CALLBACK_INFO_NODE_SEQNUM	int	sequence number of the node
CPX_CALLBACK_INFO_NODE_USERHANDLE	void	userhandle associated with the node upon its creation
CPX_CALLBACK_INFO_NODE_NODENUM	int	node index of the node (only available for CPXgetcallbackseqinfo)

Table 2: Branch Types Returned when whichinfo = CPX_CALLBACK_INFO_NODE_TYPE

Symbolic Constant	Value	Branch Type
CPX_TYPE_VAR	'0'	variable branch
CPX_TYPE_SOS1	'1'	SOS1 branch
CPX_TYPE_SOS2	'2'	SOS2 branch
CPX_TYPE_USER	'X'	user-defined
CPX_TYPE_ANY	'A'	multiple bound changes and/or constraints were used for branching

See also *Advanced MIP Control Interface* in the *CPLEX User's Manual*.

See Also: CPXgetcallbackinfo, CPXgetcallbackseqinfo

Parameters:

env	A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
cbdata	The pointer passed to the user-written callback. This argument must be the value of cbdata passed to the user-written callback.
wherefrom	An integer value reporting where the user-written callback was called from. This argument must be the value of wherefrom passed to the user-written callback.
nodeindex	The index of the node for which information is requested. Nodes are indexed from 0 (zero) to (nodecount - 1) where nodecount is obtained from the callback information function CPXgetcallbackinfo, with a whichinfo value of CPX_CALLBACK_INFO_NODES_LEFT.
whichinfo	An integer specifying which information is requested. Table 1 summarizes the possible values. Table 2 summarizes possible values returned when the type of information requested is branch type (that is, whichinfo = CPX_CALLBACK_INFO_NODE_TYPE).
result_p	A generic pointer to a variable of type double or int, representing the value returned by whichinfo. (The column C Type in Table 1 shows the type of various values returned by whichinfo.)

Returns:

The routine returns zero if successful and nonzero if an error occurs. The return value `CPXERR_NODE_ON_DISK` reports an attempt to access a node currently located in a node file on disk.

Global function CPXbinvrow

```
int CPXbinvrow(CPXCENVptr env, CPXCLPptr lp, int i, double * y)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXbinvrow` computes the i-th row of the basis inverse.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX LP problem object, as returned by `CPXcreateprob`.
i	An integer that specifies the index of the row to be computed.
y	An array containing the i-th row of **Binv** (the inverse of the matrix B). The array must be of length at least equal to the number of rows in the problem.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcleanup

`int` **`CPXcleanup`**`(CPXCENVptr env, CPXLPptr lp, double eps)`

Definition file: cplex.h

The routine `CPXcleanup` changes to zero any problem coefficients that are smaller in magnitude than the tolerance specified in the argument `eps`.

This routine may be called at any time after a problem object has been created by a call to `CPXcreateprob`. This practice is also known as *zero-ing out* the negligible coefficients. Such coefficients may arise as round-off errors if the matrix coefficients are computed with floating-point arithmetic.

Example

```
status = CPXcleanup (env, lp, eps);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
eps	A tolerance to determine whether coefficients in the problem are sufficiently small in magnitude to eliminate.

Returns:

The routine returns zero unless an error occurred during the optimization.

Global function CPXgetnodecnt

`int` **`CPXgetnodecnt`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetnodecnt` accesses the number of nodes used to solve a mixed integer problem.

Example

```
nodecount = CPXgetnodecnt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Example

```
nodecount = CPXgetnodecnt (env, lp);
```

Returns:

If a solution exists, CPXgetnodecnt returns the node count. If no solution, problem, or environment exists, CPXgetnodecnt returns the value 0.

Global function CPXgetsolnpoolintquality

```
int CPXgetsolnpoolintquality(CPXCENVptr env, CPXCLPptr lp, int soln, int *
quality_p, int what)
```

Definition file: cplex.h

The routine `CPXgetsolnpoolintquality` accesses integer-valued information about the quality of a solution in the solution pool. The quality values are returned in the `int` variable pointed to by the argument `quality_p`.

Example

```
status = CPXgetsolnpooldblquality (env, lp, soln, &max_x, CPX_MAX_X);
```

Parameters:

env	A pointer to the CPLEX environment as returned by the `CPXopenCPLEX` routine.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
soln	An integer specifying the index of the solution pool member for which the quality measure is to be computed. A value of -1 specifies that the incumbent should be used instead of a member of the solution pool.
quality_p	A pointer to an `int` variable in which the requested quality value is to be stored. If an error occurs, the quality-value remains unchanged.
what	A symbolic constant specifying the quality value to be retrieved. The possible quality values which can be evaluated for a solution pool member are listed in the group optim.cplex.solutionquality in the *Callable Library Reference Manual*.

Returns:

The routine returns zero if successful and nonzero if an error occurs. If an error occurs, the quality-value remains unchanged.

Global function CPXgetsolnpooldblquality

`int` **`CPXgetsolnpooldblquality`**`(CPXCENVptr env, CPXCLPptr lp, int soln, double * quality_p, int what)`

Definition file: cplex.h

The routine `CPXgetsolnpooldblquality` accesses double-valued information about the quality of a solution in the solution pool. The quality values are returned in the `double` variable pointed to by the argument `quality_p`.

Example

```
status = CPXgetsolnpooldblquality (env, lp, soln, CPX_MAX_X, &max_x);
```

Parameters:

env	A pointer to the CPLEX environment as returned by the `CPXopenCPLEX` routine.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
soln	An integer giving the index of the solution pool member for which the quality measure is to be computed. A value of -1 specifies that the incumbent should be used instead of a member of the solution pool.
quality_p	A pointer to a `double` variable in which the requested quality value is to be stored. If an error occurs, the quality-value remains unchanged.
what	A symbolic constant specifying the quality value to be retrieved. The possible quality values for a solution are listed in the group optim.cplex.solutionquality in the *Callable Library Reference Manual*.

Returns:

The routine returns zero if successful and nonzero if an error occurs. If an error occurs, the quality-value remains unchanged.

Global function CPXgetsolnpoolsolnname

```
int CPXgetsolnpoolsolnname(CPXCENVptr env, CPXCLPptr lp, char * store, int storesz,
int * surplus_p, int which)
```

Definition file: cplex.h

The routine `CPXgetsolnpoolsolnname` accesses the name of a solution, specified by the argument `soln`, of the solution pool associated with the problem object specified by the argument `lp`.

If the value of `storesz` is 0 (zero), then the negative of the value of `surplus_p` returned specifies the total number of characters needed for the array `store`.

A nonnegative value of `surplus_p` specifies that the length of the array `store` was sufficient. A negative value specifies that the length of the array was insufficient and that the routine could not complete its task. In this case, `CPXgetsolnpoolsolnname` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `surplus_p` specifies the amount of insufficient space in the array `store`.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
store	A pointer to a buffer of size `storesz`. It may be NULL if `storesz` is 0 (zero).
storesz	An integer specifying the length of the array `store`. It may be 0 (zero).
surplus_p	A pointer to an integer to contain the difference between `storesz` and the amount of memory required to store the name of the solution.
which	An integer specifying the index of the solution for which the name is returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` specifies that insufficient space was available in the array `store` to hold the name of the solution.

Global function CPXgetstatstring

```
CPXCHARptr CPXgetstatstring(CPXCENVptr env, int statind, char * buffer_str)
```

Definition file: cplex.h

The routine `CPXgetstatstring` places in a buffer, a string corresponding to the value of `statind` as returned by the routine `CPXgetstat`. The buffer to hold the string can be up to 510 characters maximum; the buffer must be at least 56 characters.

Example

```
statind = CPXgetstat (env, lp);
p = CPXgetstatstring (env, statind, buffer);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
statind	An integer specifying the status value to return.
buffer_str	A pointer to a buffer to hold the string corresponding to the value of `statind`.

Returns:

The routine returns a pointer to a buffer if the `statind` value corresponds to a valid string. Otherwise, it returns NULL.

Global function CPXgetqpcoef

`int` **`CPXgetqpcoef`**`(CPXCENVptr env, CPXCLPptr lp, int rownum, int colnum, double * coef_p)`

Definition file: cplex.h

The routine `CPXgetqpcoef` accesses the quadratic coefficient in the matrix Q of a CPLEX problem object for the variable pair indexed by (`rownum`, `colnum`). The result is stored in `*coef_p`.

Example

```
status = CPXgetqpcoef (env, lp, 10, 20, &coef);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
rownum	The first variable number (row number in Q).
colnum	The second variable number (column number in Q).
coef_p	A pointer to a double where the coefficient should be stored.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetcallbacknodelp

`int` **`CPXgetcallbacknodelp`**`(CPXCENVptr env, void * cbdata, int wherefrom, CPXLPptr * nodelp_p)`

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbacknodelp` returns a pointer to the current continuous relaxation at the current branch and cut node from within a user-written callback. Generally, this pointer may be used only in CPLEX Callable Library query routines, such as `CPXsolution` or `CPXgetrows`.

Note that the setting of the parameter `CPX_PARAM_MIPCBREDLP` does not affect this `lp` pointer. Since CPLEX does not explicitly maintain an unpresolved node LP, the `lp` pointer will correspond to the presolved node LP unless CPLEX presolve has been turned off or CPLEX has made no presolve reductions at all.

Example

```
status = CPXgetcallbacknodelp (env, cbdata, wherefrom, &nodelp);
```

See also the example `admipex1.c` and `admipex6.c` in the standard distribution.

`CPXgetcallbacknodelp` may be called only when its `wherefrom` argument has one of the following values:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_CUT`,
- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_INCUMBENT`, or
- `CPX_CALLBACK_MIP_SOLVE`.

When the `wherefrom` argument has the value `CPX_CALLBACK_MIP_SOLVE`, the subproblem pointer may also be used in CPLEX optimization routines.

Note

Any modification to the subproblem may result in corruption of the problem and of the CPLEX environment.

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The `cbdata` pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting where the user-written callback was called from. This argument must be the value of the `wherefrom` passed to the user-written callback.
nodelp_p	The `lp` pointer specifying the current subproblem. If no subproblem is defined, the pointer is set to NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs. A nonzero return value may mean that the requested value is not available.

Global function CPXgetindconstr

```
int CPXgetindconstr(CPXCENVptr env, CPXCLPptr lp, int * indvar_p, int *
complemented_p, int * nzcnt_p, double * rhs_p, char * sense_p, int * linind, double
* linval, int space, int * surplus_p, int which)
```

Definition file: cplex.h

The routine `CPXgetindconstr` accesses a specified indicator constraint on the variables of a CPLEX problem object. The length of the arrays in which the nonzero coefficients of the constraint are to be returned must be specified.

Note

If the value of `space` is 0 (zero), then the negative of the value of `*surplus_p` returned specifies the length needed for the arrays `linind` and `linval`.

Example

```
status = CPXgetindconstr (env, lp, &indvar, &complemented,
                          &linnzcnt, &rhs, &sense, linind, linval,
                          space, &surplus, 0);
```

Parameters:

env	A pointer to the CPLEX environment as returned by the `CPXopenCPLEX` routine.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
indvar_p	A pointer to an integer to contain the index of the binary indicator variable. May be NULL.
complemented_p	A pointer to a Boolean value that specifies whether the indicator variable is complemented. May be NULL.
nzcnt_p	A pointer to an integer to contain the number of nonzero values in the linear portion of the indicator constraint; that is, the true length of the arrays `linind` and `linval`.
rhs_p	A pointer to a `double` containing the righthand side value of the linear portion of the indicator constraint.
sense_p	A pointer to a character specifying the sense of the linear portion of the constraint. Possible values are L for a <= constraint, E for an = constraint, or G for a >= constraint.
linind	An array to contain the variable indices of the entries of `linval`. May be NULL if `space` is 0 (zero).
linval	An array to contain the coefficients of the linear portion of the specified indicator constraint. May be NULL if `space` is 0.
space	An integer specifying the length of the arrays `linind` and `linval`. May be 0 (zero).
surplus_p	A pointer to an integer to contain the difference between `space` and the number of entries in each of the arrays `linind` and `linval`. A nonnegative value of `surplus_p` reports that the length of the arrays was sufficient. A negative value reports that the length was insufficient and that the routine could not complete its task. In this case, the routine `CPXgetindconstr` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of `surplus_p` specifies the amount of insufficient space in the arrays. May be NULL if `space` is 0 (zero).
which	An integer specifying which indicator constraint to return.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` reports that insufficient space was available in either of the arrays `linind` and `linval` to hold the nonzero coefficients.

Global function CPXgetphase1cnt

`int` **`CPXgetphase1cnt`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetphase1cnt` accesses the number of Phase I iterations to solve a problem using the primal or dual simplex method.

Example

```
itcnt = CPXgetphase1cnt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If a solution exists, CPXgetphase1cnt returns the Phase I iteration count. If no solution exists, CPXgetphase1cnt returns the value 0.

Global function CPXchgrngval

`int` **`CPXchgrngval`**`(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices, const double * values)`

Definition file: cplex.h

The routine `CPXchgrngval` changes the range coefficients of a set of linear constraints in the CPLEX problem object.

Example

```
status = CPXchgrngval (env, lp, cnt, indices, values);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt	An integer that specifies the total number of range coefficients to be changed, and thus specifies the length of the arrays `indices` and `values`.
indices	An array of length `cnt` containing the numeric indices of the rows corresponding to the linear constraints for which range coefficients are to be changed.
values	An array of length `cnt` containing the new values of the range coefficients of the linear constraints present in `indices`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETaddnodes

```
int CPXNETaddnodes(CPXCENVptr env, CPXNETptr net, int nnodes, const double *
supply, char ** name)
```

Definition file: cplex.h

The routine `CPXNETaddnodes` adds new nodes to the network stored in a network problem object.

Example

```
status = CPXNETaddnodes (env, net, nnodes, supply, NULL);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
nnodes	Number of nodes to add.
supply	Supply values for the added nodes. If NULL is passed, all supplies defaults to 0 (zero). Otherwise, the size of the array must be at least `nnodes`.
name	Pointer to an array of names for added nodes. If NULL is passed and the existing nodes have names, default names are assigned to the added nodes. If NULL is passed but the existing nodes have no names, the new nodes are assigned no names. Otherwise, the size of the array must be at least `nnodes` and every name in the array must be a string terminating in 0. If the existing nodes have no names and `nnames` is not NULL, default names are assigned to the existing nodes.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXNETgetobj

`int` **`CPXNETgetobj`**`(CPXCENVptr env, CPXCNETptr net, double * obj, int begin, int end)`

Definition file: cplex.h

The routine `CPXNETgetobj` is used to access the objective function values for a range of arcs in the network stored in a network problem object.

Example

```
status = CPXNETgetobj (env, net, obj, 0, cur_narcs-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
obj	Array in which to write the objective values for the requested range of arcs. If NULL is passed, no objective values are retrieved. Otherwise, `obj` must point to an array of size at least (`end-begin+1`).
begin	Index of the first arc for which the objective value is to be obtained.
end	Index of the last arc for which the objective value is to be obtained.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetcrossdpushcnt

`int` **`CPXgetcrossdpushcnt`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetcrossdpushcnt` accesses the number of dual push iterations in the crossover method. A push occurs when a nonbasic variable switches bounds and does not enter the basis.

Example

```
itcnt = CPXgetcrossdpushcnt (env, lp);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

lp A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns the dual push iteration count if a solution exists. If no solution exists, it returns zero.

Global function CPXgetparamtype

`int` **`CPXgetparamtype`**`(CPXCENVptr env, int whichparam, int * paramtype)`

Definition file: cplex.h

The routine `CPXgetparamtype` returns the type of a CPLEX parameter, given the symbolic constant or reference number for it.

The *CPLEX Parameters Reference Manual* provides a list of parameters with their types, options, and default values.

Example

```
status = CPXgetparamtype (env, CPX_PARAM_ADVIND, &paramtype);
```

Possible values returned in `paramtype`

CPX_PARAMTYPE_NONE	0 (zero)
CPX_PARAMTYPE_INT	1 (one)
CPX_PARAMTYPE_DOUBLE	2
CPX_PARAMTYPE_STRING	3

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
whichparam	An integer specifying the symbolic constant or reference number of the parameter for which the type is to be obtained.
paramtype	A pointer to an integer to receive the type.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETsolution

```
int CPXNETsolution(CPXCENVptr env, CPXCNETptr net, int * netstat_p, double *
objval_p, double * x, double * pi, double * slack, double * dj)
```

Definition file: cplex.h

The routine CPXNETsolution accesses solution values for a network problem object computed by the most recent call to CPXNETprimopt for that object. The solution values are maintained in the object as long as no changes are applied to it with one of the CPXNETchg..., CPXNETcopy... or CPXNETadd... functions. Whether or not a solution exists can be determined by CPXNETsolninfo.

The arguments to CPXNETsolution are pointers to locations where data is to be written. Such data includes the solution status, the value of the objective function, primal, dual and slack values and the reduced costs.

Although all the above data exists after a successful call to CPXNETprimopt, it is possible that the user only needs a subset of the available data. Thus, if any part of the solution represented by an argument to CPXNETsolution is not required, a NULL pointer can be passed for that argument.

Example

```
status = CPXNETsolution (env, net, &netstatus, &objval, x, pi,
                         slack, dj);
```

Parameters:

env	A pointer to the CPLEX environment as returned by CPXopenCPLEX.
net	A pointer to a CPLEX network problem object as returned by CPXNETcreateprob.
netstat_p	Pointer to which the solution status is to be written. The specific values that *netstat_p can take and their meanings are the same as the return values documented for CPXNETgetstat.
objval_p	Pointer to which the objective value is to be written. If NULL is passed, no objective value is returned. If the solution status is one of the CPX_STAT_ABORT codes, the value returned depends on the setting of parameter CPX_PARAM_NETDISPLAY. If this parameter is set to 2, objective function values that are penalized for infeasible flows are used to compute the objective value of the solution. Otherwise, the true objective function values are used.
x	Array to which the solution (flow) vector is to be written. If NULL is passed, no solution vector is returned. Otherwise, x must point to an array of size at least that returned by CPXNETgetnumarcs.
pi	Array to which the dual values are to be written. If NULL is passed, no dual values are returned. Otherwise, pi must point to an array of size at least that returned by CPXNETgetnumnodes.
slack	Array to which the slack values (violations of supplies/demands) are to be written. If NULL is passed, no slack values are returned. Otherwise, slack must point to an array of size at least that returned by CPXNETgetnumnodes.
dj	Array to which the reduced cost values are to be written. If NULL is passed, no reduced cost values are returned. Otherwise, dj must point to an array of size at least that returned by CPXNETgetnumarcs.

Returns:

If a solution exists, it returns zero; if not, it returns nonzero to indicate an error.

Global function CPXgetstrparam

```
int CPXgetstrparam(CPXCENVptr env, int whichparam, char * value_str)
```

Definition file: cplex.h

The routine `CPXgetstrparam` obtains the current value of a CPLEX string parameter.

Example

```
status = CPXgetstrparam (env, CPX_PARAM_WORKDIR, dirname);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
whichparam	The symbolic constant (or reference number) of the parameter for which the value is to be obtained.
value_str	A pointer to a buffer of length at least `CPX_STR_PARAM_MAX` to hold the current value of the CPLEX parameter.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXstrlen

```
int CPXstrlen(const char * s_str)
```

Definition file: cplex.h

The routine `CPXstrlen` determines the length of a string. It is exactly the same as the standard C library routine `strlen`. This routine is provided so that strings passed to the message function routines (see `CPXaddfuncdest`) can be analyzed by languages that do not allow dereferencing of pointers (for example, older versions of Visual Basic).

Example

```
len = CPXstrlen (p);
```

Parameters:

s_str A pointer to a character string.

Returns:

The routine returns the length of the string.

Global function CPXNETgetnodearcs

```
int CPXNETgetnodearcs(CPXCENVptr env, CPXCNETptr net, int * arccnt_p, int * arcbeg,
int * arc, int arcspace, int * surplus_p, int begin, int end)
```

Definition file: cplex.h

The routine `CPXNETgetnodearcs` is used to access the arc indices incident to a range of nodes in the network stored in a network problem object.

Example

```
status = CPXNETgetnodearcs (env, net, &arccnt, arcbeg, arc,
                            arcspace, &surplus, begin, end);
```

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`net`

A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

`arccnt_p`

A pointer to an integer to contain the total number of arc indices returned in the array `arc`.

`arcbeg`

An array that contain indices indicating where each of the requested arc lists start in array `arc`. Specifically, the list of arcs incident to node `i` (< `end`) consists of the entries in `arc` in the range from `arcbeg[i-begin]` to `arcbeg[(i+1)-begin]-1`. The list of arcs incident to node `end` consists of the entries in `arc` in the range from `arcbeg[end-begin]` to `*arccnt_p-1`. This array must have a length of at least `end-begin+1`.

`arc`

An array that contain the arc indices for the arcs incident to the nodes in the specified range. May be NULL if `arcspace` is zero.

`arcspace`

An integer indicating the length of the array `arc`. May be zero.

`surplus_p`

A pointer to an integer to contain the difference between `arcspace` and the number of arcs incident to the nodes in the specified range. A nonnegative value indicates that `arcspace` was sufficient. A negative value indicates that it was insufficient and that the routine could not complete its task. In that case, `CPXERR_NEGATIVE_SURPLUS` is returned and the negative value of `surplus_p` indicates the amount of insufficient space in the array `arc`.

`begin`

Index of the first node for which arcs are to be obtained.

`end`

Index of the last node for which arcs are to be obtained.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgeterrorstring

`CPXCCHARptr` **`CPXgeterrorstring`**`(CPXCENVptr env, int errcode, char * buffer_str)`

Definition file: cplex.h

The routine `CPXgeterrorstring` returns an error message string corresponding to an error code. Error codes are returned by CPLEX routines when an error occurs.

> **Note**
>
> This routine allows the CPLEX environment argument to be NULL so that errors caused by the routine `CPXopenCPLEX` can be translated.

Example

```
char *errstr;
     errstr = CPXgeterrorstring (env, errcode, buffer);
     if ( errstr != NULL ) {
        printf ("%sn", buffer);
     }
     else {
        printf ("CPLEX Error %5d:  Unknown error code.n",
                errcode);
     }
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
errcode	The error code to be translated.
buffer_str	A character string buffer. This buffer must be at least 4096 characters to hold the error string.

Returns:

This routine returns a pointer to the argument `buffer_str` if the string does exist. In that case, `buffer_str` contains the error message string. It returns NULL if the error code does not have a corresponding string.

Global function CPXgetconflictext

`int` **`CPXgetconflictext`**`(CPXCENVptr env, CPXCLPptr lp, int * grpstat, int beg, int end)`

Definition file: cplex.h

For an infeasible problem, if the infeasibility has been analysed by `CPXrefineconflictext`, this routine accesses information about the conflict computed by it. The conflict status codes of the groups numbered `beg` (for begin) through `end` in the most recent call to `CPXrefineconflictext` are returned.

Group Status

The conflict status for group `beg+i` will be returned in `grpstat[i]`. Possible values for the status of a group as returned in `grpstat` are the following:

- `CPX_CONFLICT_EXCLUDED` if the group was proven to be not relevant to the conflict;
- `CPX_CONFLICT_POSSIBLE_MEMBER` if the group may be relevant to the conflict but has not (yet) been proven so;
- `CPX_CONFLICT_MEMBER` if the group has been proven to be relevant for the conflict.

Example

```
status = CPXgetconflictext (env, lp, grpstat, 0, ngrp-1);
```

See Also: CPXrefineconflictext, CPXclpwrite

Parameters:

env	A pointer to the CPLEX environment as returned by the routine `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
grpstat	Pointer to an array where the values denoting the conflict status of the groups are returned. This array must have a length of at least `end-beg+1`.
beg	The index of the first group defined at the most recent call to `CPXrefineconflictext` for which the conflict status will be returned.
end	The index of the last group defined at the most recent call to `CPXrefineconflictext` for which the conflict status will be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsettuningcallbackfunc

```
int CPXsettuningcallbackfunc(CPXENVptr env, int(CPXPUBLIC *callback)(CPXCENVptr,
void *, int, void *), void * cbhandle)
```

Definition file: cplex.h

The routine `CPXsettuningcallbackfunc` modifies the user-written callback routine to be called before each trial run during the tuning process.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle);
```

This is the user-written callback routine.

Callback return value

A nonzero terminates the tuning.

Callback arguments

`env`

A pointer to the CPLEX environment that was passed into the associated tuning routine.

`cbdata`

A pointer passed from the tuning routine to the user-written callback function that contains information about the tuning process. The only purpose for the `cbdata` pointer is to pass it to the routine `CPXgetcallbackinfo`.

`wherefrom`

An integer value specifying from which procedure the user-written callback function was called. This value will always be `CPX_CALLBACK_TUNING` for this callback.

`cbhandle`

Pointer to user private data, as passed to `CPXsettuningcallbackfunc`.

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`myfunc`

A pointer to a user-written callback function. Setting `callback` to NULL prevents any callback function from being called during tuning. The call to `callback` occurs before each trial run of the tuning. This function is written by the user; its prototype is documented here.

`cbhandle`

A pointer to user private data. This pointer is passed to the callback function.

Example

```
status = CPXsettuningcallbackfunc (env, myfunc, NULL);
```

See Also: CPXgetcallbackinfo

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETgetarcname

```
int CPXNETgetarcname(CPXCENVptr env, CPXCNETptr net, char ** nnames, char *
namestore, int namespc, int * surplus_p, int begin, int end)
```

Definition file: cplex.h

The routine `CPXNETgetarcname` is used to access the names of a range of arcs in a network stored in a network problem object. The beginning and end of the range, along with the length of the array in which the arc names are to be returned, must be specified.

Example

```
status = CPXNETgetarcname (env, net, nnames, namestore, namespc,
                           &surplus, 0, narcs-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
nnames	Where to copy pointers to arc names stored in the `namestore` array. The length of this array must be at least (`end-begin+1`). The pointer to the name of arc `i` is returned in `nnames[i-begin]`.
namestore	Array of characters to which the specified arc names are to be copied. It may be NULL if `namespc` is 0.
namespc	Length of the `namestore` array.
surplus_p	Pointer to an integer to which the difference between `namespc` and the number of characters required to store the requested names is returned. A nonnegative value indicates that `namespc` was sufficient. A negative value indicates that it was insufficient. In that case, `CPXERR_NEGATIVE_SURPLUS` is returned and the negative value of `surplus_p` indicates the amount of insufficient space in the array `namestore`.
begin	Index of the first arc for which a name is to be obtained.
end	Index of the last arc for which a name is to be obtained.

Returns:

The routine returns zero on success and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` indicates that insufficient space was available in the `namestore` array to hold the names.

Global function CPXgetsense

```
int CPXgetsense(CPXCENVptr env, CPXCLPptr lp, char * sense, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetsense` accesses the sense for a range of constraints in a CPLEX problem object. The beginning and end of the range must be specified.

Example

```
status = CPXgetsense (env, lp, sense, 0, cur_numrows-1);
```

Values of sense

`sense[i]`	= 'L'	<= constraint
`sense[i]`	= 'E'	= constraint
`sense[i]`	= 'G'	>= constraint
`sense[i]`	= 'R'	ranged constraint

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
sense	An array where the specified constraint senses are to be returned. This array must be of length at least (`end - begin + 1`). The sense of constraint `i` is returned in `sense[i - begin]`. Possible values appear in the table.
begin	An integer specifying the beginning of the range of constraint senses to be returned.
end	An integer specifying the end of the range of constraint senses to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetcols

```
int CPXgetcols(CPXCENVptr env, CPXCLPptr lp, int * nzcnt_p, int * cmatbeg, int *
cmatind, double * cmatval, int cmatspace, int * surplus_p, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetcols` accesses a range of columns of the constraint matrix of a CPLEX problem object. The beginning and end of the range, along with the length of the arrays in which the nonzero entries of these columns are to be returned, must be specified.

> **Note**
>
> If the value of `cmatspace` is zero, the negative of the value of `surplus_p` returned specifies the length needed for the arrays `cmatind` and `cmatval`.

Example

```
status = CPXgetcols (env, lp, &nzcnt, cmatbeg, cmatind,
                     cmatval, cmatspace, &surplus, 0,
                     cur_numcols-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
nzcnt_p	A pointer to an integer to contain the number of nonzeros returned; that is, the true length of the arrays `cmatind` and `cmatval`.
cmatbeg	An array to contain indices specifying where each of the requested columns begins in the arrays `cmatval` and `cmatind`. Specifically, column `j` consists of the entries in `cmatval` and `cmatind` in the range from `cmatbeg[j - begin]` to `cmatbeg[(j + 1) - begin]-1`. (Column `end` consists of the entries from `cmatbeg[end - begin]` to `nzcnt_p-1`.) This array must be of length at least (`end - begin + 1`).
cmatind	An array to contain the row indices associated with the elements of `cmatval`. May be NULL if `cmatspace` is zero.
cmatval	An array to contain the nonzero coefficients of the specified columns. May be NULL if `cmatspace` is zero.
cmatspace	An integer specifying the length of the arrays `cmatind` and `cmatval`. May be zero.
surplus_p	A pointer to an integer to contain the difference between `cmatspace` and the number of entries in each of the arrays `cmatind` and `cmatval`. A nonnegative value of `surplus_p` specifies that the length of the arrays was sufficient. A negative value specifies that the length was insufficient and that the routine could not complete its task. In this case, `CPXgetcols` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of `surplus_p` specifies the amount of insufficient space in the arrays.
begin	An integer specifying the beginning of the range of columns to be returned.
end	An integer specifying the end of the range of columns to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` specifies that insufficient space was available in the arrays `cmatind` and `cmatval` to hold the nonzero coefficients.

Global function CPXgetparamnum

```
int CPXgetparamnum(CPXCENVptr env, const char * name_str, int * whichparam_p)
```

Definition file: cplex.h

The routine `CPXgetparamnum` returns the reference number of a CPLEX parameter, given a character string containing the name for it.

The *CPLEX Parameters Reference Manual* provides a list of parameters with their types, options, and default values.

Example

```
status = CPXgetparamnum (env, "CPX_PARAM_ADVIND", param_number);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
name_str	A character array containing the name of the target parameter.
whichparam_p	A pointer to an integer to receive the reference number.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetbasednorms

```
int CPXgetbasednorms(CPXCENVptr env, CPXCLPptr lp, int * cstat, int * rstat, double
* dnorm)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetbasednorms` works in conjunction with the routine `CPXcopybasednorms`. `CPXgetbasednorms` retrieves the resident basis and dual norms from a specified problem object.

Each of the arrays `cstat`, `rstat`, and `dnorm` must be non NULL. That is, each of these arrays must be allocated. The allocated size of `cstat` is assumed by this routine to be at least the number returned by `CPXgetnumcols`. The allocated size of `rstat` and `dnorm` are assumed to be at least the number returned by `CPXgetnumrows`. (Other details of `cstat`, `rstat`, and `dnorm` are not documented.)

Success, Failure

If this routine succeeds, `cstat` and `rstat` contain information about the resident basis, and `dnorm` contains the dual steepest-edge norms. If there is no basis, or if there is no set of dual steepest-edge norms, this routine returns an error code. The returned data are intended solely for use by `CPXcopybasednorms`.

Example

For example, if a given LP has just been successfully solved by the Callable Library optimizer `CPXdualopt` with the dual pricing option `CPX_PARAM_DPRIIND` set to `CPX_DPRIIND_STEEP`, `CPX_DPRIIND_FULLSTEEP`, or `CPX_DPRIIND_STEEPQSTART`, then a call to `CPXgetbasednorms` should succeed. (That optimizer and those pricing options are documented in the *Callable Library Reference Manual*, and their use is illustrated in the *CPLEX User's Manual*.)

Motivation

When the Callable Library optimizer `CPXdualopt` is called to solve a problem with the dual pricing option `CPX_PARAM_DPRIIND` set to `CPX_DPRIIND_STEEP` or `CPX_DPRIIND_FULLSTEEP`, there must be values of appropriate dual norms available before the optimizer can begin. If these norms are not already resident, they must be computed, and that computation may be expensive. The functions `CPXgetbasednorms` and `CPXcopybasednorms` can, in some cases, avoid that expense. Suppose, for example, that in some application an LP is solved by `CPXdualopt` with one of those pricing settings. After the solution of the LP, some intermediate optimizations are carried out on the same LP, and those subsequent optimizations are in turn followed by some changes to the LP, and a re-solve. In such a case, copying the basis and norms that were resident before the intermediate solves, back into CPLEX data structures can greatly increase the speed of the re-solve.

See Also: CPXcopybasednorms

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX LP problem object, as returned by `CPXcreateprob`.
cstat	An array containing the basis status of the columns in the constraint matrix. The length of the allocated array is at least the value returned by `CPXgetnumcols`.
rstat	An array containing the basis status of the rows in the constraint matrix. The length of the allocated array is at least the value returned by `CPXgetnumrows`.

dnorm An array containing the dual steepest-edge norms. The length of the allocated array is at least the value returned by `CPXgetnumrows`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXfreepresolve

`int` **`CPXfreepresolve`**`(CPXCENVptr env, CPXLPptr lp)`

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXfreepresolve` frees the presolved problem from the LP problem object. Under the default setting of `CPX_PARAM_REDUCE`, the presolved problem is freed when an optimal solution is found. It is not freed when `CPX_PARAM_REDUCE` is set to `CPX_PREREDUCE_PRIMALONLY` (`1`) or `CPX_PREREDUCE_DUALONLY` (`2`), so the routine `CPXfreepresolve` can be used to free it manually.

Example

```
status = CPXfreepresolve (env, lp);
```

Parameters:

env A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

lp A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetindconstrinfeas

```
int CPXgetindconstrinfeas(CPXCENVptr env, CPXCLPptr lp, const double * x, double *
infeasout, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetindconstrinfeas` computes the infeasibility of a given solution for a range of indicator constraints. The beginning and end of the range must be specified. For each constraint, the infeasibility value returned is 0 (zero) if the constraint is satisfied. In particular, the infeasibility value returned is 0 (zero) if the indicator constraint is not active in the queried solution. Otherwise, the infeasibility value returned is the amount by which the righthand side of the linear portion of the constraint must be changed to make the queried solution valid. It is positive for a less-than-or-equal-to constraint, negative for a greater-than-or-equal-to constraint, and can be of any sign for an equality constraint.

Example

```
status = CPXgetindconstrinfeas (env, lp, NULL, infeasout, 0, CPXgetnumindconstrs(env,lp)-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
x	The solution whose infeasibility is to be computed. May be NULL in which case the resident solution is used.
infeasout	An array to receive the infeasibility value for each of the indicator constraints. This array must be of length at least (`end - begin + 1`).
begin	An integer specifying the beginning of the range of indicator constraints whose infeasibility is to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetcrossdexchcnt

```
int CPXgetcrossdexchcnt(CPXCENVptr env, CPXCLPptr lp)
```

Definition file: cplex.h

The routine `CPXgetcrossdexchcnt` accesses the number of dual exchange iterations in the crossover method. An exchange occurs when a nonbasic variable is forced to enter the basis as it is pushed toward a bound.

Example

```
itcnt = CPXgetcrossdexchcnt (env, lp);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

lp A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns the dual exchange iteration count if a solution exists. If no solution exists, it returns zero.

Global function CPXaddsos

```
int CPXaddsos(CPXCENVptr env, CPXLPptr lp, int numsos, int numsosnz, const char *
sostype, const int * sosbeg, const int * sosind, const double * soswt, char **
sosname)
```

Definition file: cplex.h

The routine `CPXaddsos` adds information about a special ordered set (SOS) to a problem object of type `CPXPROB_MILP`, `CPXPROB_MIQP`, or `CPXPROB_MIQCP`. The problem may already contain SOS information.

Table 1: Values of elements of sostype

`CPX_TYPE_SOS1`	`'1'`	Type 1
`CPX_TYPE_SOS2`	`'2'`	Type 2

The arrays `sosbeg`, `sosind`, and `soswts` follow the same conventions as similar arrays in other routines of the Callable Library. For `j < numsos-1`, the indices of the set `j` must be stored in `sosind[sosbeg[j]], ..., sosind[sosbeg[j+1]-1]` and the weights in `soswt[sosbeg[j],..., soswt[sosbeg[j+1]-1]`. For the last set, `j = numsos-1`, the indices must be stored in `sosind[sosbeg[numsos-1]],..., sosind[numsosnz-1]` and the corresponding weights in `soswt[sosbeg[numsos-1]],..., soswt[numsosnz-1]`. Hence, the length of `sosbeg` must be at least `numsos`, while the lengths of `sosind` and `soswt` must must be at least `numsosnz`.

Example

```
status = CPXaddsos (env, lp, numsos, numsosnz, sostype,
                    sosbeg, sosind, soswt, NULL);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
numsos	The number of sets to be added to existing SOS sets, if any.
numsosnz	The total number of members in all of the sets to be added to existing SOS sets, if any.
sostype	An array containing SOS type information for the sets to be added. According to Table 1, `sostype[i]` specifies the SOS type of set `i`. The length of this array must be at least `numsos`.
sosbeg	An array that with `sosind` and `soswt` defines the weights for the sets to be added.
sosind	An array that with `sosbeg` and `soswt` defines the weights of the sets to be added.
soswt	An array that with `sosbeg` and `sosind` defines the indices and weights for the sets to be added. The indices of each set must be stored in sequential locations in `sosind`. The weights of each set must be stored in sequential locations in `soswt`. The array `sosbeg[j]` containing the index of the beginning of set `j`. The weights must be unique within each set.
sosname	An array containing pointers to character strings that represent the names of the new SOSs. May be NULL, in which case the new SOSs are assigned default names if the SOSs already resident in the CPLEX problem object have names; otherwise, no names are associated with the sets. If SOS names are passed to `CPXaddsos` but existing SOSs have no names assigned, default names are created for them.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXaddsolnpooldivfilter

```
int CPXaddsolnpooldivfilter(CPXCENVptr env, CPXLPptr lp, double lower_bound, double
upper_bound, int nzcnt, const int * ind, const double * weight, const double *
refval, const char * lname_str)
```

Definition file: cplex.h

The routine `CPXaddsolnpooldivfilter` adds a new diversity filter to the solution pool.

A *diversity filter* drives the search for multiple solutions toward new solutions that satisfy a measure of diversity specified in the filter.

This diversity measure applies only to binary variables.

Potential new solutions are compared to a reference set. You must specify which variables are to be compared. You do so with the argument `ind` designating the indices of variables to include in the diversity measure.

A *reference set* is the set of values specified by the argument `refval`.

You may optionally specify weights (that is, coefficients to form a linear expression in terms of the variables) in the diversity measure; if you do not specify weights, all differences between the reference set and potential new solutions will be weighted by the value 1.0 (one). CPLEX computes the diversity measure by summing the pair-wise weighted absolute differences from the reference values, like this:

```
differences(x) = sum {weight[i] times |x[ind[i]] - refval[i]|}.
```

A diversity filter makes sure that the solutions satisfy the constraint:

```
lower bound <= differences(x) <= upper bound
```

You may specify both a lower and upper bound on diversity.

In order to say, *Give me solutions that are close to this one, within this specified set of variables,* specify a `lower_bound` of 0.0 (zero) and a finite `upper_bound`. CPLEX then looks for solutions that differ from the reference values by at most the value of `upper_bound`, within the specified set of variables.

In order to say, *Give me solutions that are different from this one,* specify a finite `lower_bound` and an infinite (that is, very large) `upper_bound` on the diversity. CPLEX then looks for solutions that differ from the reference values by at least the value of `lower_bound`, within the specified set of variables.

Example

```
status = CPXaddsolnpooldivfilter (env, lp, loval, hival,
                                  cnt, ind, val, refval, fnamestr);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lower_bound	Lower bound on the diversity measure for new solutions allowed in the pool.
upper_bound	Upper bound on the diversity measure for new solutions allowed in the pool.
nzcnt	Number of variables used to define the diversity measure.
ind	An array of indices of variables in the diversity measure.
weight	An array of weights to be used in the diversity measure. The indices and corresponding weights must be stored in sequential locations in the arrays `ind` and `weight` from positions 0 (zero) to `num-1`. Each entry, `ind[i]`, specifies the variable index of the corresponding weight, `weight[i]`. May be NULL, in which case CPLEX uses weights of 1.0 (one).

refval	An array of reference values for the variables with indices in the array `ind` to compare with a solution when CPLEX computes the diversity measure.
lname_str	The name of the filter. May be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXbaropt

```
int CPXbaropt(CPXCENVptr env, CPXLPptr lp)
```

Definition file: cplex.h

The routine `CPXbaropt` may be used to find a solution to a linear program (LP), quadratic program (QP), or quadratically constrained program (QCP) by means of the barrier algorithm at any time after the problem is created by a call to `CPXcreateprob`. The optimization results are recorded in the CPLEX problem object.

Example

```
status = CPXbaropt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (`CPXERR_NO_MEMORY`) or encountering invalid data in the CPLEX problem object (`CPXERR_NO_PROBLEM`). Exceeding a user-specified CPLEX limit or proving the model infeasible or unbounded are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines `CPXsolninfo`, `CPXgetstat`, and `CPXsolution` to obtain further information about the status of the optimization.

Global function CPXdelsolnpoolfilters

`int` **`CPXdelsolnpoolfilters`**`(CPXCENVptr env, CPXLPptr lp, int begin, int end)`

Definition file: cplex.h

The routine `CPXdelsolnpoolfilters` deletes filters from the the problem object specified by the argument `lp`. The range of filters to delete is specified by the argument `begin`, the lower index that represents the first filter to be deleted, and the argument `end`, representing the last filter to be deleted. The indices of the filters following those deleted are decreased by the number of deleted filters.

Example

```
status = CPXdelsolnpoolfilters (env, lp, 10, 20);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
begin	An integer that specifies the numeric index of the first filter to be deleted.
end	An integer that specifies the numeric index of the last filter to be deleted.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXmipopt

`int` **`CPXmipopt`**`(CPXCENVptr env, CPXLPptr lp)`

Definition file: cplex.h

At any time after a mixed integer program has been created by a call to `CPXcreateprob`, the routine `CPXmipopt` may be used to find a solution to that problem.

An LP solution does not exist at the end of `CPXmipopt`. To obtain post-solution information for the LP subproblem associated with the integer solution, use the routine `CPXchgprobtype`.

Example

```
status = CPXmipopt (env, lp);
```

See also the example `mipex1.c` in the standard distribution.

Examples of errors include exhausting available memory (`CPXERR_NO_MEMORY`) or encountering invalid data in the CPLEX problem object (`CPXERR_NO_PROBLEM`).

Another possible error is the inability to solve a subproblem satisfactorily, as reported by `CPXERR_SUBPROB_SOLVE`. The solution status of the subproblem optimization can be obtained with the routine `CPXgetsubstat`.

Exceeding a user-specified CPLEX limit is not considered an error. Proving the problem infeasible or unbounded is not considered an error.

Note that a zero return value does not necessarily mean that a solution exists. Use the query routines `CPXsolninfo`, `CPXgetstat`, `CPXsolution` and the special mixed integer solution routines to obtain further information about the status of the optimization.

See Also: CPXgetstat, CPXsolninfo, CPXsolution, CPXgetobjval

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

lp A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Examples of errors include exhausting available memory (`CPXERR_NO_MEMORY`) or encountering invalid data in the CPLEX problem object (`CPXERR_NO_PROBLEM`).

Another possible error is the inability to solve a subproblem satisfactorily, as reported by `CPXERR_SUBPROB_SOLVE`. The solution status of the subproblem optimization can be obtained with the routine `CPXgetsubstat`.

Exceeding a user-specified CPLEX limit is not considered an error. Proving the problem infeasible or unbounded is not considered an error.

Note that a zero return value does not necessarily mean that a solution exists. Use the query routines `CPXsolninfo`, `CPXgetstat`, `CPXsolution` and the special mixed integer solution routines to obtain further information about the status of the optimization.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXpivotin

```
int CPXpivotin(CPXCENVptr env, CPXLPptr lp, const int * rlist, int rlen)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXpivotin` forcibly pivots slacks that appear on a list of inequality rows into the basis. If equality rows appear among those specified on the list, they are ignored.

Motivation

In the implementation of cutting-plane algorithms for integer programming, it is occasionally desirable to delete some of the added constraints (that is, cutting planes) when they no longer appear to be useful. If the slack on some such constraint (that is, row) is not in the resident basis, the deletion of that row may destroy the quality of the basis. Pivoting the slack in before the deletion avoids that difficulty.

Dual Steepest-Edge Norms

If one of the dual steepest-edge algorithms is in use when this routine is called, the corresponding norms are automatically updated as part of the pivot. (Primal steepest-edge norms are not automatically updated in this way because, in general, the deletion of rows invalidates those norms.)

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
rlist	An array of length `rlen`, containing distinct row indices of slack variables that are not basic in the current solution. If `rlist[]` contains negative entries or entries exceeding the number of rows, `CPXpivotin` returns an error code. Entries of nonslack rows are ignored.
rlen	An integer that specifies the number of entries in the array `rlist[]`. If `rlen` is negative or greater than the number of rows, `CPXpivotin` returns an error code.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXtuneparam

```
int CPXtuneparam(CPXENVptr env, CPXLPptr lp, int intcnt, const int * intnum, const
int * intval, int dblcnt, const int * dblnum, const double * dblval, int strcnt,
const int * strnum, char ** strval, int * tunestat_p)
```

Definition file: cplex.h

The routine CPXtuneparam tunes the parameters of the environment for improved optimizer performance on the specified problem object. Tuning is carried out by making a number of trial runs with a variety parameter settings. Parameters and associated values which should not be changed by the tuning process (known as the fixed parameters), can be specified as arguments.

This routine does not apply to network models, nor to quadratically constrained programming problems (QCP).

After CPXtuneparam has finished, the environment will contain the combined fixed and tuned parameter settings which the user can query or write to a file. The problem object will not have a solution.

The parameter CPX_PARAM_TUNINGREPEAT specifies how many problem variations for CPLEX to try while tuning. Using a number of variations can give more robust results when tuning is applied to a single problem. The tuning evaluation measure is meaningful only when CPX_PARAM_TUNINGREPEAT is larger than one.

All callbacks, except the tuning informational callback, will be ignored. Tuning will monitor the value set by CPXsetterminate and terminate when this value is set.

A few of the parameter settings in the environment control the tuning process. They are specified in the table; other parameter settings in the environment are ignored.

Parameter	Use
CPX_PARAM_TILIM	Limits the total time spent tuning
CPX_PARAM_TUNINGTILIM	Limits the time of each trial run
CPX_PARAM_TUNINGMEASURE	Controls the tuning evaluation measure
CPX_PARAM_TUNINGREPEAT	Sets the number of repeated problem variations
CPX_PARAM_TUNINGDISPLAY	Controls the level of the tuning display
CPX_PARAM_SCRIND	Controls screen output

The value tunestat is 0 (zero) when tuning has completed and nonzero when it has not yet completed. The two nonzero statuses are CPX_TUNE_ABORT, which will be set when the terminate value passed to CPXsetterminate is set, and CPX_TUNE_TILIM, which will be set when the time limit specified by CPX_PARAM_TILIM is reached. Tuning will set any parameters which have been tuned so far even when tuning has not completed for the problem as a whole.

Parameters:

env	A pointer to the CPLEX environment, as returned by CPXopenCPLEX.
lp	A pointer to a CPLEX problem object as returned by CPXcreateprob.
intcnt	An integer that specifies the number of integer parameters to be fixed during tuning. This specifies the length of the arrays intnum and intval.
intnum	An array containing the parameter numbers (unique identifiers) of the integer parameters which remain fixed. May be NULL if intcnt is 0 (zero).
intval	An array containing the values for the parameters listed in intnum. May be NULL if intcnt is 0 (zero).
dblcnt	An integer that specifies the number of double parameters to be fixed during tuning. This specifies the length of the arrays dblnum and dblval.

dblnum	An array containing the parameter numbers (unique identifiers) of the double parameters which remain fixed. May be NULL if `dblcnt` is 0 (zero).
dblval	An array containing the values for the parameters listed in `dblnum`. May be NULL if `dblcnt` is 0 (zero).
strcnt	An integer that specifies the number of string parameters to be fixed during tuning. This specifies the length of the arrays `strnum` and `strval`.
strnum	An array containing the parameter numbers (unique identifiers) of the integer parameters which remain fixed. May be NULL if `strcnt` is 0 (zero).
strval	An array containing the values for the parameters listed in `strnum`. May be NULL if `strcnt` is 0 (zero).
tunestat_p	A pointer to an integer to receive the tuning status.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetlpcallbackfunc

```
int CPXgetlpcallbackfunc(CPXCENVptr env, int(CPXPUBLIC **callback_p)(CPXCENVptr,
void *, int, void *), void ** cbhandle_p)
```

Definition file: cplex.h

The routine `CPXgetlpcallbackfunc` accesses the user-written callback routine to be called after each iteration during the optimization of a continuous problem (LP, QP, or QCP), and also periodically during the CPLEX presolve algorithm.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle);
```

This is the user-written callback routine.

Callback return value

A nonzero terminates the optimization.

Callback arguments

`env`

A pointer to the CPLEX environment that was passed into the associated optimization routine.

`cbdata`

A pointer passed from the optimization routine to the user-written callback function that identifies the LP problem being optimized. The only purpose for the `cbdata` pointer is to pass it to the routine `CPXgetcallbackinfo`.

`wherefrom`

An integer value specifying which optimization algorithm the user-written callback function was called from. Possible values and their meaning appear in the table.

Value	Symbolic Constant	Meaning
1	CPX_CALLBACK_PRIMAL	From primal simplex
2	CPX_CALLBACK_DUAL	From dual simplex
4	CPX_CALLBACK_PRIMAL_CROSSOVER	From primal crossover
5	CPX_CALLBACK_DUAL_CROSSOVER	From dual crossover
6	CPX_CALLBACK_BARRIER	From barrier
7	CPX_CALLBACK_PRESOLVE	From presolve
8	CPX_CALLBACK_QPBARRIER	From QP barrier
9	CPX_CALLBACK_QPSIMPLEX	From QP simplex

`cbhandle`

Pointer to user private data, as passed to `CPXsetlpcallbackfunc`.

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`callback_p`

The address of the pointer to the current user-written callback function. If no callback function has been set, the pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the user's private pointer.

Example

```
status = CPXgetlpcallbackfunc (env, mycallback, NULL);
```

See Also: CPXgetcallbackinfo

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetchannels

```
int CPXgetchannels(CPXCENVptr env, CPXCHANNELptr * cpxresults_p, CPXCHANNELptr *
cpxwarning_p, CPXCHANNELptr * cpxerror_p, CPXCHANNELptr * cpxlog_p)
```

Definition file: cplex.h

The routine `CPXgetchannels` obtains pointers to the four default channels created when `CPXopenCPLEX` is called. To manipulate the messages for any of these channels, this routine must be called.

Example

```
status = CPXgetchannels (env, &cpxresults, &cpxwarning,
                         &cpxerror, &cpxlog);
```

See also `lpex5.c` in the *CPLEX User's Manual*.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
cpxresults_p	A pointer to a variable of type `CPXCHANNELptr` to hold the address of the channel corresponding to `cpxresults`. May be NULL.
cpxwarning_p	A pointer to a variable of type `CPXCHANNELptr` to hold the address of the channel corresponding to `cpxwarning`. May be NULL.
cpxerror_p	A pointer to a variable of type `CPXCHANNELptr` to hold the address of the channel corresponding to `cpxerror`. May be NULL.
cpxlog_p	A pointer to a variable of type `CPXCHANNELptr` to hold the address of the channel corresponding to `cpxlog`. May be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsolnpoolfiltername

```
int CPXgetsolnpoolfiltername(CPXCENVptr env, CPXCLPptr lp, char * buf_str, int
bufspace, int * surplus_p, int which)
```

Definition file: cplex.h

Accesses the name of a filter of the solution pool.

This routine accesses the name of a filter, specified by the argument `which`, of the problem object specified by the argument `lp`.

Note

If the value of `bufspace` is 0 (zero), then the negative of the value of `surplus_p` returned specifies the total number of characters needed for the array `buf_str`.

Example

```
status = CPXgetsolnpoolfiltername (env, lp,
                                   fnamestr,
                                   fnamespace,
                                   &surplus,
                                   i);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
buf_str	A pointer to a buffer of size `bufspace`. It may be NULL if `bufspace` is 0 (zero).
bufspace	An integer specifying the length of the array `buf_str`. It may be 0 (zero).
surplus_p	A pointer to an integer to contain the difference between `bufspace` and the amount of memory required to store the name of the filter. A nonnegative value of `surplus_p` specifies that the length of the array `buf_str` was sufficient. A negative value specifies that the length of the array was insufficient and that the routine could not complete its task. In this case, `CPXgetsolnpoolfiltername` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `surplus_p` specifies the amount of insufficient space in the array `buf_str`.
which	An integer specifying the index of the filter for which the name is returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` specifies that insufficient space was available in the array `buf_str` to hold the name of the filter.

Global function CPXNETgetobjsen

```
int CPXNETgetobjsen(CPXCENVptr env, CPXCNETptr net)
```

Definition file: cplex.h

The routine `CPXNETgetobjsen` returns the sense of the objective function (i.e., maximization or minimization) of a network problem object.

Example

```
objsen = CPXNETgetobjsen (env, net);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

Returns:

The value `CPX_MAX` (-1) is returned for a maximization problem; the value `CPX_MIN` (1) is returned for a minimization problem. In case of an error, the value zero is returned.

Global function CPXgetcallbacksosinfo

```
int CPXgetcallbacksosinfo(CPXCENVptr env, void * cbdata, int wherefrom, int
sosindex, int member, int whichinfo, void * result_p)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbacksosinfo` accesses information about special ordered sets (SOSs) during MIP optimization from within user-written callbacks. This routine may be called only when the value of its `wherefrom` argument is one of these values:

- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`, or
- `CPX_CALLBACK_MIP_CUT`.

The information returned is for the original problem if the parameter `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, it is for the presolved problem.

Example

```
status = CPXgetcallbacksosinfo(env, curlp, wherefrom, 6, 4,
                               CPX_CALLBACK_INFO_SOS_IS_FEASIBLE,
                               &isfeasible);
```

See also the example `admipex3.c` in the standard distribution.

Table 1: Information Requested for a User-Written SOS Callback

Symbolic Constant	C Type	Meaning
`CPX_CALLBACK_INFO_SOS_NUM`	`int`	number of SOSs
`CPX_CALLBACK_INFO_SOS_TYPE`	`char`	one of the values in Table 4
`CPX_CALLBACK_INFO_SOS_SIZE`	`int`	size of SOS
`CPX_CALLBACK_INFO_SOS_IS_FEASIBLE`	`int`	1 if SOS is feasible 0 if SOS is not
`CPX_CALLBACK_INFO_SOS_MEMBER_INDEX`	`int`	variable index of member-th member of SOS
`CPX_CALLBACK_INFO_SOS_MEMBER_REFVAL`	`double`	reference value (weight) of this member

Table 2: SOS Types Returned when whichinfo = CPX_CALLBACK_INFO_SOS_TYPE

Symbolic Constant	SOS Type
`CPX_SOS1`	type 1
`CPX_SOS2`	type 2

Parameters:

env A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting where the user-written callback was called from. This argument must be the value of `wherefrom` passed to the user-written callback.
sosindex	The index of the special ordered set (SOS) for which information is requested. SOSs are indexed from zero to `(numsets - 1)` where `numsets` is the result of calling this routine with a `whichinfo` value of `CPX_CALLBACK_INFO_SOS_NUM`.
member	The index of the member of the SOS for which information is requested.
whichinfo	An integer specifying which information is requested. Table 1 summarizes the possible values. Table 2 summarizes possible values returned when the type of information requested is the SOS type (that is, `whichinfo` = `CPX_CALLBACK_INFO_SOS_TYPE`).
result_p	A generic pointer to a variable of type `double`, `int`, or `char`. The variable represents the value returned by `whichinfo`. (The column C Type in the table shows the type of various values returned by `whichinfo`.)

Returns:

The routine returns zero if successful and nonzero if an error occurs. If the return value is nonzero, the requested value may not be available.

Global function CPXgetobjsen

`int` **`CPXgetobjsen`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetobjsen` accesses whether the objective function sense of a CPLEX problem object is maximization or minimization.

Example

```
cur_objsen = CPXgetobjsen (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

A value of CPX_MIN=1 is returned for minimization and CPX_MAX=-1 is returned for maximization. If the problem object or environment does not exist, a 0 is returned.

Global function CPXpreslvwrite

`int` **`CPXpreslvwrite`**`(CPXCENVptr env, CPXLPptr lp, const char * filename_str, double * objoff_p)`

Definition file: cplex.h

The routine `CPXpreslvwrite` writes a presolved version of the problem to a file. The file is saved in binary format, and can be read using the routine `CPXreadcopyprob`.

Note

Reductions done by the CPLEX presolve algorithms can cause the objective value to shift. As a result, the optimal objective obtained from solving the presolved problem created using `CPXpreslvwrite` may not be the same as the optimal objective of the original problem. The argument `objoff_p` can be used to reconcile this difference.

Example

```
status = CPXpreslvwrite (env, lp, "myfile.pre", &objoff);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the presolved problem should be written.
objoff_p	A pointer to a double precision variable that is used to hold the objective value difference between the original problem and the presolved problem. That is: `orginal objective value = (*objoff_p) + presolved objective value`

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXqpindefcertificate

```
int CPXqpindefcertificate(CPXCENVptr env, CPXCLPptr lp, double * x)
```

Definition file: cplex.h

The routine `CPXqpindefcertificate` computes a vector `x` that satisfies the inequality `x'Qx < 0`. Such a vector demonstrates that the matrix Q violates the assumption of positive semi-definiteness, and can be an aid in debugging a user's program if indefiniteness is an unexpected outcome.

Example

```
status = CPXqpindefcertificate (env, lp, x);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
x	An array to receive the values of the vector that is to be returned. The length of this array must be the same as the number of quadratic variables in the problem, which can be obtained by calling `CPXgetnumquad` for example.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetcrosspexchcnt

`int` **`CPXgetcrosspexchcnt`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetcrosspexchcnt` accesses the number of primal exchange iterations in the crossover method. An exchange occurs when a nonbasic variable is forced to enter the basis as it is pushed toward a bound.

Example

```
itcnt = CPXgetcrosspexchcnt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns the primal exchange iteration count if a solution exists. If no solution exists, it returns zero.

Global function CPXNETgetlb

`int` **`CPXNETgetlb`**`(CPXCENVptr env, CPXCNETptr net, double * low, int begin, int end)`

Definition file: cplex.h

The routine `CPXNETgetlb` is used to access the lower capacity bounds for a range of arcs of the network stored in a network problem object.

Example

```
status = CPXNETgetlb (env, net, low, 0, cur_narcs-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
low	Array in which to write the lower bound on the flow for the requested arcs. If NULL is passed, no lower bounds are retrieved. Otherwise, the size of the array must be (`end-begin+1`).
begin	Index of the first arc for which lower bounds are to be obtained.
end	Index of the last arc for which lower bounds are to be obtained.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXcompletelp

`int` **`CPXcompletelp`**`(CPXCENVptr env, CPXLPptr lp)`

Definition file: cplex.h

The routine `CPXcompletelp` is provided to allow users to handle those rare cases where modification steps need to be closely managed; for example, when careful timings are desired for the individual steps in a user's solution process, or more control of memory allocations for problem modifications is needed.

Example

```
status = CPXcompletelp (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXuncrushpi

```
int CPXuncrushpi(CPXCENVptr env, CPXCLPptr lp, double * pi, const double * prepi)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXuncrushpi` uncrushes a dual solution for the presolved problem to a dual solution for the original problem. This routine is for linear programs. Use `CPXqpuncrushpi` for quadratic programs.

Example

```
status = CPXuncrushpi (env, lp, pi, prepi);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
pi	An array to receive dual solution (`pi`) values for the original problem as computed from the dual values of the presolved problem object. The array must be of length at least the number of rows in the LP problem object.
prepi	An array that contains dual solution (`pi`) values for the presolved problem, as returned by routines such as `CPXgetpi` and `CPXsolution` when applied to the presolved problem object. The array must be of length at least the number of rows in the presolved problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsolnpoolsolnindex

```
int CPXgetsolnpoolsolnindex(CPXCENVptr env, CPXCLPptr lp, const char * lname_str,
int * index_p)
```

Definition file: cplex.h

The routine `CPXgetsolnpoolsolnindex` searches for the index number of the specified solution in the solution pool of a CPLEX problem object.

Example

```
status = CPXgetsolnpoolsolnindex (env, lp, "p4", &setindex);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lname_str	A solution name to search for.
index_p	A pointer to an integer to hold the index number of the solution with name `lname_str`. If the routine is successful, `*index_p` contains the index number; otherwise, `*index_p` is undefined.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXdelsetmipstarts

```
int CPXdelsetmipstarts(CPXCENVptr env, CPXLPptr lp, int * delstat)
```

Definition file: cplex.h

The routine `CPXdelsetmipstarts` deletes a set of MIP starts. Unlike the routine `CPXdelmipstarts`, `CPXdelsetmipstarts` does not require the MIP starts to be in a contiguous range. After the deletion occurs, the remaining MIP starts are indexed consecutively starting at 0, and in the same order as before the deletion.

Note

The `delstat` array must have at least `CPXgetnummipstarts(env, lp)` elements.

Example

```
status = CPXdelsetmipstarts (env, lp, delstat);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
delstat	An array specifying the MIP starts to be deleted. The routine `CPXdelsetmipstarts` deletes each MIP start `i` for which `delstat[i] = 1`. The deletion of MIP starts results in a renumbering of the remaining MIP starts. After termination, `delstat[i]` is either -1 for MIP starts that have been deleted or the new index number that has been assigned to the remaining MIP starts.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetprestat

```
int CPXgetprestat(CPXCENVptr env, CPXCLPptr lp, int * prestat_p, int * pcstat, int
* prstat, int * ocstat, int * orstat)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetprestat` accesses presolve status information for the columns and rows of the presolved problem in the original problem and of the original problem in the presolved problem.

Table 1: Value of prestat_p

0	lp is not presolved or there were no reductions
1	lp has a presolved problem
2	lp was reduced to an empty problem

For variable `i` in the original problem, values for `pcstat[i]` appear in Table 2.

Table 2: Values for pcstat[i]

	>= 0	variable `i` corresponds to variable `pcstat[i]` in the presolved problem
`CPX_PRECOL_LOW`	-1	variable `i` is fixed to its lower bound
`CPX_PRECOL_UP`	-2	variable `i` is fixed to its upper bound
`CPX_PRECOL_FIX`	-3	variable `i` is fixed to some other value
`CPX_PRECOL_AGG`	-4	variable `i` is aggregated out
`CPX_PRECOL_OTHER`	-5	variable `i` is deleted or merged for some other reason

For row i in the original problem, values for `prstat[i]` appear in Table 3.

Table 3: Values for prstat[i]

	>= 0	row `i` corresponds to row `prstat[i]` in the original problem
`CPX_PREROW_RED`	-1	if row `i` is redundant
`CPX_PREROW_AGG`	-2	if row `i` is used for aggregation
`CPX_PREROW_OTHER`	-3	if row `i` is deleted for some other reason

For variable i in the presolved problem, values for `ocstat[i]` appear in Table 4.

Table 4: Values for ocstat[i]

>= 0	variable `i` in the presolved problem corresponds to variable `ocstat[i]` in the original problem.
-1	variable `i` corresponds to a linear combination of some variables in the original problem.

For row i in the original problem, values for `orstat[i]` appear in Table 5.

Table 5: Values for orstat

>= 0	if row `i` in the presolved problem corresponds to row `orstat[i]` in the original problem
-1	if row `i` is created by, for example, merging two rows in the original problem.

Example

```
status = CPXgetprestat (env, lp, &presolvestat,
                        precstat, prerstat,
                        origcstat, origrstat);
```

See also `admipex6.c` in the *CPLEX User's Manual*.

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to the original CPLEX LP problem object, as returned by `CPXcreateprob`.
prestat_p	A pointer to an integer that will receive the status of the presolved problem associated with LP problem object lp. May be NULL.
pcstat	The array where the presolve status values of the columns are to be returned. The array must be of length at least the number of columns in the original problem object. May be NULL.
prstat	The array where the presolve status values of the rows are to be returned. The array must be of length at least the number of rows in the original problem object. May be NULL.
ocstat	The array where the presolve status values of the columns of the presolved problem are to be returned. The array must be of length at least the number of columns in the presolved problem object. May be NULL.
orstat	The array where the presolve status values of the rows of the presolved problem are to be returned. The array must be of length at least the number of rows in the presolved problem object. May be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXreadcopymipstart

```
int CPXreadcopymipstart(CPXCENVptr env, CPXLPptr lp, const char * filename_str)
```

Definition file: cplex.h

This routine is **deprecated**. Use `CPXreadcopymipstarts` instead.

The routine `CPXreadcopymipstart` reads a `MST` file and copies the information of the first MIP start contained in this file into a CPLEX problem object. The parameter `CPX_PARAM_ADVIND` must be set to 1 (one), its default value, or 2 (two) in order for the MIP start to be used.

Example

```
status = CPXreadcopymipstart(env, lp, "myprob.mst");
```

See Also: CPXmstwrite, CPXreadcopymipstarts

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A string containing the name of the `MST` file.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXbranchcallbackbranchgeneral

```
int CPXbranchcallbackbranchgeneral(CPXCENVptr env, void * cbdata, int wherefrom,
double nodeest, int varcnt, const int * varind, const char * varlu, const int *
varbd, int rcnt, int nzcnt, const double * rhs, const char * sense, const int *
rmatbeg, const int * rmatind, const double * rmatval, void * userhandle, int *
seqnum_p)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXbranchcallbackbranchgeneral` specifies the branches to be taken from the current node when the branch includes variable bound changes and additional constraints. It may be called only from within a user-written branch callback function.

Branch variables are in terms of the original problem if the parameter `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, branch variables are in terms of the presolved problem.

Table 1: Values of varlu[i]

L	change the lower bound
U	change the upper bound
B	change both upper and lower bounds

Table 2: Values of sense[i]

L	less than or equal to constraint
E	equal to constraint
G	greater than or equal to constraint

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	A pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value that reports where the user-written callback was called from. This argument must be the value of `wherefrom` passed to the user-written callback.
nodeest	A double that specifies the value of the node estimate for the node to be created with this branch. The node estimate is used to select nodes from the branch-and-cut tree with certain values of the node selection parameter `CPX_PARAM_NODESEL`.
varcnt	An integer that specifies the number of bound changes that are specified in the arrays `varind`, `varlu`, and `varbd`.
varind	Together with var`lu` and var`bd`, this array defines the bound changes for the branch. The entry `varind[i]` is the index for the variable.
varlu	Together with `varind` and `varbd`, this array defines the bound changes for the branch. The entry `varlu[i]` is one of three possible values specifying which bound to change. Those values appear in Table 1.

varbd	Together with `varind` and `varlu`, this array defines the bound changes for the branch. The entry `varbd[i]` specifies the new value of the bound.
rcnt	An integer that specifies the number of constraints for the branch.
nzcnt	An integer that specifies the number of nonzero constraint coefficients for the branch. This specifies the length of the arrays `rmatind` and `rmatval`.
rhs	An array of length `rcnt` containing the righthand side term for each constraint for the branch.
sense	An array of length `rcnt` containing the sense of each constraint to be added for the branch. Possible values appear in Table 2.
rmatbeg	An array that with `rmatbeg` and `rmatind` defines the constraints for the branch.
rmatind	An array that with `rmatbeg` and `rmatind` defines the constraints for the branch.
rmatval	An array that with `rmatbeg` and `rmatind` defines the constraints for the branch. The format is similar to the format used to describe the constraint matrix in the routine `CPXaddrows`. Every row must be stored in sequential locations in this array from position `rmatbeg[i]` to `rmatbeg[i+1]-1` (or from `rmatbeg[i]` to `nzcnt -1` if `i=rcnt-1`). Each entry, `rmatind[i]`, specifies the column index of the corresponding coefficient, `rmatval[i]`. All rows must be contiguous, and `rmatbeg[0]` must be `0`.
userhandle	A pointer to user private data that should be associated with the node created by this branch. May be NULL.
seqnum_p	A pointer to an integer that, on return, will contain the sequence number that CPLEX has assigned to the node created from this branch. The sequence number may be used to select this node in later calls to the node callback.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETchgarcnodes

`int` **`CPXNETchgarcnodes`**`(CPXCENVptr env, CPXNETptr net, int cnt, const int * indices, const int * fromnode, const int * tonode)`

Definition file: cplex.h

The routine `CPXNETchgarcnodes` changes the nodes associated with a set of arcs in the network stored in a network problem object.

Any solution information stored in the problem object is lost.

Example

```
status = CPXNETchgarcs (env, net, cnt, indices, newfrom, newto);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
cnt	Number of arcs to change.
indices	An array of arc indices that indicate the arcs to be changed. This array must have a length of at least `cnt`. All indices must be in the range [`0, narcs-1`].
fromnode	An array of from-node indices. The from-node for each arc listed in `indices` is changed to the corresponding value from this array. All node indices must be in the range [`0, nnodes-1`]. The size of the array must be at least `cnt`.
tonode	An array of to-node indices. The to-node for each arc listed in `indices` is changed to the corresponding value from this array. All node indices must be in the range [`0, nnodes-1`]. The size of the array must be at least `cnt`.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetsiftphase1cnt

`int` **`CPXgetsiftphase1cnt`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetsiftphase1cnt` accesses the number of Phase I sifting iterations to solve an LP problem.

Example

```
itcnt = CPXgetsiftphase1cnt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object as returned by `CPXcreateprob`.

Returns:

The routine returns the Phase I iteration count if a solution exists. It returns zero if no solution exists or any other type of error occurs.

Global function CPXgetcoef

```
int CPXgetcoef(CPXCENVptr env, CPXCLPptr lp, int i, int j, double * coef_p)
```

Definition file: cplex.h

The routine `CPXgetcoef` accesses a single constraint matrix coefficient of a CPLEX problem object. The row and column indices must be specified.

Example

```
status = CPXgetcoef (env, lp, 10, 20, &coef);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
i	An integer specifying the numeric index of the row.
j	An integer specifying the numeric index of the column.
coef_p	A pointer to a double to contain the specified matrix coefficient.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetgrad

```
int CPXgetgrad(CPXCENVptr env, CPXCLPptr lp, int j, int * head, double * y)
```

Definition file: cplex.h

The routine `CPXgetgrad` can be used, after an LP has been solved and a basis is available, to access information useful for different types of post-solution analysis. `CPXgetgrad` provides two arrays that can be used to project the impact of making changes to optimal variable values or objective function coefficients.

For a unit change in the value of the `j`th variable, the value of the `i`th basic variable, sometimes referred to as the variable basic in the `i`th row, changes by the amount `y[i]`. Also, for a unit change of the objective function coefficient of the `i`th basic variable, the reduced-cost of the `j`th variable changes by the amount `y[i]`. The vector `y` is equal to the product of the inverse of the basis matrix and the column `j` of the constraint matrix. Thus, `y` can be thought of as the representation of the `j`th column in terms of the basis.

Example

```
status = CPXgetgrad (env, lp, 13, head, y);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
j	An integer specifying the index of the column of interest. A negative value for `j` specifies a column representing the slack or artificial variable for row `-j-1`
head	An array to contain a listing of the indices of the basic variables in the order in which they appear in the basis. This listing is sometimes called the basis header. The `i`th entry in this list is also sometimes viewed as the variable in the `i`th row of the basis. If the `i`th basic variable is a structural variable, `head[i]` simply contains the column index of that variable. If it is a slack variable, `head[i]` contains one less than the negative of the row index of that slack variable. This array should be of length at least `CPXgetnumrows(env,lp)`. May be NULL.
y	An array to contain the coefficients of the `j`th column relative to the current basis. See the discussion above on how to interpret the entries in `y`. This array should be of length at least `CPXgetnumrows(env,lp)`. May be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs. This routine fails if no basis exists.

Global function CPXdelsetcols

`int` **`CPXdelsetcols`**`(CPXCENVptr env, CPXLPptr lp, int * delstat)`

Definition file: cplex.h

The routine `CPXdelsetcols` deletes a set of columns from a CPLEX problem object. Unlike the routine `CPXdelcols`, `CPXdelsetcols` does not require the columns to be in a contiguous range. After the deletion occurs, the remaining columns are indexed consecutively starting at 0, and in the same order as before the deletion.

Note

The `delstat` array must have at least `CPXgetnumcols(env,lp)` elements.

Example

```
status = CPXdelsetcols (env, lp, delstat);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
delstat	An array specifying the columns to be deleted. The routine `CPXdelsetcols` deletes each column `j` for which `delstat[j] = 1`. The deletion of columns results in a renumbering of the remaining columns. After termination, `delstat[j]` is either -1 for columns that have been deleted or the new index number that has been assigned to the remaining columns.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetcallbackorder

```
int CPXgetcallbackorder(CPXCENVptr env, void * cbdata, int wherefrom, int *
priority, int * direction, int begin, int end)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbackorder` retrieves MIP priority order information during MIP optimization from within a user-written callback. The values are from the original problem if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`. Otherwise, they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following values:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

Example

```
status = CPXgetcallbackorder (env, cbdata, wherefrom,
                              priority, NULL, 0, cols-1);
```

Branching direction

`CPX_BRANCH_GLOBAL`	0	use global branching direction setting `CPX_PARAM_BRDIR`
`CPX_BRANCH_DOWN`	-1	branch down first on variable `j+begin`
`CPX_BRANCH_UP`	1	branch up first on variable `j+begin`

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting from where the user-written callback was called. The argument must be the value of `wherefrom` passed to the user-written callback.
priority	An array where the priority values are to be returned. This array must be of length at least (`end - begin + 1`). If successful, `priority[0]` through `priority[end-begin]` contain the priority order values. May be NULL.
direction	An array where the preferred branch directions are to be returned. This array must be of length at least (`end - begin + 1`). The value of direction[j] will be a value from the table of branching directions. May be NULL.
begin	An integer specifying the beginning of the range of priority order information to be returned.
end	An integer specifying the end of the range of priority order information to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsolnpoolnumsolns

```
int CPXgetsolnpoolnumsolns(CPXCENVptr env, CPXCLPptr lp)
```

Definition file: cplex.h

Returns the number of solutions in the pool.

The routine `CPXgetsolnpoolnumsolns` accesses the number of solutions in the solution pool in the problem object.

Example

```
numsolns = CPXgetsolnpoolnumsolns (env, lp);
```

See also the example `populate.c` in the in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If the CPLEX problem object or environment does not exist, `CPXgetsolnpoolnumsolns` returns the value 0 (zero); otherwise, it returns the number of solutions.

Global function CPXdelchannel

`void` **`CPXdelchannel`**`(CPXENVptr env, CPXCHANNELptr * channel_p)`

Definition file: cplex.h

The routine `CPXdelchannel` flushes all message destinations for a channel, clears the message destination list, and frees the memory allocated to the channel. On completion, the pointer to the channel is set to NULL.

Example

```
CPXdelchannel (env, &mychannel);
```

See also `lpex5.c` in the *CPLEX User's Manual*.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
channel_p	A pointer to the pointer to the channel containing the message destinations to be flushed, cleared, and destroyed.

Returns:

This routine does not have a return value.

Global function CPXfputs

```
int CPXfputs(const char * s_str, CPXFILEptr stream)
```

Definition file: cplex.h

The routine `CPXfputs` can be used to write output to a file opened with `CPXfopen`. The purpose of this routine is to allow user-defined output in a file to be interspersed with the output created by using the routines `CPXaddfpdest` or `CPXsetlogfile`. The syntax of `CPXfputs` is the same as the standard C library function `fputs`.

Example

```
CPXfputs ("Solved first problem.n", fp);
```

Parameters:

s_str	A pointer to a string to be output to the file.
stream	A pointer to a file opened by the routine `CPXfopen`.

Returns:

This routine returns a nonnegative value if successful. Otherwise, it returns the system constant EOF (end of file).

Global function CPXgetlb

`int` **`CPXgetlb`**`(CPXCENVptr env, CPXCLPptr lp, double * lb, int begin, int end)`

Definition file: cplex.h

The routine `CPXgetlb` accesses a range of lower bounds on the variables of a CPLEX problem object. The beginning and end of the range must be specified.

Unbounded Variables

If a variable lacks a lower bound, then `CPXgetlb` returns a value less than or equal to `-CPX_INFBOUND`.

Example

```
status = CPXgetlb (env, lp, lb, 0, cur_numcols-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lb	An array where the specified lower bounds on the variables are to be returned. This array must be of length at least (`end - begin + 1`). The lower bound of variable `j` is returned in `lb[j - begin]`.
begin	An integer specifying the beginning of the range of lower bounds to be returned.
end	An integer specifying the end of the range of lower bounds to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsolnpoolmeanobjval

```
int CPXgetsolnpoolmeanobjval(CPXCENVptr env, CPXCLPptr lp, double * meanobjval_p)
```

Definition file: cplex.h

The routine `CPXgetsolnpoolmeanobjval` accesses the the mean objective value for solutions in the pool.

Example

```
status = CPXgetsolnpoolmeanobjval (env, lp, &meanobjval);
```

See also the example `populate.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns zero if successful and nonzero if the solution pool does not exist.

Global function CPXpivot

```
int CPXpivot(CPXCENVptr env, CPXLPptr lp, int jenter, int jleave, int leavestat)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXpivot` performs a basis change where variable `jenter` replaces variable `jleave` in the basis.

Use the constant `CPX_NO_VARIABLE` for `jenter` or for `jleave` if you want CPLEX to determine one of the two variables involved in the basis change.

It is invalid to pass a basic variable for `jenter`. Also, no nonbasic variable may be specified for `jleave`, except for `jenter == jleave` when the variable has both finite upper and lower bounds. In that case, the variable is moved from the current to the other bound. No shifting or perturbation is performed.

Example

```
status = CPXpivot (env, lp, jenter, jleave, CPX_AT_LOWER);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by the `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
jenter	An index specifying the variable to enter the basis. The slack or artificial variable for row `i` is denoted by `jenter = -i-1`. The argument `jenter` must either identify a nonbasic variable or take the value `CPX_NO_VARIABLE`. When `jenter` is set to `CPX_NO_VARIABLE`, CPLEX will use the leaving variable `jleave` to perform a dual simplex method ratio test that determines the entering variable.
jleave	An index specifying the variable to leave the basis. The slack or artificial variable for row `i` is denoted by `jleave = -i-1`. The argument `jleave` typically identifies a basic variable. However, if `jenter` denotes a variable with finite upper and lower bounds, `jleave` may be set to `jenter` to specify that the variable moves from its current bound to the other. The argument `jleave` may also be set to `CPX_NO_VARIABLE`. In that case, CPLEX will use the incoming variable `jenter` to perform a primal simplex method ratio test that determines the leaving variable.
leavestat	An integer specifying the nonbasic status to be assigned to the leaving variable after the basis change. This is important for the case where `jleave` specifies a variable with finite upper and lower bounds, as it may become nonbasic at its lower or upper bound.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXchgcolname

```
int CPXchgcolname(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices, char
** newname)
```

Definition file: cplex.h

The routine `CPXchgcolname` changes the names of variables in a CPLEX problem object. If this routine is performed on a problem object with no variable names, default names are created before the change is made.

See Also: CPXdelnames

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt	An integer that specifies the total number of variable names to be changed. Thus `cnt` specifies the length of the arrays `indices` and `newname`.
indices	An array of length `cnt` containing the numeric indices of the variables for which the names are to be changed.
newname	An array of length `cnt` containing the strings of the new variable names for the columns specified in `indices`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXinfostrparam

```
int CPXinfostrparam(CPXCENVptr env, int whichparam, char * defvalue_str)
```

Definition file: cplex.h

The routine `CPXinfostrparam` obtains the default value of a CPLEX string parameter

Example

```
status = CPXinfostrparam (env, CPX_PARAM_WORKDIR, defdirname);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
whichparam	The symbolic constant (or reference number) of the parameter for which the default value is to be obtained.
defvalue_str	A pointer to a buffer of length at least `CPX_STR_PARAM_MAX` to hold the default value of the CPLEX parameter.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetitcnt

```
int CPXgetitcnt(CPXCENVptr env, CPXCLPptr lp)
```

Definition file: cplex.h

The routine `CPXgetitcnt` accesses the total number of simplex iterations to solve an LP problem, or the number of crossover iterations in the case that the barrier optimizer is used.

Example

```
itcnt = CPXgetitcnt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If a solution exists, CPXgetitcnt returns the total iteration count. If no solution exists, CPXgetitcnt returns the value 0.

See `lpex6.c` in the *CPLEX User's Manual*.

Global function CPXgetbaritcnt

`int` **`CPXgetbaritcnt`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetbaritcnt` accesses the total number of Barrier iterations to solve an LP problem.

Example

```
itcnt = CPXgetbaritcnt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns the total iteration count if a solution exists. It returns zero if no solution exists or any other type of error occurs.

Global function CPXgetcutoff

```
int CPXgetcutoff(CPXCENVptr env, CPXCLPptr lp, double * cutoff_p)
```

Definition file: cplex.h

The routine `CPXgetcutoff` accesses the MIP cutoff value being used during mixed integer optimization. The `cutoff` is updated with the objective function value, each time an integer solution is found during branch and cut.

Example

```
status = CPXgetcutoff (env, lp, &cutoff);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cutoff_p	A pointer to a location where the value of the `cutoff` is returned.

Example

```
status = CPXgetcutoff (env, lp, &cutoff);
```

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETchgobjsen

`int` **`CPXNETchgobjsen`**`(CPXCENVptr env, CPXNETptr net, int maxormin)`

Definition file: cplex.h

The routine `CPXNETchgobjsen` is used to change the sense of the network problem to a minimization or maximization problem.

Any solution information stored in the problem object is lost.

Changed optimization sense in a network problem

`CPX_MAX`	For a maximization problem.
`CPX_MIN`	For a minimization problem.

Example

```
status = CPXNETchgobjsen (env, net, CPX_MAX);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
maxormin	New optimization sense for the network problem. The possible values are in the table.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetsolnpoolfilterindex

```
int CPXgetsolnpoolfilterindex(CPXCENVptr env, CPXCLPptr lp, const char * lname_str,
int * index_p)
```

Definition file: cplex.h

The routine `CPXgetsolnpoolfilterindex` searches for the index number of the specified filter of a CPLEX problem object.

Example

```
status = CPXgetsolnpoolfilterindex (env, lp, "p4", &setindex);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lname_str	A filter name to search for.
index_p	A pointer to an integer to hold the index number of the filter with name `lname_str`. If the routine is successful, `*index_p` contains the index number; otherwise, `*index_p` is undefined.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXdelfpdest

`int` **`CPXdelfpdest`**`(CPXCENVptr env, CPXCHANNELptr channel, CPXFILEptr fileptr)`

Definition file: cplex.h

The routine `CPXdelfpdest` removes a file from the list of message destinations for a channel. Failure occurs when the channel does not exist or the file pointer is not in the message destination list.

Example

```
CPXdelfpdest (env, mychannel, fileptr);
```

See `lpex5.c` in the *CPLEX User's Manual*.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
channel	The pointer to the channel for which destinations are to be deleted.
fileptr	A CPXFILEptr for the file to be removed from the destination list.

Returns:

The routines return zero if successful and nonzero if an error occurs.

Global function CPXgetorder

```
int CPXgetorder(CPXCENVptr env, CPXCLPptr lp, int * cnt_p, int * indices, int *
priority, int * direction, int ordspace, int * surplus_p)
```

Definition file: cplex.h

The routine `CPXgetorder` accesses all the MIP priority order information stored in a CPLEX problem object. A priority order is generated if there is no order and parameter `CPX_PARAM_MIPORDTYPE` is nonzero.

Note

If the value of `ordspace` is 0, then the negative of the value of `*surplus_p` returned specifies the length needed for the arrays indices, priority, and direction.

Example

```
status = CPXgetorder (env, lp, &listsize, indices, priority,
                      direction, numcols, &surplus);
```

Possible settings for direction

CPX_BRANCH_GLOBAL	(0)	use global branching direction setting `CPX_PARAM_BRDIR`
CPX_BRANCH_DOWN	(1)	branch down first on variable `indices[k]`
CPX_BRANCH_UP	(2)	branch up first on variable `indices[k]`

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt_p	A pointer to an integer to contain the number of order entries returned; i.e., the true length of the arrays `indices`, `priority`, and `direction`.
indices	An array where the indices of the variables in the order are to be returned. `indices[k]` is the index of the variable which is entry `k` in the order information.
priority	An array where the priority values are to be returned. The priority corresponding to the `indices[k]` is returned in `priority[k]`. May be NULL. If `priority` is not NULL, it must be of length at least `ordspace`.
direction	An array where the preferred branching directions are to be returned. The direction corresponding to `indices[k]` is returned in `direction[k]`. May be NULL. If `direction` is not NULL, it must be of length at least `ordspace`. Possible settings for `direction[k]` appear in the table.
ordspace	An integer specifying the length of the non-NULL arrays `indices`, `priority`, and `direction`. May be 0.
surplus_p	A pointer to an integer to contain the difference between `ordspace` and the number of entries in each of the arrays `indices`, `priority`, and `direction`. A nonnegative value of `*surplus_p` reports that the length of the arrays was sufficient. A negative value reports that the length was insufficient and that the routine could not complete its task. In this case, the routine `CPXgetorder` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of `*surplus_p` specifies the amount of insufficient space in the arrays.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` reports that insufficient space was available in the indices, priority, and direction arrays to hold the priority order information.

Global function CPXchgqpcoef

```
int CPXchgqpcoef(CPXCENVptr env, CPXLPptr lp, int i, int j, double newvalue)
```

Definition file: cplex.h

The routine `CPXchgqpcoef` changes the coefficient in the quadratic objective of a quadratic problem (QP) corresponding to the variable pair `(i,j)` to the value `newvalue`. If `i` is not equal to `j`, both `Q(i,j)` and `Q(j,i)` are changed to `newvalue`.

Example

```
status = CPXchgqpcoef (env, lp, 10, 12, 82.5);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
i	An integer that indicates the first variable number (row number in `Q`).
j	An integer that indicates the second variable number (column number in `Q`).
newvalue	The new coefficient value.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXreadcopysol

```
int CPXreadcopysol(CPXCENVptr env, CPXLPptr lp, const char * filename_str)
```

Definition file: cplex.h

The routine `CPXreadcopysol` reads a solution from a SOL format file, and copies that basis or solution into a CPLEX problem object. The solution is used to initiate a crossover from a barrier solution, to restart the simplex method with an advanced basis, or to specify variable values for a MIP start. The file may contain basis status values, primal values, dual values, or a combination of those values.

The parameter `CPX_PARAM_ADVIND` must be set to 1 (one), its default value, or 2 (two) in order for the start to be used for starting a subsequent optimization.

The SOL format is an XML format and is documented in the stylesheet `solution.xsl` and schema `solution.xsd` in the `include` directory of the CPLEX distribution. *CPLEX File Formats Reference Manual* also documents this format briefly.

Example

```
status = CPXreadcopysol (env, lp, "myprob.sol");
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	The name of the file from which the solution information should be read.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETdelnodes

`int` **`CPXNETdelnodes`**`(CPXCENVptr env, CPXNETptr net, int begin, int end)`

Definition file: cplex.h

The routine `CPXNETdelnodes` is used to remove a range of nodes from the network stored in a network problem object. The remaining nodes are renumbered starting at zero; their order is preserved. All arcs incident to the nodes that are deleted are also deleted from the network.

Any solution information stored in the problem object is lost.

Example

```
status = CPXNETdelnodes (env, net, 10, 20);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
begin	Index of the first node to be deleted.
end	Index of the last node to be deleted.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXcutcallbackadd

```
int CPXcutcallbackadd(CPXCENVptr env, void * cbdata, int wherefrom, int nzcnt,
double rhs, int sense, const int * cutind, const double * cutval, int purgeable)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXcutcallbackadd` adds cuts and lazy constraints to the current node LP subproblem during MIP branch and cut. This routine may be called only from within user-written cut callbacks; thus it may be called only when the value of its `wherefrom` argument is `CPX_CALLBACK_MIP_CUT`.

The cut may be for the original problem if the parameter `CPX_PARAM_MIPCBREDLP` was set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. In this case, the parameter `CPX_PARAM_PRELINEAR` should also be set to `CPX_OFF` (zero). Otherwise, the cut is used on the presolved problem.

Example

```
status = CPXcutcallbackadd (env,
                            cbdata,
                            wherefrom,
                            mynzcnt,
                            myrhs,
                            'L',
                            mycutind,
                            mycutval,
                            0);
```

See also the example `admipex5.c` in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value that reports where the user-written callback was called from. This argument must be the value of `wherefrom` passed to the user-written callback.
nzcnt	An integer value that specifies the number of coefficients in the cut, or equivalently, the length of the arrays `cutind` and `cutval`.
rhs	A double value that specifies the value of the righthand side of the cut.
sense	An integer value that specifies the sense of the cut.
cutind	An array containing the column indices of cut coefficients.
cutval	An array containing the values of cut coefficients.
purgeable	A Boolean value specifying whether CPLEX is allowed to purge the cut if CPLEX deems the cut ineffective.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetdnorms

`int` **`CPXgetdnorms`**`(CPXCENVptr env, CPXCLPptr lp, double * norm, int * head, int * len_p)`

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetdnorms` accesses the norms from the dual steepest edge. As in `CPXcopydnorms`, the argument `head` is an array of column or row indices corresponding to the array of norms. Column indices are indexed with nonnegative values. Row indices are indexed with negative values offset by 1 (one). For example, if `head[0] = -5`, `norm[0]` is associated with `row 4`.

See Also: CPXcopydnorms

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX LP problem object, as returned by `CPXcreateprob`.
norm	An array containing the dual steepest-edge norms in the ordered specified by `head[]`. The array must be of length at least equal to the number of rows in the LP problem object.
head	An array containing column or row indices. The allocated length of the array must be at least equal to the number of rows in the LP problem object.
len_p	A pointer to an integer that specifies the number of entries in both `norm[]` and `head[]`. The value assigned to the pointer `*len_p is` needed by the routine `CPXcopydnorms`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXbinvcol

```
int CPXbinvcol(CPXCENVptr env, CPXCLPptr lp, int j, double * x)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXbinvcol` computes the j-th column of the basis inverse.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
j	An integer that specifies the index of the column of the basis inverse to be computed.
x	An array containing the j-th column of **Binv** (the inverse of the matrix B). The array must be of length at least equal to the number of rows in the problem.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetcallbackgloballb

```
int CPXgetcallbackgloballb(CPXCENVptr env, void * cbdata, int wherefrom, double *
lb, int begin, int end)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbackgloballb` retrieves the best known global lower bound values during MIP optimization from within a user-written callback. The global lower bounds are tightened after a new incumbent is found, so the values returned by `CPXgetcallbacknodex` may violate these bounds at nodes where new incumbents have been found. The values are from the original problem if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`; otherwise, they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

Example

```
status = CPXgetcallbackgloballb (env, cbdata, wherefrom,
                                 glb, 0, cols-1);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting from where the user-written callback was called. The argument must be the value of `wherefrom` passed to the user-written callback.
lb	An array to receive the values of the global lower bound values. This array must be of length at least (`end - begin + 1`). If successful, `lb[0]` through `lb[end-begin]` contain the global lower bound values.
begin	An integer specifying the beginning of the range of lower bound values to be returned.
end	An integer specifying the end of the range of lower bound values to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetqconstrindex

`int` **`CPXgetqconstrindex`**`(CPXCENVptr env, CPXCLPptr lp, const char * lname_str, int * index_p)`

Definition file: cplex.h

The routine `CPXgetqconstrindex` searches for the index number of the specified quadratic constraint in a CPLEX problem object.

Example

```
status = CPXgetqconstrindex (env, lp, "resource89", &qconstrindex);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lname_str	A quadratic constraint name to search for.
index_p	A pointer to an integer to hold the index number of the quadratic constraint with name `lname_str`. If the routine is successful, `*index_p` contains the index number; otherwise, `*index_p` is undefined.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXNETgetprobname

```
int CPXNETgetprobname(CPXCENVptr env, CPXCNETptr net, char * buf_str, int bufspace,
int * surplus_p)
```

Definition file: cplex.h

The routine `CPXNETgetprobname` is used to access the name of the problem stored in a network problem object.

Example

```
status = CPXNETgetprobname (env, net, name, namesize, &surplus);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
buf_str	Buffer into which the problem name is copied.
bufspace	Size of the array `buf_str` in bytes.
surplus_p	Pointer to an integer in which the difference between `bufspace` and the number of characters required to store the problem name is returned. A nonnegative value indicates that `bufspace` was sufficient. A negative value indicates that it was insufficient. In that case, `CPXERR_NEGATIVE_SURPLUS` is returned and the negative value of `surplus_p` indicates the amount of insufficient space in the array `buf`.

Returns:

The routine returns zero on success and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` indicates that there was not enough space in the `buf` array to hold the name.

Global function CPXdelrows

```
int CPXdelrows(CPXCENVptr env, CPXLPptr lp, int begin, int end)
```

Definition file: cplex.h

The routine `CPXdelrows` deletes a range of rows. The range is specified using a lower and upper index that represent the first and last row to be deleted, respectively. The indices of the rows following those deleted are decreased by the number of deleted rows.

Example

```
status = CPXdelrows (env, lp, 10, 20);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
begin	An integer that specifies the numeric index of the first row to be deleted.
end	An integer that specifies the numeric index of the last row to be deleted.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetnumqconstrs

`int` **`CPXgetnumqconstrs`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetnumqconstrs` is used to access the number of quadratic constraints in a CPLEX problem object.

Example

```
cur_numqconstrs = CPXgetnumqconstrs (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If the problem object or environment does not exist, `CPXgetnumqconstrs` returns the value 0 (zero); otherwise, it returns the number of quadratic constraints.

Global function CPXgetijdiv

```
int CPXgetijdiv(CPXCENVptr env, CPXCLPptr lp, int * idiv_p, int * jdiv_p)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetijdiv` returns the index of the diverging row (that is, constraint) or column (that is, variable) when one of the CPLEX simplex optimizers terminates due to a diverging vector. This function can be called after an unbounded solution status for a primal simplex call or after an infeasible solution status for a dual simplex call.

If one of the CPLEX simplex optimizers has concluded that the LP problem object is unbounded, and if the diverging variable is a slack or ranged variable, `CPXgetijdiv` returns the index of the corresponding row in `*idiv_p`. Otherwise, `*idiv_p` is set to `-1`.

If one of the CPLEX simplex optimizers has concluded that the LP problem object is unbounded, and if the diverging variable is a normal, structural variable, `CPXgetijdiv` sets `*jdiv_p` to the index of that variable. Otherwise, `*jdiv_p` is set to `-1`.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX LP problem object, as returned by `CPXcreateprob`.
idiv_p	A pointer to an integer indexing the row of a diverging variable. If one of the CPLEX simplex optimizers has concluded that the LP problem object is unbounded, and if the diverging variable is a slack or ranged variable, `CPXgetijdiv` returns the index of the corresponding row in `*idiv_p`. Otherwise, `*idiv_p` is set to `-1`.
jdiv_p	A pointer to an integer indexing the column of a diverging variable. If one of the CPLEX simplex optimizers has concluded that the LP problem object is unbounded, and if the diverging variable is a normal, structural variable, `CPXgetijdiv` sets `*jdiv_p` to the index of that variable. Otherwise, `*jdiv_p` is set to `-1`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcopynettolp

`int` **`CPXcopynettolp`**`(CPXCENVptr env, CPXLPptr lp, CPXCNETptr net)`

Definition file: cplex.h

The routine `CPXcopynettolp` copies a network problem stored in a network problem object to a CPLEX problem object (as an LP). Any problem data previously stored in the CPLEX problem object is overridden.

Example

```
status = CPXcopynettolp (env, lp, net);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
net	A pointer to a CPLEX network problem object containing the network problem to be copied.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetindconstrname

```
int CPXgetindconstrname(CPXCENVptr env, CPXCLPptr lp, char * buf_str, int bufspace,
int * surplus_p, int which)
```

Definition file: cplex.h

The routine `CPXgetindconstrname` accesses the name of a specified indicator constraint of a CPLEX problem object.

> **Note**
>
> If the value of `bufspace` is 0, then the negative of the value of `*surplus_p` returned specifies the total number of characters needed for the array `buf_str`.

Example

```
status = CPXgetindconstrname (env, lp, indname, lenindname,
                              &surplus, 5);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
buf_str	A pointer to a buffer of size `bufspace`. May be NULL if `bufspace` is 0.
bufspace	An integer specifying the length of the array `buf_str`. May be 0.
surplus_p	A pointer to an integer to contain the difference between `bufspace` and the amount of memory required to store the indicator constraint name. A nonnegative value of `*surplus_p` reports that the length of the array `buf_str` was sufficient. A negative value reports that the length of the array was insufficient and that the routine could not complete its task. In this case, `CPXgetindconstrname` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `*surplus_p` specifies the amount of insufficient space in the array `buf_str`.
which	An integer specifying the index of the indicator constraint for which the name is to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` reports that insufficient space was available in the `buf_str` array to hold the indicator constraint name.

Global function CPXgetcallbacklp

```
int CPXgetcallbacklp(CPXCENVptr env, void * cbdata, int wherefrom, CPXCLPptr *
lp_p)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbacklp` retrieves the pointer to the MIP problem that is in use when the user-written callback function is called. It is the original MIP if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`; otherwise, it is the presolved MIP. To obtain information about the node LP associated with this MIP, use the following routines:

- `CPXgetcallbacknodeintfeas`
- `CPXgetcallbacknodelb`
- `CPXgetcallbacknodeub`
- `CPXgetcallbacknodex`
- `CPXgetcallbackgloballb`
- `CPXgetcallbackglobalub`

Each of those routines will return node information associated with the original MIP if `CPX_PARAM_MIPCBREDLP` is turned off (that is, set to `CPX_OFF`); otherwise, they return information associated with the presolved MIP.

In contrast, the function `CPXgetcallbacknodelp` returns a pointer to the node subproblem, which is an LP. Note that the setting of `CPX_PARAM_MIPCBREDLP` does not affect this `lp` pointer. Since CPLEX does not explicitly maintain an unpresolved node LP, the `lp` pointer will correspond to the presolved node LP unless CPLEX presolve has been turned off or CPLEX has made no presolve reductions at all. Generally, this pointer may be used only in CPLEX Callable Library query routines, such as `CPXsolution` or `CPXgetrows`.

The routine `CPXgetcallbacklp` may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

Example

```
status = CPXgetcallbacklp (env, cbdata, wherefrom, &origlp);
```

See also `admipex1.c`, `admipex2.c`, and `admipex3.c` in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting from where the user-written callback was called. The argument must be the value of `wherefrom` passed to the user-written callback.
lp_p	A pointer to a variable of type `CPXLPptr` to receive the pointer to the LP problem object, which is a MIP.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETbasewrite

```
int CPXNETbasewrite(CPXCENVptr env, CPXCNETptr net, const char * filename_str)
```

Definition file: cplex.h

The routine `CPXNETbasewrite` writes the current basis stored in a network problem object to a file in BAS format. If no arc or node names are available for the problem object, default names are used.

Example

```
status = CPXNETbasewrite (env, net, "netbasis.bas");
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
filename_str	Name of the basis file to write.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetsolnpoolobjval

`int` **`CPXgetsolnpoolobjval`**`(CPXCENVptr env, CPXCLPptr lp, int soln, double * objval_p)`

Definition file: cplex.h

The routine `CPXgetsolnpoolobjval` accesses the objective value for a solution in the solution pool.

Example

```
status = CPXgetsolnpoolobjval (env, lp, 0, &objval);
```

See also the example `populate.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
soln	An integer specifying the index of the solution pool member for which to return the objective value. A value of -1 specifies that the incumbent should be used instead of a solution pool member.
objval_p	A pointer to a variable of type `double` where the objective value is stored.

Returns:

The routine returns zero if successful and nonzero if the specified solution does not exist.

Global function CPXopenCPLEX

```
CPXENVptr CPXopenCPLEX(int * status_p)
```

Definition file: cplex.h

The routine `CPXopenCPLEX` initializes a CPLEX environment when accessing a license for CPLEX and works only if the computer is licensed for Callable Library use. The routine `CPXopenCPLEX` must be the first CPLEX routine called. The routine returns a pointer to a CPLEX environment. This pointer is used as an argument to every other nonadvanced CPLEX routine (except `CPXmsg`).

Example

```
env = CPXopenCPLEX (&status);
```

See `lpex1.c` in the *CPLEX User's Manual*.

Parameters:

status_p A pointer to an integer, where an error code is placed by this routine.

Returns:

A pointer to the CPLEX environment. If an error occurs (including licensing problems), the value NULL is returned. The reason for the error is returned in the variable `*status_p`. If the routine is successful, then `*status_p` is 0 (zero).

Global function CPXsetbranchnosolncallbackfunc

```
int CPXsetbranchnosolncallbackfunc(CPXENVptr env, int(CPXPUBLIC
*branchnosolncallback)(CALLBACK_BRANCH_ARGS), void * cbhandle)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXsetbranchnosolncallbackfunc` sets the callback function that will be called instead of the branch callback when there is a failure due to such situations as an iteration limit being reached, unboundedness being detected, numeric difficulties being encountered, while the node LP is being solved. In consequence of the failure, whether the node is feasible or infeasible cannot be known and thus CPLEX routines such as `CPXsolution` may fail. In this situation, CPLEX will attempt to fix some variables and continue.

These conditions are rare (except when the user has set a very low iteration limit), so it is acceptable to let CPLEX follow its default action in these cases.

Global function CPXaddmipstarts

```
int CPXaddmipstarts(CPXCENVptr env, CPXLPptr lp, int mcnt, int nzcnt, const int *
beg, const int * varindices, const double * values, const int * effortlevel, char
** mipstartname)
```

Definition file: cplex.h

The routine `CPXaddmipstarts` adds multiple MIP starts to a CPLEX problem object of type `CPXPROB_MILP`, `CPXPROB_MIQP`, or `CPXPROB_MIQCP`. It does **not** replace the existing MIP starts.

MIP start values may be specified for any subset of the integer or continuous variables in the problem. When optimization begins or resumes, CPLEX processes each MIP start to attempt to find a feasible MIP solution that is compatible with the set of values specified in the MIP start. The processing of each MIP start depends on the corresponding effort level:

- Level 0 (zero) CPX_MIPSTART_AUTO: Automatic, CPLEX decides.
- Level 1 (one) CPX_MIPSTART_CHECKFEAS: CPLEX checks the feasibility of the MIP start.
- Level 2 CPX_MIPSTART_SOLVEFIXED: CPLEX solves the fixed problem specified by the MIP start.
- Level 3 CPX_MIPSTART_SOLVEMIP: CPLEX solves a subMIP.
- Level 4 CPX_MIPSTART_REPAIR: CPLEX attempts to repair the MIP start if it is infeasible, according to the parameter that sets the frequency to try to repair infeasible MIP start, `CPX_PARAM_REPAIRTRIES`.

By default, CPLEX expends effort at level CPX_MIPSTART_REPAIR (4) for the first MIP start and at level CPX_MIPSTART_CHECKFEAS (1, one) for all other MIP starts. The user may change that level of effort. A user may specify a different level of effort for each MIP start, for example, differing levels of effort for a MIP start derived from the incumbent, for a MIP start derived from a solution in the solution pool, for a MIP start supplied by the user.

When a partial MIP start is provided and the effort level is at level CPX_MIPSTART_SOLVEMIP or higher, CPLEX tries to extend it to a complete solution by solving a MIP over the variables whose values are **not** specified in the MIP start. The parameter `CPX_PARAM_SUBMIPNODELIM` controls the amount of effort CPLEX expends in trying to solve this secondary MIP. If CPLEX is able to find a complete feasible solution, that solution becomes the incumbent. If the specified MIP start values are infeasible and the effort level is CPX_MIPSTART_REPAIR, these values are retained for use in a subsequent repair heuristic. See the description of the parameter `CPX_PARAM_REPAIRTRIES` for more information about this repair heuristic.

Use the routine `CPXchgmipstarts` to modify or extend existing MIP starts.

Example

```
status = CPXaddmipstarts (env, lp, mcnt, nzcnt, beg, varindices,
                          values, effortlevel, mipstartname);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
mcnt	An integer giving the number of MIP starts to be added. This specifies the length of the arrays `beg`, `effortlevel` and `mipstartname`.
nzcnt	An integer giving the number of variable values to be added. This specifies the length of the arrays `varindices` and `values`.
beg	An array of length `mcnt` used with `varindices` and `values`. `beg[0]` must be 0 (zero). The elements specific to each MIP start `i` must be stored in sequential locations in arrays `varindices` and `values` from position `beg[i]` to `beg[i+1]-1` (or from `beg[i]` to `nzcnt -1` if `i=mcnt-1`).
varindices	An array of length `nzcnt` containing the numeric indices of the columns corresponding to the variables which are assigned starting values.
values	

	An array of length `nzcnt` containing the values to use for the MIP starts. The entry `values[j]` is the value assigned to the variable `indices[j]`. An entry `values[j]` greater than or equal to `CPX_INFBOUND` specifies that no value is set for the variable `indices[j]`.
effortlevel	An array of length `mcnt`. The value `effortlevel[i]` specifies specifies the level of effort CPLEX should exert to solve the i-th MIP start. Can be NULL, in which case, CPLEX assigns an effort level of CPX_MIPSTART_AUTO to each MIP start.
mipstartname	An array of length `mcnt` which specifies the names of the MIP starts. Can be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsetcutcallbackfunc

```
int CPXsetcutcallbackfunc(CPXENVptr env, int(CPXPUBLIC
*cutcallback)(CALLBACK_CUT_ARGS), void * cbhandle)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXsetcutcallbackfunc` sets and modifies the user-written callback for adding cuts. The user-written callback is called by CPLEX during MIP branch and cut for every node that has an LP optimal solution with objective value below the cutoff and is integer infeasible. CPLEX also calls the callback when comparing an integer feasible solution, including one provided by a MIP start before any nodes exist, against lazy constraints.

The callback routine adds globally valid cuts to the LP subproblem. The cut may be for the original problem if the parameter `CPX_PARAM_MIPCBREDLP` was set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, the cut is for the presolved problem.

Within the user-written cut callback, the routine `CPXgetcallbacknodelp` and other query routines from the Callable Library access information about the subproblem. The routines `CPXgetcallbacknodeintfeas` and `CPXgetcallbacksosinfo` examine the status of integer entities.

The routine `CPXcutcallbackadd` adds cuts and lazy constraints to the current node LP subproblem during MIP branch and cut. Cuts added to the problem are first put into a cut pool, so they are not present in the subproblem LP until after the user-written cut callback is finished.

Any cuts that are duplicates of cuts already in the subproblem are not added to the subproblem. Cuts that are added remain part of all subsequent subproblems; there is no cut deletion. However, CPLEX will purge global cuts that would not ordinarily be removed by backtracking if those global cuts have been designated *purgeable* by the routine `CPXaddusercuts` and if CPLEX deems it useful to do so. See the routine `CPXaddusercuts` for more detail about this distinction between local and global cuts with respect to purging.

If cuts have been added, the subproblem is re-solved and evaluated, and, if the LP solution is still integer infeasible and not cut off, the cut callback is called again.

If the problem has names, user-added cuts have names of the form `unumber` where `number` is a sequence number among all cuts generated.

The parameter `CPX_PARAM_REDUCE` must be set to `CPX_PREREDUCE_PRIMALONLY (1)` or `CPX_PREREDUCE_NOPRIMALORDUAL (0)` if the constraints to be added in the callback are lazy constraints, that is, not implied by the constraints in the constraint matrix. The parameter `CPX_PARAM_PRELINEAR` must be set to 0 if the constraints to be added are in terms of the original problem and the constraints are valid cutting planes.

Example

```
status = CPXsetcutcallbackfunc(env, mycutfunc, mydata);
```

See also the example `admipex5.c` in the standard distribution.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`cutcallback`

The pointer to the current user-written cut callback. If no callback has been set, the pointer evaluates to NULL.

`cbhandle`

A pointer to user private data. This pointer is passed to the user-written cut callback.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle,
              int        *useraction_p);
```

CPLEX calls the cut callback when the LP subproblem for a node has an optimal solution with objective value below the cutoff and is integer infeasible.

Callback return value

The callback returns zero if successful and nonzero if an error occurs.

Callback arguments

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`cbdata`

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

`wherefrom`

An integer value reporting where in the optimization this function was called. It has the value `CPX_CALLBACK_MIP_CUT`.

`cbhandle`

A pointer to user private data.

`useraction_p`

A pointer to an integer specifying the action for CPLEX to take at the completion of the user callback. The table summarizes possible actions.

Actions to be Taken After a User-Written Cut Callback

Value	Symbolic Constant	Action
0	CPX_CALLBACK_DEFAULT	Use cuts as added
1	CPX_CALLBACK_FAIL	Exit optimization
2	CPX_CALLBACK_SET	Use cuts as added

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsetstrparam

```
int CPXsetstrparam(CPXENVptr env, int whichparam, const char * newvalue_str)
```

Definition file: cplex.h

The routine `CPXsetstrparam` sets the value of a CPLEX string parameter.

Example

```
status = CPXsetstrparam (env, CPX_PARAM_WORKDIR, "mydir");
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
whichparam	The symbolic constant (or reference number) of the parameter to change.
newvalue_str	The new value of the parameter. The maximum length of `newvalue_str`, including the NULL terminator (the character `'0'` or `char(0)`), is `CPX_STR_PARAM_MAX`, defined in `cplex.h`. Setting `newvalue_str` to a string longer than this results in an error.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsetintparam

`int` **`CPXsetintparam`**`(CPXENVptr env, int whichparam, int newvalue)`

Definition file: cplex.h

The routine `CPXsetintparam` sets the value of a CPLEX parameter of type `int`.

The *CPLEX Parameters Reference Manual* provides a list of parameters with their types, options, and default values.

Example

```
status = CPXsetintparam (env, CPX_PARAM_SCRIND, CPX_ON);
```

See also `lpex1.c` in the *CPLEX User's Manual*.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
whichparam	The symbolic constant or reference number of the parameter to change.
newvalue	The new value of the parameter.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXchgctype

```
int CPXchgctype(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices, const
char * xctype)
```

Definition file: cplex.h

The routine `CPXchgctype` changes the types of a set of variables of a CPLEX problem object. Several types can be changed at once, with each type specified by the index of the variable with which it is associated.

> **Note**
>
> If a variable is to be changed to binary, a call to `CPXchgbds` should also be made to change the bounds to 0 and 1.

Table 1: Values of elements of ctype

`CPX_CONTINUOUS`	`C`	make column `indices[j]` continuous
`CPX_BINARY`	`B`	make column `indices[j]` binary
`CPX_INTEGER`	`I`	make column `indices[j]` general integer
`CPX_SEMICONT`	`S`	make column `indices[j]` semi-continuous
`CPX_SEMIINT`	`N`	make column `indices[j]` semi-integer

Example

```
status = CPXchgctype (env, lp, cnt, indices, ctype);
```

Parameters:

env	A pointer to the CPLEX environment as returned by the `CPXopenCPLEX` routine.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt	An integer that states the total number of types to be changed, and thus specifies the length of the arrays `indices` and `ctype`.
indices	An array containing the numeric indices of the columns corresponding to the variables the types of which are to be changed.
xctype	An array containing characters that represent the new types for the columns specified in indices. Possible values for `ctype[j]` appear in Table 1.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsosinfeas

```
int CPXgetsosinfeas(CPXCENVptr env, CPXCLPptr lp, const double * x, double *
infeasout, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetsosinfeas` computes the infeasibility of a given solution for a range of special ordered sets (SOSs). The beginning and end of the range must be specified. This routine checks whether the SOS type 1 or SOS type 2 condition is satisfied but it does not check for integer feasibility in the case of integer variables. For each SOS, the infeasibility value returned is 0 (zero) if the SOS condition is satisfied and nonzero otherwise.

Example

```
status = CPXgetsosinfeas (env, lp, NULL, infeasout, 0, CPXgetnumsos(env,lp)-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
x	The solution whose infeasibility is to be computed. May be NULL, in which case the resident solution is used.
infeasout	An array to receive the infeasibility value for each of the special ordered sets. This array must be of length at least (`end - begin + 1`).
begin	An integer specifying the beginning of the range of special ordered sets whose infeasibility is to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETcreateprob

`CPXNETptr` **`CPXNETcreateprob`**`(CPXENVptr env, int * status_p, const char * name_str)`

Definition file: cplex.h

The routine `CPXNETcreateprob` constructs a new network problem object. The new object contains a minimization problem for a network with 0 (zero) nodes and 0 (zero) arcs. Other network problem data can be copied to a network with one of the routines `CPXNETaddnodes`, `CPXNETaddarcs`, `CPXNETcopynet`, `CPXNETextract`, or `CPXNETreadcopyprob`.

Example

```
CPXNETptr net = CPXNETcreateprob (env, &status, "mynet");
```

See Also: CPXNETaddnodes, CPXNETaddarcs, CPXNETcopynet, CPXNETextract, CPXNETreadcopyprob

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
status_p	A pointer to an integer used to return any error code produced by this routine.
name_str	Name of the network to be created.

Returns:

If the operation is successful, `CPXNETcreateprob` returns the newly constructed network problem object; if not, it returns either NULL or a nonzero value to indicate an error. In case of an error, the value pointed to by `status_p` contains an integer indicating the cause of the error.

Global function CPXgetcallbacknodex

```
int CPXgetcallbacknodex(CPXCENVptr env, void * cbdata, int wherefrom, double * x,
int begin, int end)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbacknodex` retrieves the primal variable (x) values for the subproblem at the current node during MIP optimization from within a user-written callback. The values are from the original problem if the parameter `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`; otherwise, they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`, or
- `CPX_CALLBACK_MIP_CUT`.

Example

```
status = CPXgetcallbacknodex (env, cbdata, wherefrom,
                              nodex, 0, cols-1);
```

See also `admipex1.c`, `admipex3.c`, and `admipex5.c` in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting from where the user-written callback was called. The argument must be the value of `wherefrom` passed to the user-written callback.
x	An array to receive the values of the primal variables for the node subproblem. This array must be of length at least (`end - begin + 1`). If successful, `x[0]` through `x[end-begin]` contain the primal values.
begin	An integer specifying the beginning of the range of primal variable values for the node subproblem to be returned.
end	An integer specifying the end of the range of primal variable values for the node subproblem to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsetmipcallbackfunc

```
int CPXsetmipcallbackfunc(CPXENVptr env, int(CPXPUBLIC *callback)(CPXCENVptr, void
*, int, void *), void * cbhandle)
```

Definition file: cplex.h

The routine `CPXsetmipcallbackfunc` sets the user-written callback routine to be called prior to solving each subproblem in the branch-and-cut tree, including the root node, during the optimization of a mixed integer program and during some cut generation routines.

This routine works in the same way as the routine `CPXsetlpcallbackfunc`. It enables the user to create a separate callback function to be called during the solution of mixed integer programming problems (MIPs).

The prototype for the callback function is identical to that of `CPXsetlpcallbackfunc`.

Example

```
status = CPXsetmipcallbackfunc (env, mycallback, NULL);
```

Parameters

`env`

A pointer to the CPLEX environment, as returned by one of the `CPXopenCPLEX` routines.

`callback`

A pointer to a user-written callback function. Setting `callback` to NULL will prevent any callback function from being called during optimization. The call to `callback` will occur after every node during optimization and during certain cut generation routines. This function must be written by the user. Its prototype is explained in the Callback description.

`cbhandle`

A pointer to user private data. This pointer will be passed to the callback function.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle);
```

This is the user-written callback routine.

Callback return value

A nonzero terminates the optimization.

Callback arguments

`env`

A pointer to the CPLEX environment that was passed into the associated optimization routine.

`cbdata`

A pointer passed from the optimization routine to the user-written callback function that identifies the problem being optimized. The only purpose for the `cbdata` pointer is to pass it to the routine `CPXgetcallbackinfo`.

`wherefrom`

An integer value reporting from which optimization algorithm the user-written callback function was called. Possible values and their meaning appear in the table.

Value	Symbolic Constant	Meaning
101	CPX_CALLBACK_MIP	From mipopt
107	CPX_CALLBACK_MIP_PROBE	From probing or clique merging
108	CPX_CALLBACK_MIP_FRACCUT	From Gomory fractional cuts
109	CPX_CALLBACK_MIP_DISJCUT	From disjunctive cuts
110	CPX_CALLBACK_MIP_FLOWMIR	From Mixed Integer Rounding cuts

`cbhandle`

A pointer to user private data as passed to `CPXsetmipcallbackfunc`.

See Also: CPXgetcallbackinfo, CPXsetlpcallbackfunc, CPXsetnetcallbackfunc

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETreadcopyprob

```
int CPXNETreadcopyprob(CPXCENVptr env, CPXNETptr net, const char * filename_str)
```

Definition file: cplex.h

The routine `CPXNETreadcopyprob` reads a network, in the CPLEX `.net` or DIMACS `.min` format, from a file and copies it to a network problem object. Any existing network or solution data in the problem object is replaced.

Example

```
status = CPXNETreadcopyprob (env, net, "network.net");
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
filename_str	Name of the network file to read.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXnewrows

```
int CPXnewrows(CPXCENVptr env, CPXLPptr lp, int rcnt, const double * rhs, const
char * sense, const double * rngval, char ** rowname)
```

Definition file: cplex.h

The routine `CPXnewrows` adds empty constraints to a specified CPLEX problem object. This routine may be called any time after a call to `CPXcreateprob`.

For each row, the user can specify the sense, righthand side value, range value and name of the constraint. The added rows are indexed to put them at the end of the problem. Thus, if `rcnt` rows are added to a problem object already having `k` rows, the new rows have indices `k`, `k+1`, ... `k+rcnt-1`. The constraint coefficients in the new rows are zero; the constraint coefficients can be changed with calls to `CPXchgcoef`, `CPXchgcoeflist` or `CPXaddcols`.

Table 1: Settings for elements of the array sense

`sense[i]`	`= 'L'`	<= constraint
`sense[i]`	`= 'E'`	= constraint
`sense[i]`	`= 'G'`	>= constraint
`sense[i]`	`= 'R'`	ranged constraint

Example

```
status = CPXnewrows (env, lp, rcnt, rhs, sense, NULL, newrowname);
```

See also the example `lpex1.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
rcnt	An integer that specifies the number of new rows to be added to the problem object.
rhs	An array of length `rcnt` containing the righthand side term for each constraint to be added to the problem object. May be NULL, in which case the righthand side terms are set to 0.0 for the new constraints.
sense	An array of length `rcnt` containing the sense of each constraint to be added to the problem object. This array may be NULL, in which case the sense of each constraint is set to `'E'` The values of the elements of this array appear in Table 1.
rngval	An array of length `rcnt` containing the range values for the new constraints. If a new constraint has `sense[i]='R'`, the value of constraint i can be between `rhs[i]` and `rhsi[i]+rngval[i]`. May be NULL, in which case the range values are all set to zero.
rowname	An array of length `rcnt` containing pointers to character strings that represent the names of the new rows, or equivalently, the constraint names. May be NULL, in which case the new rows are assigned default names if the rows already resident in the problem object have names; otherwise, no names are associated with the constraints. If row names are passed to `CPXnewrows` but existing constraints have no names assigned, default names are created for the existing constraints.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetobjoffset

`int` **`CPXgetobjoffset`**`(CPXCENVptr env, CPXCLPptr lp, double * objoffset_p)`

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetobjoffset` returns the objective offset between the original problem and the presolved problem.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a reduced CPLEX LP problem object, as returned by `CPXgetredlp`.
objoffset_p	A pointer to a variable of type `double` to hold the objective offset value.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXflushchannel

`void` **`CPXflushchannel`**`(CPXCENVptr env, CPXCHANNELptr channel)`

Definition file: cplex.h

The routine `CPXflushchannel` flushes (outputs and clears the buffers of) all message destinations for a channel. Use this routine in cases when it is important to have output written to disk immediately after it is generated. For most applications this routine need not be used.

Example

```
CPXflushchannel (env, mychannel);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
channel	A pointer to the channel containing the message destinations to be flushed.

Returns:

This routine does not return a value.

Global function CPXfeasoptext

```
int CPXfeasoptext(CPXCENVptr env, CPXLPptr lp, int grpcnt, int concnt, const double
* grppref, const int * grpbeg, const int * grpind, const char * grptype)
```

Definition file: cplex.h

The routine `CPXfeasoptext` extends `CPXfeasopt` in several ways. Unlike `CPXfeasopt`, `CPXfeasoptext` enables the user to relax quadratic constraints and indicator constraints. In addition, it allows the user to treat a group of constraints as a single constraint for the purposes of determining the penalty for relaxation.

Thus, according to the various INF relaxation penalty metrics (see `CPXfeasopt` for a list of the available metrics), all constraints in a group can be relaxed for a penalty of one unit. Similarly, according to the various QUAD metrics, the penalty of relaxing a group grows as the square of the sum of the individual member relaxations, rather than as the sum of the squares of the individual relaxations.

If you use INF mode, the resulting feasopt problems will be MIPs even if your problem is continuous. Similarly, if you use QUAD mode, the feasopt problems will become quadratic even if your original problem is linear. This difference can result in greater than expected solve times.

The routine also computes a relaxed solution vector that can be queried with `CPXsolution`, `CPXgetcolinfeas` for columns, `CPXgetrowinfeas` for rows, `CPXgetqconstrinfeas` for quadratic constraints, `CPXgetindconstrinfeas` for indicator constraints, or `CPXgetsosinfeas` for special ordered sets.

The arguments to this routine define the set of groups, Each group contains a list of member constraints, and each member has a type (lower bound, upper bound, linear constraint, quadratic constraint, or indicator constraint). The group members and member types are entered by means of a data structure similar to the sparse matrix data structure used throughout CPLEX. (See `CPXcopylp` for one example.) The argument `grpbeg` gives the starting location of each group in `grpind` and `grptype`. The list of members for group `i` can be found in `grpind[grpbeg[i]]` through `grpind[grpbeg[i+1]-1]`, for i less than `grpcnt-1` and `grpind[grpbeg[i]]` through `grpind[concnt-1]` for `i = grpcnt-1`. The corresponding constraint types for these members can be found in `grptype[grpbeg[i]]` through `grptype[grpbeg[i+1]-1]`, for i less than `concnt-1` and `grptype[grpbeg[grpcnt-1]]` through `grptype[concnt-1]` for `i = grpcnt-1`. A constraint can appear in at most one group. A constraint that appears in no group will not be relaxed.

Table 1: Possible values for elements of grptype

`CPX_CON_LOWER_BOUND`	`= 1`	variable lower bound
`CPX_CON_UPPER_BOUND`	`= 2`	variable upper bound
`CPX_CON_LINEAR`	`= 3`	linear constraint
`CPX_CON_QUADRATIC`	`= 4`	quadratic constraint
`CPX_CON_INDICATOR`	`= 6`	indicator constraint

The parameters `CPX_PARAM_CUTUP`, `CPX_PARAM_CUTLO`, `CPX_PARAM_OBJULIM`, `CPX_PARAM_OBJLLIM` do not influence this routine. If you want to study infeasibilities introduced by those parameters, consider adding an objective function constraint to your model to enforce their effect before you invoke this routine.

Parameters:

env	A pointer to the CPLEX environment as returned by the routine `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
grpcnt	The number of constraint groups to be considered.
concnt	An integer specifying the total number of indices passed in the array `grpind`, or, equivalently, the end of the last group in `grpind`.
grppref	An array of preferences for the groups. The value `grppref[i]` specifies the preference for the group

designated by the index `i`. A negative or zero value specifies that the corresponding group should not be relaxed.

grpbeg An array of integers specifying where the constraint indices for each group begin in the array `grpind`. Its length must be at least `grpcnt`.

grpind An array of integers containing the constraint indices for the constraints as they appear in groups. Group `i` contains the constraints with the indices `grpind[grpbeg[i]], ..., grpind[grpbeg[i+1]-1]` for `i` less than `grpcnt-1` and `grpind[grpbeg[i]], ..., grpind[concnt-1]` for `i == grpcnt-1`. Its length must be at least `concnt`, and a constraint must not be referenced more than once in this array. If a constraint does not appear in this array, the constraint will not be relaxed.

grptype An array of characters containing the constraint types for the constraints as they appear in groups. The types of the constraints in group `i` are specified in `grptype[grpbeg[i]], ..., grptype[grpbeg[i+1]-1]` for `i` less than `grpcnt-1` and `grptype[grpbeg[i]], ..., grptype[concnt-1]` for `i == grpcnt-1`. Its length must be at least `concnt`, and every constraint must appear at most once in this array. Possible values appear in Table 1.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetnumbin

```
int CPXgetnumbin(CPXCENVptr env, CPXCLPptr lp)
```

Definition file: cplex.h

The routine `CPXgetnumbin` accesses the number of binary variables in a CPLEX problem object.

Example

```
numbin = CPXgetnumbin (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Example

```
numbin = CPXgetnumbin (env, lp);
```

Returns:

If the problem object or environment does not exist, CPXgetnumbin returns zero. Otherwise, it returns the number of binary variables in the problem object.

Global function CPXhybbaropt

`int` **`CPXhybbaropt`**`(CPXCENVptr env, CPXLPptr lp, int method)`

Definition file: cplex.h

The routine `CPXhybbaropt` may be used, at any time after a linear program has been created via a call to `CPXcreateprob`, to find a solution to that problem. When this function is called, the specified problem is solved using CPLEX Barrier followed by an automatic crossover to a basic solution if barrier determines that the problem is both primal and dual feasible. Otherwise, crossover is not performed. In this case, a call to `CPXprimopt` or `CPXdualopt` can force a crossover to occur. The results of the optimization are recorded in the problem object.

Methods of CPXhybbaropt

`method`	= `0`	use `CPX_PARAM_BARCROSSALG` to choose a crossover method
`method`	= `CPX_ALG_PRIMAL`	primal crossover
`method`	= `CPX_ALG_DUAL`	dual crossover
`method`	= `CPX_ALG_NONE`	no crossover

Example

```
status = CPXhybbaropt (env, lp, CPX_ALG_PRIMAL);
```

See also the example `lpex2.c` in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
method	Crossover method to be implemented, according to the table.

Returns:

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines CPXsolninfo, CPXgetstat, and CPXsolution to obtain further information about the status of the optimization.

Global function CPXdisconnectchannel

`void` **`CPXdisconnectchannel`**`(CPXCENVptr env, CPXCHANNELptr channel)`

Definition file: cplex.h

The routine `CPXdisconnectchannel` flushes all message destinations associated with a channel and clears the corresponding message destination list.

Example

```
CPXdisconnectchannel (env, mychannel);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
channel	A pointer to the channel containing the message destinations to be flushed and cleared.

Returns:

This routine does not have a return value.

Global function CPXNETprimopt

`int` **`CPXNETprimopt`**`(CPXCENVptr env, CPXNETptr net)`

Definition file: cplex.h

The routine `CPXNETprimopt` can be called after a network problem has been copied to a network problem object, to find a solution to that problem using the primal network simplex method. When this function is called, the CPLEX primal network algorithm attempts to optimize the problem. The results of the optimization are recorded in the problem object and can be retrieved by calling the appropriate solution functions for that object.

Example

```
status = CPXNETprimopt (env, net);
```

See also the examples `netex1.c` and `netex2.c` in the standard distribution of the product.

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

net A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.

Returns:

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (CPXERR_NO_MEMORY) or encountering invalid data in the CPLEX problem object (CPXERR_NO_PROBLEM). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded, are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use query routines `CPXNETsolninfo`, `CPXNETgetstat`, and `CPXNETsolution` to obtain further information about the status of the optimization.

Global function CPXcopybasednorms

```
int CPXcopybasednorms(CPXCENVptr env, CPXLPptr lp, const int * cstat, const int *
rstat, const double * dnorm)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXcopybasednorms` works in conjunction with the routine `CPXgetbasednorms`. `CPXcopybasednorms` copies the values in the arrays `cstat`, `rstat`, and `dnorm`, as returned by `CPXgetbasednorms`, into a specified problem object.

Each of the arrays `cstat`, `rstat`, and `dnorm` must be non NULL. Only data returned by `CPXgetbasednorms` should be copied by `CPXcopybasednorms`. (Other details of `cstat`, `rstat`, and `dnorm` are not documented.)

> **Note**
>
> The routine `CPXcopybasednorms` should be called only if the return values of `CPXgetnumrows` and `CPXgetnumcols` have not changed since the companion call to `CPXgetbasednorms`. If either of these values has increased since that companion call, a memory violation may occur. If one of those values has decreased, the call will be safe, but its meaning will be undefined.

See Also: CPXgetbasednorms

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX LP problem object, as returned by `CPXcreateprob`.
cstat	An array containing the basis status of the columns in the constraint matrix returned by a call to `CPXgetbasednorms`. The length of the allocated array must be at least the value returned by `CPXgetnumcols`.
rstat	An array containing the basis status of the rows in the constraint matrix returned by a call to `CPXgetbasednorms`. The length of the allocated array must be at least the value returned by `CPXgetnumrows`.
dnorm	An array containing the dual steepest-edge norms returned by a call to `CPXgetbasednorms`. The length of the allocated array must be at least the value returned by `CPXgetnumrows`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsetdefaults

```
int CPXsetdefaults(CPXENVptr env)
```

Definition file: cplex.h

The routine `CPXsetdefaults` resets all CPLEX parameters and settings to default values (with the exception of the log file).

> **Note**
>
> This routine also resets the CPLEX callback functions to NULL.

The *CPLEX Parameters Reference Manual* provides a list of parameters with their types, options, and default values.

Example

```
status = CPXsetdefaults (env);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETchgname

`int` **`CPXNETchgname`**`(CPXCENVptr env, CPXNETptr net, int key, int vindex, const char * name_str)`

Definition file: cplex.h

The routine `CPXNETchgname` changes the name of a node or an arc in the network stored in a network problem object.

Values of key in CPXNETchgname

`key == 'a'`	Indicates the arc name is to be changed.
`key == 'n'`	Indicates the node name is to be changed.

Example

```
status = CPXNETchgname (env, net, 'a', 10, "arc10");
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
key	A character to indicate whether an arc name should be changed, or a node name should be changed.
vindex	The index of the arc or node whose name is to be changed.
name_str	The new name for the arc or node.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXNETaddarcs

```
int CPXNETaddarcs(CPXCENVptr env, CPXNETptr net, int narcs, const int * fromnode,
const int * tonode, const double * low, const double * up, const double * obj, char
** anames)
```

Definition file: cplex.h

The routine `CPXNETaddarcs` adds new arcs to the network stored in a network problem object.

Example

```
status = CPXNETaddarcs (env, net, narcs, fromnode, tonode, NULL,
                        NULL, obj, NULL);
```

See Also: CPXNETgetnumnodes

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
narcs	Number of arcs to be added.
fromnode	Array of indices of the from-node for the arcs to be added. All the indices must be greater than or equal to 0. If a node index is greater than or equal to the number of nodes currently in the network (see `CPXNETgetnumnodes`) new nodes are created implicitly with default supply values 0. The size of the `fromnode` array must be at least `narcs`.
tonode	Array of indices of the to-node for the arcs to be added. All the indices must be greater than or equal to 0. If a node index is greater than or equal to the number of nodes currently in the network (see `CPXNETgetnumnodes`) new nodes are created implicitly with default supply values 0. The size of the `tonode` array must be at least `narcs`.
low	Pointer to an array of lower bounds on the flow through added arcs. If NULL is passed, all lower bounds default to 0 (zero). Otherwise, the size of the array must be at least `narcs`. Values less than or equal to `-CPX_INFBOUND` are considered as negative infinity.
up	Pointer to an array of upper bounds on the flow of added arcs. If NULL is passed, all upper bounds default to `CPX_INFBOUND`. Otherwise, the size of the array must be at least `narcs`. Values greater than or equal to `CPX_INFBOUND` are considered as infinity.
obj	Pointer to an array of objective values for the added arcs. If NULL is passed, all objective values default to 0. Otherwise, the size of the array must be at least `narcs`.
anames	Pointer to an array of names for added arcs. If NULL is passed and the existing arcs have names, default names are assigned to the added arcs. If NULL is passed and the existing arcs have no names, the new arcs are assigned no names. Otherwise, the size of the array must be at least `narcs` and every name in the array must be a string terminating in 0. If the existing arcs have no names and `anames` is not NULL, default names are assigned to the existing arcs.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXhybnetopt

`int` **`CPXhybnetopt`**`(CPXCENVptr env, CPXLPptr lp, int method)`

Definition file: cplex.h

The routine `CPXhybnetopt`, given a linear program that has been created via a call to `CPXcreateprob`, extracts an embedded network, uses the CPLEX Network Optimizer to attempt to obtain an optimal basis to the network, and optimizes the entire linear program using one of the CPLEX simplex methods. CPLEX takes the network basis as input for the optimization of the whole linear program.

`method`	`= CPX_ALG_PRIMAL`	primal Simplex
`method`	`= CPX_ALG_DUAL`	dual Simplex

Example

```
status = CPXhybnetopt (env, lp, CPX_ALG_DUAL);
```

See also the example `lpex3.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
method	The type of simplex method to follow the network optimization.

Returns:

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (`CPXERR_NO_MEMORY`) or encountering invalid data in the CPLEX problem object (`CPXERR_NO_PROBLEM`).

Exceeding a user-specified CPLEX limit is not considered an error. Proving the problem infeasible or unbounded is not considered an error.

A zero return value does not necessarily mean that a solution exists. Use query routines `CPXsolninfo`, `CPXgetstat`, and `CPXsolution` to obtain further information about the status of the optimization.

Global function CPXrefinemipstartconflict

`int` **`CPXrefinemipstartconflict`**`(CPXCENVptr env, CPXLPptr lp, int mipstartindex, int * confnumrows_p, int * confnumcols_p)`

Definition file: cplex.h

The routine `CPXrefinemipstartconflict` refines a conflict in order to determine why a given MIP start is not feasible. In other words, this routine identifies a minimal conflict for the infeasibility of the linear constraints and bounds in a MIP start.

In order to analyze the infeasibility of a MIP start containing quadratic constraints, indicator constraints, or special ordered sets, as well as linear contraints and bounds, use the routine `CPXrefinemipstartconflictext`.

The given MIP start need not be complete; that is, it is not necessary to specify all the variables.

When this routine returns, the value in `confnumrows_p` specifies the number of constraints participating in the conflict, and the value in `confnumcols_p` specifies the number of variables participating in the conflict.

Use the routine `CPXgetconflict` to determine which constraints and variables participate in the conflict. Use the routine `CPXclpwrite` to write the conflict to a file.

This conflict is a submodel of the original model with the property that CPLEX cannot generate a solution from the chosen MIP start using the given level of effort and that removal of any constraint or bound in the conflict invalidates that property.

The parameters `CPX_PARAM_CUTUP`, `CPX_PARAM_CUTLO`, `CPX_PARAM_OBJULIM`, `CPX_PARAM_OBJLLIM` do not influence this routine. If you want to study infeasibilities introduced by those parameters, consider adding an objective function constraint to your model to enforce their effect before you invoke this routine.

When the MIP start was added to the current model, an effort level may have been associated with it to specify to CPLEX how much effort to expend in transforming the MIP start into a feasible solution. This routine respects effort levels **except** level 1 (one): check feasibility. It does not check feasibility.

Parameters:

env	A pointer to the CPLEX environment as returned by the routine `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
mipstartindex	The index of the MIP start among all the MIP starts associated with the problem.
confnumrows_p	A pointer to an integer where the number of linear constraints in the conflict is returned.
confnumcols_p	A pointer to an integer where the number of variable bounds in the conflict is returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetqconstrname

```
int CPXgetqconstrname(CPXCENVptr env, CPXCLPptr lp, char * buf_str, int bufspace,
int * surplus_p, int which)
```

Definition file: cplex.h

The routine `CPXgetqconstrname` is used to access the name of a specified quadratic constraint of a CPLEX problem object.

> **Note**
>
> If the value of `bufspace` is 0, then the negative of the value of `*surplus_p` returned indicates the total number of characters needed for the array `buf_str`.

Example

```
status = CPXgetqconstrname (env, lp, qname, lenqname,
                            &surplus, 5);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
buf_str	A pointer to a buffer of size `bufspace`. May be NULL if `bufspace` is 0.
bufspace	An integer indicating the length of the array `buf_str`. May be 0.
surplus_p	A pointer to an integer to contain the difference between `bufspace` and the amount of memory required to store the quadratic constraint name. A nonnegative value of `*surplus_p` indicates that the length of the array `buf_str` was sufficient. A negative value indicates that the length of the array was insufficient and that the routine could not complete its task. In this case, `CPXgetqconstrname` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `*surplus_p` indicates the amount of insufficient space in the array `buf_str`.
which	An integer indicating the index of the quadratic constraint for which the name is to be returned.

Returns:

The routine returns zero on success and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` indicates that insufficient space was available in the `buf_str` array to hold the quadratic constraint name.

Global function CPXcopystart

```
int CPXcopystart(CPXCENVptr env, CPXLPptr lp, const int * cstat, const int * rstat,
const double * cprim, const double * rprim, const double * cdual, const double *
rdual)
```

Definition file: cplex.h

The routine `CPXcopystart` provides starting information for use in a subsequent call to a simplex optimization routine (`CPXlpopt` with `CPX_PARAM_LPMETHOD` or `CPX_PARAM_QPMETHOD` set to `CPX_ALG_PRIMAL` or `CPX_ALG_DUAL`, `CPXdualopt`, `CPXprimopt`, or `CPXhybnetopt`). Starting information is not applicable to the barrier optimizer or the mixed integer optimizer.

When a basis (arguments `cstat` and `rstat`) is installed for a linear problem and `CPXlpopt` is used with `CPX_PARAM_LPMETHOD` set to `CPX_ALG_AUTOMATIC`, CPLEX will use the primal simplex algorithm if the basis is primal feasible and the dual simplex method otherwise.

Any of three different kinds of starting points can be provided: a starting basis (`cstat`, `rstat`), starting primal values (`cprim`, `rprim`), and starting dual values (`cdual`, `rdual`). Only a starting basis is applicable to a `CPXhybnetopt` call, but for Dual Simplex and Primal Simplex any combination of these three types of information can be of use in providing a starting point. If no starting-point is provided, this routine returns an error; otherwise, any resident starting information in the CPLEX problem object is freed and the new information is copied into it.

If you provide a starting basis, then both `cstat` and `rstat` must be specified. It is permissible to provide `cprim` with or without `rprim`, or `rdual` with or without `cdual`; arrays not being provided must be passed as NULL pointers.

Note

The starting information is ignored by the optimizers if the parameter `CPX_PARAM_ADVIND` is set to zero.

Table 1: Values for cstat[j]

`CPX_AT_LOWER`	0	variable at lower bound
`CPX_BASIC`	1	variable is basic
`CPX_AT_UPPER`	2	variable at upper bound
`CPX_FREE_SUPER`	3	variable free and nonbasic

Table 2: Values of rstat elements other than ranged rows

`CPX_AT_LOWER`	0	associated slack variable nonbasic at value 0.0
`CPX_BASIC`	1	associated slack artificial variable basic

Table 3: Values of rstat elements that are ranged rows

`CPX_AT_LOWER`	0	associated slack variable nonbasic at its lower bound
`CPX_BASIC`	1	associated slack variable basic
`CPX_AT_UPPER`	2	associated slack variable nonbasic at upper bound

Example

```
status = CPXcopystart (env,
                       lp,
                       cstat,
```

```
                    rstat,
                    cprim,
                    rprim,
                    cdual,
                    rdual);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cstat	An array containing the basis status of the columns in the constraint matrix. The length of the array is equal to the number of columns in the CPLEX problem object. If this array is NULL, `rstat` must be NULL. Table 1 shows the possible values.
rstat	An array containing the basis status of the slack, surplus, or artificial variable associated with each row in the constraint matrix. The length of the array is equal to the number of rows in the LP problem. For rows other than ranged rows, the array element `rstat[i]` can be set according to Table 2. For ranged rows, the array element `rstat[i]` can be set according to Table 3. If this array is NULL, `cstat` must be NULL.
cprim	An array containing the initial primal values of the column variables. The length of the array must be no less than the number of columns in the CPLEX problem object. If this array is NULL, `rprim` must be NULL.
rprim	An array containing the initial primal values of the slack (row) variables. The length of the array must be no less than the number of rows in the CPLEX problem object. This array may be NULL.
cdual	An array containing the initial values of the reduced costs for the column variables. The length of the array must be no less than the number of columns in the CPLEX problem object. This array may be NULL.
rdual	An array containing the initial values of the dual variables for the rows.The length of the array must be no less than the number of rows in the CPLEX problem object. If this array is NULL, `cdual` must be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetredlp

```
int CPXgetredlp(CPXCENVptr env, CPXCLPptr lp, CPXCLPptr * redlp_p)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetredlp` returns a pointer for the presolved problem. It returns NULL if the problem is not presolved or if all the columns and rows are removed by presolve. Generally, the returned pointer may be used only in CPLEX Callable Library query routines, such as `CPXsolution` or `CPXgetrows`.

The presolved problem must not be modified. Any modifications must be done on the original problem. If `CPX_PARAM_REDUCE` is set appropriately, the modifications are automatically carried out on the presolved problem at the same time. Optimization and query routines can be used on the presolved problem.

Example

```
status = CPXgetredlp (env, lp, &reducelp);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
redlp_p	A pointer to receive the problem object pointer that results when presolve has been applied to the LP problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetmipstarts

```
int CPXgetmipstarts(CPXCENVptr env, CPXCLPptr lp, int * nzcnt_p, int * beg, int *
varindices, double * values, int * effortlevel, int startspace, int * surplus_p,
int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetmipstarts` accesses a range of MIP starts of a CPLEX problem object. The beginning and end of the range, along with the length of the arrays in which the entries of these MIP starts are to be returned, must be specified.

Note

If the value of `startspace` is 0 (zero) then the negative of the value of `surplus_p` returned specifies the length needed for the arrays `varindices` and `values`.

Example

```
status = CPXgetmipstarts (env, lp, &nzcnt, beg, varindices,
                          values, effortlevel, startspace,
                          &surplus, 0, cur_numstarts-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by the `CPXopenCPLEX` routine.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
nzcnt_p	A pointer to an integer to contain the number of entries returned; that is, the true length of the arrays `varindices` and `values`.
beg	An array of length `(end - beg + 1)` specifying where each of the requested MIP starts begins in the arrays `varindices` and `values`. The elements specific to each MIP start `i` are stored in sequential locations in arrays `varindices` and `values` from position `beg[i]` to `beg[i+1]-1` (or from `beg[i]` to `nzcnt -1` if `i=end`).
varindices	An array of length `nzcnt` containing the numeric indices of the columns corresponding to the variables which are assigned starting values.
values	An array of length `nzcnt` containing the values of the MIP starts. The entry `values[j]` is the value assigned to the variable `indices[j]`.
effortlevel	An array of length `end-begin+1` containing the effort level for each MIP start requested.
startspace	An integer specifying the length of the arrays `varindices` and `values`. May be 0.
surplus_p	A pointer to an integer to contain the difference between `startspace` and the number of entries in each of the arrays `varindices` and `values`. A nonnegative value of `surplus_p` specifies that the length of the arrays was sufficient. A negative value specifies that the length was insufficient and that the routine could not complete its task. In this case, the routine `CPXgetmipstarts` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of `surplus_p` specifies the amount of insufficient space in the arrays.
begin	An integer specifying the beginning of the range of MIP starts to be returned.
end	An integer specifying the end of the range of MIP starts to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` specifies that insufficient space was available in the arrays `varindices` and `values` to hold the MIP start entries.

Global function CPXftran

```
int CPXftran(CPXCENVptr env, CPXCLPptr lp, double * x)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXftran` solves By = x and puts the answer in the vector x, where B is the basis matrix.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
x	An array that holds the righthand side vector on input and the solution vector on output. The array must be of length at least equal to the number of rows in the LP problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsetsolvecallbackfunc

```
int CPXsetsolvecallbackfunc(CPXENVptr env, int(CPXPUBLIC
*solvecallback)(CALLBACK_SOLVE_ARGS), void * cbhandle)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXsetsolvecallbackfunc` sets and modifies the user-written callback to be called during MIP optimization to optimize subproblems (for example, node and heuristic subproblems).

Example

```
status = CPXsetsolvecallbackfunc(env, mysolvefunc, mydata);
```

See also the example `admipex1.c` in the standard distribution.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`solvecallback`

A pointer to a user-written solve callback. If the callback is set to NULL, no callback is called during optimization.

`cbhandle`

A pointer to user private data. This pointer is passed to the callback.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle,
              int        *useraction_p);
```

CPLEX calls the solve callback before CPLEX solves the subproblem currently associated with the current node. The user can choose to solve the subproblem in the solve callback instead by setting the user action argument of the callback. The optimization that the user provides to solve the subproblem must provide a CPLEX solution. That is, the Callable Library routine `CPXgetstat` must return a nonzero value. The user may access the lp pointer of the subproblem with the Callable Library routine `CPXgetcallbacknodelp`.

Callback return value

The callback returns zero if successful and nonzero if an error occurs.

Callback arguments

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`cbdata`

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

`wherefrom`

An integer value reporting where in the optimization this function was called. It will have the value `CPX_CALLBACK_MIP_SOLVE`.

`cbhandle`

A pointer to user private data.

`useraction_p`

A pointer to an integer specifying the action to be taken on completion of the user callback. Table 11 summarizes the possible actions.

Actions to be Taken after a User-Written Solve Callback

Value	Symbolic Constant	Action
0	CPX_CALLBACK_DEFAULT	Use CPLEX subproblem optimizer
1	CPX_CALLBACK_FAIL	Exit optimization
2	CPX_CALLBACK_SET	The subproblem has been solved in the callback

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXuncrushform

`int` **`CPXuncrushform`**`(CPXCENVptr env, CPXCLPptr lp, int plen, const int * pind, const double * pval, int * len_p, double * offset_p, int * ind, double * val)`

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXuncrushform` uncrushes a linear formula of the presolved problem to a linear formula of the original problem.

Let `cols = CPXgetnumcols (env, lp)`. If `ind[i] < cols` then the `i-th` variable in the formula is the variable with index `ind[i]` in the original problem. If `ind[i] >= cols`, then the `i-th` variable in the formula is the slack for the (`ind[i] - cols`)`-th` ranged row. The arrays `ind` and `val` must be of length at least the number of columns plus the number of ranged rows in the original LP problem object.

Example

```
status = CPXuncrushform (env, lp, plen, pind, pval,
                         &len, &offset, ind, val);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
plen	The number of entries in the arrays `pind` and `pval`.
pind	An array containing the column indices of coefficients in the array `pval`.
pval	The linear formula in terms of the presolved problem. Each entry, `pind[i]`, specifies the column index of the corresponding coefficient, `pval[i]`.
len_p	A pointer to an integer to receive the number of nonzero coefficients, that is, the true length of the arrays `ind` and `val`.
offset_p	A pointer to a double to contain the value of the linear formula corresponding to variables that have been removed in the presolved problem.
ind	An array containing indices of coefficients in the array `val`.
val	The linear formula in terms of the original problem.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXinfointparam

```
int CPXinfointparam(CPXCENVptr env, int whichparam, int * defvalue_p, int *
minvalue_p, int * maxvalue_p)
```

Definition file: cplex.h

The routine `CPXinfointparam` obtains the default, minimum, and maximum values of a CPLEX parameter of type `int`.

The *CPLEX Parameters Reference Manual* provides a list of parameters with their types, options, and default values.

Example

```
status = CPXinfointparam (env, CPX_PARAM_PREIND, &default_preind,
                          &min_preind, &max_preind);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
whichparam	The symbolic constant (or reference number) of the parameter for which the value is to be obtained.
defvalue_p	A pointer to an integer variable to hold the default value of the CPLEX parameter. May be NULL.
minvalue_p	A pointer to an integer variable to hold the minimum value of the CPLEX parameter. May be NULL.
maxvalue_p	A pointer to an integer variable to hold the maximum value of the CPLEX parameter. May be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXprimopt

`int` **`CPXprimopt`**`(CPXCENVptr env, CPXLPptr lp)`

Definition file: cplex.h

The routine `CPXprimopt` may be used after a linear program has been created via a call to `CPXcreateprob`, to find a solution to that problem using the primal simplex method. When this function is called, the CPLEX primal simplex algorithm attempts to optimize the specified problem. The results of the optimization are recorded in the CPLEX problem object.

Example

```
status = CPXprimopt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (`CPXERR_NO_MEMORY`) or encountering invalid data in the CPLEX problem object (`CPXERR_NO_PROBLEM`).

Exceeding a user-specified CPLEX limit is not considered an error. Proving the problem infeasible or unbounded is not considered an error.

A zero return value does not necessarily mean that a solution exists. Use the query routines `CPXsolninfo`, `CPXgetstat`, and `CPXsolution` to obtain further information about the status of the optimization.

Global function CPXgetsubmethod

`int` **`CPXgetsubmethod`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetsubmethod` accesses the solution method of the last subproblem optimization, in the case of an error termination during mixed integer optimization.

Example

```
submethod = CPXgetsubmethod (env, lp);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

lp A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Example

```
submethod = CPXgetsubmethod (env, lp);
```

Returns:

The possible return values are summarized here.

Value	Symbolic Constant	Algorithm
0	`CPX_ALG_NONE`	None
1	`CPX_ALG_PRIMAL`	Primal simplex
2	`CPX_ALG_DUAL`	Dual simplex
4	`CPX_ALG_BARRIER`	Barrier optimizer (no crossover)

Global function CPXcheckaddrows

```
int CPXcheckaddrows(CPXCENVptr env, CPXCLPptr lp, int ccnt, int rcnt, int nzcnt,
const double * rhs, const char * sense, const int * rmatbeg, const int * rmatind,
const double * rmatval, char ** colname, char ** rowname)
```

Definition file: cplex.h

The routine `CPXcheckaddrows` validates the arguments of the corresponding `CPXaddrows` routine. This data checking routine is found in source format in the file `check.c` which is provided with the standard CPLEX distribution. To call this routine, you must compile and link `check.c` with your program as well as the CPLEX Callable Library.

The `CPXcheckaddrows` routine has the same argument list as the `CPXaddrows` routine. The second argument, `lp`, is technically a pointer to a constant LP object of type `CPXCLPptr` rather than type `CPXLPptr`, as this routine will not modify the problem. For most user applications, this distinction is unimportant.

Example

```
status = CPXcheckaddrows (env, lp, ccnt, rcnt, nzcnt, rhs,
                          sense, rmatbeg, rmatind, rmatval,
                          newcolname, newrowname);
```

Returns:

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.

Global function CPXcopylpwnames

```
int CPXcopylpwnames(CPXCENVptr env, CPXLPptr lp, int numcols, int numrows, int objsense, const double * objective, const double * rhs, const char * sense, const int * matbeg, const int * matcnt, const int * matind, const double * matval, const double * lb, const double * ub, const double * rngval, char ** colname, char ** rowname)
```

Definition file: cplex.h

The routine `CPXcopylpwnames` copies LP data into a CPLEX problem object in the same way as the routine `CPXcopylp`, but using some additional arguments to specify the names of constraints and variables in the CPLEX problem object. The arguments to `CPXcopylpwnames` define an objective function, constraint matrix, variable bounds, righthand side constraint senses, and range values. Unlike the routine `CPXcopylp`, `CPXcopylpwnames` also copies names. This routine is used in the same way as `CPXcopylp`.

Table 1: Settings for objsense

`objsense`	= `1`	(`CPX_MIN`) minimize
`objsense`	= `-1`	(`CPX_MAX`) maximize

Table 2: Settings for sense

`sense[i]`	= `'L'`	<= constraint
`sense[i]`	= `'E'`	= constraint
`sense[i]`	= `'G'`	>= constraint
`sense[i]`	= `'R'`	ranged constraint

With respect to the arguments `matbeg` (beginning of the matrix), `matcnt` (count of the matrix), `matind` (indices of the matrix), and `matval` (values of the matrix), CPLEX needs to know only the nonzero coefficients. These are grouped by column in the array `matval`. The nonzero elements of every column must be stored in sequential locations in this array with `matbeg[j]` containing the index of the beginning of column `j` and `matcnt[j]` containing the number of entries in column `j`. The components of `matbeg` must be in ascending order. For each k, `matind[k]` specifies the row number of the corresponding coefficient, `matval[k]`.

These arrays are accessed as follows. Suppose that CPLEX wants to access the entries in some column `j`. These are assumed to be given by the array entries:

```
matval[matbeg[j]],.., matval[matbeg[j]+matcnt[j]-1]
```

The corresponding row indices are:

```
matind[matbeg[j]],.., matind[matbeg[j]+matcnt[j]-1]
```

Entries in `matind` are not required to be in row order. Duplicate entries in `matind` and `matval` within a single column are not allowed. The length of the arrays `matbeg` and `matind` should be at least `numcols`. The length of arrays `matind` and `matval` should be at least `matbeg[numcols-1]+matcnt[numcols-1]`.

When you build or modify your problem with this routine, you can verify that the results are as you intended by calling `CPXcheckcopylpwnames` during application development.

Example

```
status = CPXcopylpwnames (env,
                          lp,
                          numcols,
```

```
                    numrows,
                    objsen,
                    obj,
                    rhs,
                    sense,
                    matbeg,
                    matcnt,
                    matind,
                    matval,
                    lb,
                    ub,
                    rngval,
                    colname,
                    rowname);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
numcols	An integer that specifies the number of columns in the constraint matrix, or equivalently, the number of variables in the problem object.
numrows	An integer that specifies the number of rows in the constraint matrix, not including the objective function or bounds on the variables.
objsense	An integer that specifies whether the problem is a minimization or maximization problem. Table 1 shows its possible settings.
objective	An array of length at least `numcols` containing the objective function coefficients.
rhs	An array of length at least `numrows` containing the righthand side value for each constraint in the constraint matrix.
sense	An array of length at least `numrows` containing the sense of each constraint in the constraint matrix. Table 2 shows the possible settings.
matbeg	An array that defines the constraint matrix.
matcnt	An array that defines the constraint matrix.
matind	An array that defines the constraint matrix.
matval	An array that defines the constraint matrix.
lb	An array of length at least `numcols` containing the lower bound on each of the variables. Any lower bound that is set to a value less than or equal to that of the constant `-CPX_INFBOUND` is treated as negative infinity. `CPX_INFBOUND` is defined in the header file `cplex.h`.
ub	An array of length at least `numcols` containing the upper bound on each of the variables. Any upper bound that is set to a value greater than or equal to that of the constant `CPX_INFBOUND` is treated as infinity. `CPX_INFBOUND` is defined in the header file `cplex.h`.
rngval	An array of length at least `numrows` containing the range value of each ranged constraint. Ranged rows are those designated by `R` in the `sense` array. If the row is not ranged, the `rngval` array entry is ignored. If `rngval[i]` > 0, then row `i` activity is in `[rhs[i],rhs[i]+rngval[i]]`, and if `rngval[i]` <= 0, then row `i` activity is in `[rhs[i]+rngval[i],rhs[i]]`. This argument may be NULL.
colname	An array of length at least `numcols` containing pointers to character strings. Each string is terminated with the NULL character. These strings represent the names of the matrix columns or, equivalently, the variable names. May be NULL if no names are associated with the variables. If `colname` is not NULL, every variable must be given a name. The addresses in `colname` do not have to be in ascending order.
rowname	An array of length at least `numrows` containing pointers to character strings. Each string is terminated with the NULL character. These strings represent the names of the matrix rows or, equivalently, the constraint names. May be NULL if no names are associated with the constraints. If `rowname` is not NULL, every constraint must be given a name. The addresses in `rowname` do not have to be in ascending order.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXchgcoeflist

```
int CPXchgcoeflist(CPXCENVptr env, CPXLPptr lp, int numcoefs, const int * rowlist,
const int * collist, const double * vallist)
```

Definition file: cplex.h

The routine `CPXchgcoeflist` changes a list of matrix coefficients of a CPLEX problem object. The list is prepared as a set of triples (`i, j, value`), where `i` is the row index, `j` is the column index, and `value` is the new value. The list may be in any order.

Note

The corresponding rows and columns must already exist in the CPLEX problem object.

This routine cannot be used to change objective, righthand side, range, or bound coefficients.

Duplicate entries, that is, two triplets with identical `i` and `j`, are not allowed.

When you build or modify your problem with this routine, you can verify that the results are as you intended by calling `CPXcheckchgcoeflist` during application development.

Example

```
status = CPXchgcoeflist (env, lp, numcoefs, rowlist, collist, vallist);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
numcoefs	The number of coefficients to change, or, equivalently, the length of the arrays `rowlist`, `collist`, and `vallist`.
rowlist	An array of length `numcoefs` that with `collist` and `vallist` specifies the coefficients to change.
collist	An array of length `numcoefs` that with `rowlist` and `vallist` specifies the coefficients to change.
vallist	An array of length `numcoefs` that with `rowlist` and `collist` specifies the coefficients to change. The entries `rowlist[k]`, `collist[k]`, and `vallist[k]` specify that the matrix coefficient in row `rowlist[k]` and column `collist[k]` should be changed to the value `vallist[k]`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXbtran

```
int CPXbtran(CPXCENVptr env, CPXCLPptr lp, double * y)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXbtran` solves xTB = yT and puts the answer in y. B is the basis matrix.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX LP problem object, as returned by `CPXcreateprob`.
y	An array that holds the righthand side vector on input and the solution vector on output. The array must be of length at least equal to the number of rows in the LP problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetcallbackinfo

```
int CPXgetcallbackinfo(CPXCENVptr env, void * cbdata, int wherefrom, int whichinfo,
void * result_p)
```

Definition file: cplex.h

The routine `CPXgetcallbackinfo` accesses information about the current optimization process from within a user-written callback function.

Note

This routine is the only routine that can access optimization status information from within a nonadvanced user-written callback function. It is also the only Callable Library routine that may be called from within a nonadvanced user-written callback function, and in fact, may only be called from the callback function.

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`cbdata`

The `cbdata` pointer passed to the user-written callback function. The argument `cbdata` MUST be the value of `cbdata` passed to the user-written callback function.

`wherefrom`

An integer value specifying the optimization algorithm from which the user-written callback function was called. The argument `wherefrom` MUST be the value of `wherefrom` passed to the user-written callback function. See `CPXgetlpcallbackfunc`, `CPXgetmipcallbackfunc`, and `CPXgetnetcallbackfunc` for possible values of `wherefrom` and their meaning.

`whichinfo`

An integer value specifying the specific information that should be returned by `CPXgetcallbackinfo` to the result argument. Values for `whichinfo`, the type of the information returned into `*result_p`, plus a description appear in the table.

`result_p`

A generic pointer to a variable of type `double` or `int`, dependent on the value of `whichinfo`, as documented in the following tables.

For LP algorithms:

whichinfo	type of *result_p	description
`CPX_CALLBACK_INFO_ENDTIME`	`double`	time stamp
`CPX_CALLBACK_INFO_PRIMAL_OBJ`	`double`	primal objective value
`CPX_CALLBACK_INFO_DUAL_OBJ`	`double`	dual objective value
`CPX_CALLBACK_INFO_PRIMAL_INFMEAS`	`double`	measure of primal infeasibility
`CPX_CALLBACK_INFO_DUAL_INFMEAS`	`double`	measure of dual infeasibility
`CPX_CALLBACK_INFO_PRIMAL_FEAS`	`int`	1 if primal feasible, 0 if not

CPX_CALLBACK_INFO_DUAL_FEAS	int	1 if dual feasible, 0 if not
CPX_CALLBACK_INFO_ITCOUNT	int	iteration count
CPX_CALLBACK_INFO_CROSSOVER_PPUSH	int	primal push crossover itn. count
CPX_CALLBACK_INFO_CROSSOVER_PEXCH	int	primal exchange crossover itn. count
CPX_CALLBACK_INFO_CROSSOVER_DPUSH	int	dual push crossover itn. count
CPX_CALLBACK_INFO_CROSSOVER_DEXCH	int	dual exchange crossover itn. count
CPX_CALLBACK_INFO_CROSSOVER_SBCNT	int	number of super-basic variables left in the basis
CPX_CALLBACK_INFO_USER_PROBLEM	CPXCLPptr	returns pointer to original user problem; available for primal, dual, barrier, mip

Crossover from a barrier to a simplex solution reduces the super-basic count. In fact, in a barrier crossover, if the count reaches 0 (zero), the crossover has finished. Thus, the super-basic count serves as a measure of progress for the crossover algorithm.

For Network algorithms:

whichinfo	type of *result_p	description
CPX_CALLBACK_INFO_ENDTIME	double	time stamp
CPX_CALLBACK_INFO_PRIMAL_OBJ	double	primal objective value
CPX_CALLBACK_INFO_PRIMAL_INFMEAS	double	measure of primal infeasibility
CPX_CALLBACK_INFO_ITCOUNT	int	iteration count
CPX_CALLBACK_INFO_PRIMAL_FEAS	int	1 if primal feasible, 0 if not

For Presolve algorithms:

whichinfo	type of *result_p	description
CPX_CALLBACK_INFO_ENDTIME	double	time stamp
CPX_CALLBACK_INFO_PRESOLVE_ROWSGONE	int	number of rows eliminated
CPX_CALLBACK_INFO_PRESOLVE_COLSGONE	int	number of columns eliminated
CPX_CALLBACK_INFO_PRESOLVE_AGGSUBST	int	number of aggregator substitutions
CPX_CALLBACK_INFO_PRESOLVE_COEFFS	int	number of modified coefficients

For MIP algorithms and informational callbacks:

whichinfo	type of *result_p	description
CPX_CALLBACK_INFO_ENDTIME	double	time stamp
CPX_CALLBACK_INFO_BEST_INTEGER	double	obj. value of best integer solution
CPX_CALLBACK_INFO_BEST_REMAINING	double	obj. value of best remaining node
CPX_CALLBACK_INFO_NODE_COUNT	int	total number of nodes solved

CPX_CALLBACK_INFO_NODES_LEFT	int	number of remaining nodes
CPX_CALLBACK_INFO_MIP_ITERATIONS	int	total number of MIP iterations
CPX_CALLBACK_INFO_MIP_FEAS	int	returns 1 if feasible solution exists; otherwise, 0
CPX_CALLBACK_INFO_CUTOFF	double	updated cutoff value
CPX_CALLBACK_INFO_PROBE_PHASE	int	current phase of probing (0-3)
CPX_CALLBACK_INFO_PROBE_PROGRESS	double	fraction of probing phase completed (0.0-1.0)
CPX_CALLBACK_INFO_FRACCUT_PROGRESS	double	fraction of Gomory cut generation for the pass completed (0.0 - 1.0)
CPX_CALLBACK_INFO_DISJCUT_PROGRESS	double	fraction of disjunctive cut generation for the pass completed (0.0 - 1.0)
CPX_CALLBACK_INFO_FLOWMIR_PROGRESS	double	fraction of flow cover and MIR cut generation for the pass completed (0.0 - 1.0)
CPX_CALLBACK_INFO_MIP_REL_GAP	double	relative MIP gap between the current best primal and dual bounds

You can query the relative MIP gap between the current best primal and dual bounds **outside** a callback with the routine CPXgetmiprelgap.

For MIP algorithms and advanced callbacks:

whichinfo	type of *result_p	description
CPX_CALLBACK_INFO_ENDTIME	double	time stamp
CPX_CALLBACK_INFO_BEST_INTEGER	double	obj. value of best integer solution
CPX_CALLBACK_INFO_BEST_REMAINING	double	obj. value of best remaining node
CPX_CALLBACK_INFO_NODE_COUNT	int	total number of nodes solved
CPX_CALLBACK_INFO_NODES_LEFT	int	number of remaining nodes
CPX_CALLBACK_INFO_MIP_ITERATIONS	int	total number of MIP iterations
CPX_CALLBACK_INFO_MIP_FEAS	int	returns 1 if feasible solution exists; otherwise, 0
CPX_CALLBACK_INFO_CUTOFF	double	updated cutoff value
CPX_CALLBACK_INFO_CLIQUE_COUNT	int	number of clique cuts added
CPX_CALLBACK_INFO_COVER_COUNT	int	number of cover cuts added
CPX_CALLBACK_INFO_DISJCUT_COUNT	int	number of disjunctive cuts added
CPX_CALLBACK_INFO_FLOWCOVER_COUNT	int	number of flow cover cuts added
CPX_CALLBACK_INFO_FLOWPATH_COUNT	int	number of flow path cuts added
CPX_CALLBACK_INFO_FRACCUT_COUNT	int	number of Gomory fractional cuts added
CPX_CALLBACK_INFO_GUBCOVER_COUNT	int	number of GUB cover cuts added
CPX_CALLBACK_INFO_IMPLBD_COUNT	int	number of implied bound cuts added

CPX_CALLBACK_INFO_MCFCUT_COUNT	int	number of multi-commodity flow cuts added
CPX_CALLBACK_INFO_MIRCUT_COUNT	int	number of mixed integer rounding cuts added
CPX_CALLBACK_INFO_ZEROHALFCUT_COUNT	int	number of zero-half cuts added
CPX_CALLBACK_INFO_USER_PROBLEM	CPXCLPptr	returns pointer to original user problem; available for primal, dual, barrier, MIP
CPX_CALLBACK_INFO_PROBE_PHASE	int	current phase of probing (0-3)
CPX_CALLBACK_INFO_PROBE_PROGRESS	double	fraction of probing phase completed (0.0-1.0)
CPX_CALLBACK_INFO_FRACCUT_PROGRESS	double	fraction of Gomory cut generation for the pass completed (0.0 - 1.0)
CPX_CALLBACK_INFO_DISJCUT_PROGRESS	double	fraction of disjunctive cut generation for the pass completed (0.0 - 1.0)
CPX_CALLBACK_INFO_FLOWMIR_PROGRESS	double	fraction of flow cover and MIR cut generation for the pass completed (0.0 - 1.0)
CPX_CALLBACK_INFO_MY_THREAD_NUM	int	identifier of the parallel thread making this call
CPX_CALLBACK_INFO_USER_THREADS	int	total number of parallel threads currently running

For Tuning:

whichinfo	type of *result_p	description
CPX_CALLBACK_INFO_ENDTIME	double	time stamp
CPX_CALLBACK_INFO_TUNING_PROGRESS	double	elapsed percentage of total tuning time

Example

See `lpex4.c` in the *CPLEX User's Manual*.

Suppose you want to know the objective value on each iteration for a graphical user display. In addition, if primal simplex is not feasible after 1000 iterations, you want to stop the optimization. The function `mycallback` is a callback function to do this.

```
int mycallback (CPXCENVptr env, void *cbdata, int wherefrom,
                void *cbhandle)
{
 int itcount;
 double objval;
 int ispfeas;
 int status = 0;

 if ( wherefrom == CPX_CALLBACK_PRIMAL ) {
    status = CPXgetcallbackinfo (env, cbdata, wherefrom,
                                 CPX_CALLBACK_INFO_PRIMAL_FEAS,
                                 &ispfeas);
    if ( status ) {
       fprintf (stderr,"error %d in CPXgetcallbackinfon", status);
       status = 1;
       goto TERMINATE;
    }
    if ( ispfeas ) {
       status = CPXgetcallbackinfo (env, cbdata, wherefrom,
                CPX_CALLBACK_INFO_PRIMAL_OBJ,
                &objval) )
      if ( status ) {
         fprintf (stderr,"error %d in CPXgetcallbackinfon",
```

```
                status);
         status = 1;
         goto TERMINATE;
       }

     }
     else {
          status = CPXgetcallbackinfo (env, cbdata, wherefrom,
                                       CPX_CALLBACK_INFO_ITCOUNT,
                                       &itcount);
          if ( status ) {
             fprintf (stderr,"error %d in CPXgetcallbackinfon", status);
             status = 1;
             goto TERMINATE;
          }
          if ( itcount > 1000 )  status = 1;
     }
  }

TERMINATE:
   return (status);
}
```

Returns:

The routine returns zero if successful and nonzero if an error occurs. If nonzero, the requested value may not be available for the specific optimization algorithm. For example, the dual objective is not available from primal simplex.

Global function CPXgetcallbacknodeintfeas

```
int CPXgetcallbacknodeintfeas(CPXCENVptr env, void * cbdata, int wherefrom, int *
feas, int begin, int end)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbacknodeintfeas` retrieves information for each variable about whether or not the variable is integer feasible in the node subproblem. It can be used in a user-written callback during MIP optimization. The information is from the original problem if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`. Otherwise, they are from the presolved problem.

Example

```
status = CPXgetcallbacknodeintfeas(env, cbdata, wherefrom,
                                   feas, 0, cols-1);
```

See `admipex1.c` and `admipex2.c` in the standard distribution.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`, or
- `CPX_CALLBACK_MIP_CUT`.

Integer feasibility status information for a node of the subproblem

`CPX_INTEGER_FEASIBLE`	0	variable j+begin is integer-valued
`CPX_INTEGER_INFEASIBLE`	1	variable j+begin is not integer-valued
`CPX_IMPLIED_INTEGER_FEASIBLE`	2	variable j+begin may have a fractional value in the current solution, but it will take on an integer value when all integer variables still in the problem have integer values. It should not be branched upon.

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting from where the user-written callback was called. The argument must be the value of `wherefrom` passed to the user-written callback.
feas	An array to receive integer feasibility information for the node subproblem. This array must be of length at least (`end - begin + 1`). If successful, `feas[0]` through `feas[end-begin]` will contain the integer feasibility information. Possible return values appear in the table.
begin	An integer specifying the beginning of the range of integer feasibility information to be returned.
end	An integer specifying the end of the range of integer feasibility information to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETchgarcname

`int` **`CPXNETchgarcname`**`(CPXCENVptr env, CPXNETptr net, int cnt, const int * indices, char ** newname)`

Definition file: cplex.h

This routine `CPXNETchgarcname` changes the names of a set of arcs in the network stored in a network problem object.

Example

```
status = CPXNETchgarcname (env, net, 10, indices, newname);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
cnt	An integer that indicates the total number of arc names to be changed. Thus `cnt` specifies the length of the arrays `indices` and `newname`.
indices	An array of length `cnt` containing the numeric indices of the arcs for which the names are to be changed.
newname	An array of length `cnt` containing the new names for the arcs specified in `indices`.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetsolnpoolrngfilter

```
int CPXgetsolnpoolrngfilter(CPXCENVptr env, CPXCLPptr lp, double * lb_p, double *
ub_p, int * nzcnt_p, int * ind, double * val, int space, int * surplus_p, int
which)
```

Definition file: cplex.h

Accesses a range filter of the solution pool.

This routine accesses a range filter, specified by the argument `which`, of the solution pool associated with the LP problem specified by the argument `lp`. Details about that filter are returned in the arguments of this routine.

Example

```
status = CPXgetsolnpoolrngfilter (env, lp,
                                  &limlo, &limup,
                                  &num, ind, val,
                                  cols, &surplus, i);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lb_p	Lower bound on the linear expression of a range filter.
ub_p	Upper bound on the linear expression of a range filter.
nzcnt_p	Number of variables in the linear expression of a range filter.
ind	An array of indices of variables in the linear expression of a range filter. May be NULL if `space>` is 0.
val	An array of coefficients in the linear expression of a range filter. May be NULL if `space>` is 0.
space	Integer specifying the length of the arrays `ind` and `val`.
surplus_p	A pointer to an integer to contain the difference between `space` and the number of entries in each of the arrays `ind` and `val`. A nonnegative value of `surplus_p` means that the length of the arrays was sufficient. A negative value reports that the length was insufficient and consequently the routine could not complete its task. In this case, the routine `CPXgetsolnpoolrngfilter` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of `surplus_p` specifies the amount of insufficient space in the arrays.
which	The filter.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETgetub

`int` **`CPXNETgetub`**`(CPXCENVptr env, CPXCNETptr net, double * up, int begin, int end)`

Definition file: cplex.h

The routine `CPXNETgetub` is used to access the upper capacity bounds for a range of arcs in the network stored in a network problem object.

Example

```
status = CPXNETgetub (env, net, up, 0, cur_narcs-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
up	Array in which to write the upper bound on the flow for the requested arcs. If NULL is passed, no upper bounds are retrieved. Otherwise, the array must be of size (`end-begin+1`).
begin	Index of the first arc for which upper bounds are to be obtained.
end	Index of the last arc for which upper bounds are to be obtained.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetax

```
int CPXgetax(CPXCENVptr env, CPXCLPptr lp, double * x, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetax` accesses row activity levels for a range of linear constraints. The beginning and end of the range must be specified. A row activity is the inner product of a row in the constraint matrix and the structural variables in the problem.

Example

```
status = CPXgetax (env, lp, x, 0, CPXgetnumrows(env,lp)-1);
```

The array must be of length at least (`end-begin+1`). If successful, `x[0]` through `x[end-begin]` contain the row activities.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
x	An array to receive the values of the row activity levels for each of the constraints in the specified range. The array must be of length at least (`end-begin+1`). If successful, `x[0]` through `x[end-begin]` contain the row activities.
begin	An integer specifying the beginning of the range of row activities to be returned.
end	An integer specifying the end of the range of row activities to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXqpuncrushpi

```
int CPXqpuncrushpi(CPXCENVptr env, CPXCLPptr lp, double * pi, const double * prepi,
const double * x)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXqpuncrushpi` uncrushes a dual solution for the presolved problem to a dual solution for the original problem if the original problem is a QP.

Example

```
status = CPXqpuncrushpi (env, lp, pi, prepi, x);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
pi	An array to receive dual solution (`pi`) values for the original problem as computed from the dual values of the presolved problem object. The length of the array must at least equal the number of rows in the LP problem object.
prepi	An array that contains dual solution (`pi`) values for the presolved problem, as returned by such routines as `CPXgetpi` and `CPXsolution` when applied to the presolved problem object. The length of the array must at least equal the number of rows in the presolved problem object.
x	An array that contains primal solution (`x`) values for a problem, as returned by such routines as `CPXuncrushx` and `CPXcrushx`. The length of the array must at least equal the number of columns in the LP problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetinfocallbackfunc

```
int CPXgetinfocallbackfunc(CPXCENVptr env, int(CPXPUBLIC **callback_p)(CPXCENVptr,
void *, int, void *), void ** cbhandle_p)
```

Definition file: cplex.h

The routine `CPXgetinfocallbackfunc` accesses the user-written callback routine to be called regularly during the optimization of a mixed integer program (MIP).

This routine enables the user to access a separate callback function to be called during the solution of mixed integer programming problems (MIPs). Unlike any other callback routines, this user-written callback routine is used only to retrieve information about MIP search. It does not control the search, though it allows the search to terminate. The user-written callback function that this routine invokes is allowed to call only two other routines: `CPXgetcallbackinfo` and `CPXgetcallbackincumbent`.

The prototype for the user-written callback function is identical to that of `CPXsetmipcallbackfunc`.

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`callback_p`

The address of the pointer to the current user-written callback function. If no callback function has been set, the pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the user's private pointer.

Example

```
status = CPXgetinfocallbackfunc (env, mycallback, NULL);
```

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle);
```

This is the user-written callback routine.

Callback return value

A nonzero return value terminates the optimization. That is, if your user-written callback function returns a nonzero value, it signals CPLEX that the optimization should terminate.

Callback arguments

`env`

A pointer to the CPLEX environment that was passed into the associated optimization routine.

`cbdata`

A pointer passed from the optimization routine to the user-written callback function that identifies the problem being optimized. The only purpose for the `cbdata` pointer is to pass it to the routine `CPXgetcallbackinfo`.

`wherefrom`

An integer value reporting from which optimization algorithm the user-written callback function was called. Possible values and their meaning appear in this table.

Indicators of algorithm that called user-written callback

Value	Symbolic Constant	Meaning
101	CPX_CALLBACK_MIP	From mipopt
107	CPX_CALLBACK_MIP_PROBE	From probing or clique merging
108	CPX_CALLBACK_MIP_FRACCUT	From Gomory fractional cuts
109	CPX_CALLBACK_MIP_DISJCUT	From disjunctive cuts
110	CPX_CALLBACK_MIP_FLOWMIR	From Mixed Integer Rounding cuts

`cbhandle`

Pointer to user private data, as passed to `CPXsetinfocallbackfunc`.

See Also: CPXgetcallbackinfo

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcopypnorms

```
int CPXcopypnorms(CPXCENVptr env, CPXLPptr lp, const double * cnorm, const double *
rnorm, int len)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXcopypnorms` copies the primal steepest-edge norms to the specified LP problem object.

See Also: CPXcopydnorms, CPXgetpnorms

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
cnorm	An array containing values to be used in a subsequent call to `CPXprimopt`, with a setting of `CPX_PARAM_PPRIIND` equal to 2, as the initial values for the primal steepest-edge norms of the first `len` columns in the LP problem object. The array must be of length at least equal to the value of the argument `len`.
rnorm	An array containing values to be used in a subsequent call to `CPXprimopt` with a setting of `CPX_PARAM_PPRIIND` equal to 2, as the initial values for the primal steepest-edge norms of the slacks and ranged variables that are nonbasic. The array must be of length at least equal to the number of rows in the LP problem object.
len	An integer that specifies the number of entries in the array `cnorm[]`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXbranchcallbackbranchbds

```
int CPXbranchcallbackbranchbds(CPXCENVptr env, void * cbdata, int wherefrom, double
nodeest, int cnt, const int * indices, const char * lu, const int * bd, void *
userhandle, int * seqnum_p)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXbranchcallbackbranchbds` specifies the branches to be taken from the current node. It may be called only from within a user-written branch callback function.

Branch variables are in terms of the original problem if the parameter `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, branch variables are in terms of the presolved problem.

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	A pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value that reports where the user-written callback was called from. This argument must be the value of `wherefrom` passed to the user-written callback.
nodeest	A double that specifies the value of the node estimate for the node to be created with this branch. The node estimate is used to select nodes from the branch-and-cut tree with certain values of the node selection parameter `CPX_PARAM_NODESEL`.
cnt	An integer. The integer specifies the number of bound changes that are specified in the arrays `indices`, `lu`, and `bd`.
indices	An array. Together with `lu` and `bd`, this array defines the bound changes for the branch. The entry `indices[i]` is the index for the variable.
lu	An array. Together with `indices` and `bd`, this array defines the bound changes for each of the created nodes. The entry `lu[i]` is one of the three possible values specifying which bound to change: `L` for lower bound, `U` for upper bound, or `B` for both bounds.
bd	An array. Together with `indices` and `lu`, this array defines the bound changes for each of the created nodes. The entry `bd[i]` specifies the new value of the bound.
userhandle	A pointer to user private data that should be associated with the node created by this branch. May be NULL.
seqnum_p	A pointer to an integer. On return, that integer will contain the sequence number that CPLEX has assigned to the node created from this branch. The sequence number may be used to select this node in later calls to the node callback.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetdblparam

```
int CPXgetdblparam(CPXCENVptr env, int whichparam, double * value_p)
```

Definition file: cplex.h

The routine `CPXgetdblparam` obtains the current value of a CPLEX parameter of type `double`.

The *CPLEX Parameters Reference Manual* provides a list of parameters with their types, options, and default values.

Example

```
status = CPXgetdblparam (env, CPX_PARAM_TILIM, &curtilim);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
whichparam	The symbolic constant (or reference number) of the parameter for which the value is to be obtained.
value_p	A pointer to a variable of type `double` to hold the current value of the CPLEX parameter.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXlpopt

`int` **`CPXlpopt`**`(CPXCENVptr env, CPXLPptr lp)`

Definition file: cplex.h

The routine `CPXlpopt` may be used, at any time after a linear program has been created via a call to `CPXcreateprob`, to find a solution to that problem using one of the CPLEX linear optimizers. The parameter `CPX_PARAM_LPMETHOD` controls the choice of optimizer (dual simplex, primal simplex, barrier, network simplex, sifting, or concurrent optimization). Currently, with the default parameter setting of Automatic, CPLEX invokes the dual simplex method when no advanced basis or starting vector is loaded or when the advanced indicator is zero. The behavior of the Automatic setting may change in the future.

Example

```
status = CPXlpopt (env, lp);
```

See also the example `lpex1.c` in *Getting Started* and in the standard distribution.

See Also: CPXgetstat, CPXsolninfo, CPXsolution

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns zero unless an error occurred during the optimization.

Examples of errors include exhausting available memory (`CPXERR_NO_MEMORY`) or encountering invalid data in the CPLEX problem object (`CPXERR_NO_PROBLEM`).

Exceeding a user-specified CPLEX limit is not considered an error. Proving the problem infeasible or unbounded is not considered an error.

A zero return value does not necessarily mean that a solution exists. Use the query routines `CPXsolninfo`, `CPXgetstat`, and `CPXsolution` to obtain further information about the status of the optimization.

Global function CPXNETgetpi

`int` **`CPXNETgetpi`**`(CPXCENVptr env, CPXCNETptr net, double * pi, int begin, int end)`

Definition file: cplex.h

The routine `CPXNETgetpi` is used to access dual values for a range of nodes in the network stored in a network problem object.

For this function to succeed, a solution must exist for the problem object.

Example

```
status = CPXNETgetpi (env, net, pi, 10, 20);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
pi	Array in which to write solution dual values for requested nodes. If NULL is passed, no data is returned. Otherwise, `pi` must point to an array of size at least (`end-begin+1`).
begin	Index of the first node for which the dual value is to be obtained.
end	Index of the last node for which the dual value is to be obtained.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetconflict

```
int CPXgetconflict(CPXCENVptr env, CPXCLPptr lp, int * confstat_p, int * rowind,
int * rowbdstat, int * confnumrows_p, int * colind, int * colbdstat, int *
confnumcols_p)
```

Definition file: cplex.h

This routine returns the linear constraints and variables belonging to a conflict previously computed by the routine `CPXrefineconflict`. The conflict is a subset of constraints and variables from the original, infeasible problem that is still infeasible. It is generally minimal, in the sense that removal of any of the constraints or variable bounds in the conflict will make the conflict set become feasible. However, the computed conflict will not be minimal if the previous call to `CPXrefineconflict` was not allowed to run to completion.

Conflict Status

The status of the currently available conflict is returned in `confstat_p`. If `CPXrefineconflict` was called previously, the status will be one of the following values:

- `CPX_STAT_CONFLICT_MINIMAL`,
- `CPX_STAT_CONFLICT_FEASIBLE`, or
- `CPX_STAT_CONFLICT_ABORT_reason`.

When the status of a conflict is `CPX_STAT_CONFLICT_FEASIBLE`, the routine `CPXrefineconflict` determined that the problem was feasible, and thus no conflict is available. Otherwise, a conflict is returned. The returned conflict is minimal if the status is `CPX_STAT_CONFLICT_MINIMAL`.

The conflict status can also be queried with the routine `CPXgetstat`.

Row and Column Status

In the array `rowbdstat`, integer values are returned specifying the status of the corresponding row in the conflict. For row `rowind[i]`, `rowbdstat[i]` can assume the value `CPX_CONFLICT_MEMBER` for constraints that participate in a minimal conflict. When the computed conflict is not minimal, `rowbdstat[i]` can assume the value `CPX_CONFLICT_POSSIBLE_MEMBER`, to report that row i has not been proven to be part of the conflict. If a row has been proven not to belong to the conflict, its index will not be listed in `rowind`.

Similarly, the array `colbdstat` contains integers specifying the status of the variable bounds in the conflict. The value specified in `colbdstat[i]` is the conflict status for variable `colind[i]`. If `colind[i]` has been proven to be part of the conflict, `colbdstat[i]` will take one of the following values:

- `CPX_CONFLICT_MEMBER`,
- `CPX_CONFLICT_LB`, or
- `CPX_CONFLICT_UB`.

When variable `colind[i]` has neither been proven to belong nor been proven not to belong to the conflict, the status `colbdstat[i]` will be one of the following values:

- `CPX_CONFLICT_POSSIBLE_MEMBER`,
- `CPX_CONFLICT_POSSIBLE_LB`, or
- `CPX_CONFLICT_POSSIBLE_UB`.

In both cases, the _LB status specifies that only the lower bound is part of the conflict. Similarly, the _UB status specifies that the upper bound is part of the conflict. Finally, if both bounds are required in the conflict, a _MEMBER status is assigned to that variable.

The status values marked `POSSIBLE` specify that the corresponding constraints and variables in the conflict are possibly not required to produce a minimal conflict, but the conflict refinement algorithm was not able to remove them before it terminated (for example, because it reached a time limit set by the user).

Example

```
status = CPXgetconflictext (env, lp, grpstat, 0, ngrp-1);
```

See Also: CPXrefineconflict, CPXclpwrite

Parameters:

env	A pointer to the CPLEX environment as returned by the routine `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
confstat_p	A pointer to an integer used to return the status of the conflict.
rowind	An array to receive the list of the indices of the constraints that participate in the conflict. The length of the array must not be less than the number of rows in the conflict. If that number is not known, use the total number of rows in the problem object instead.
rowbdstat	An array to receive the conflict status of the rows. Entry `rowbdstat[i]` gives the status of row `rowind[i]`. The length of the array must not be less than the number of rows in the conflict. If that number is not known, use the number of rows in the problem object instead.
confnumrows_p	A pointer to an integer where the number of rows in the conflict is returned.
colind	An array to receive the list of the indices of the variables that participate in the conflict. The length of the array must not be less than the number of columns in the conflict. If that number is not known, use the number of columns in the problem object instead.
colbdstat	An array to receive the conflict status of the columns. Entry `colbdstat[i]` gives the status of column `colind[i]`. The length of the array must not be less than the number of columns in the conflict. If that number is not known, use the number of columns in the problem object instead.
confnumcols_p	A pointer to an integer where the number of columns in the conflict is returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetmethod

`int` **`CPXgetmethod`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetmethod` returns an integer specifying the solution algorithm used to solve the resident LP, QP, or QCP problem.

The possible return values are summarized in the table.

Value	Symbolic Constant	Algorithm
0	`CPX_ALG_NONE`	None
1	`CPX_ALG_PRIMAL`	Primal simplex
2	`CPX_ALG_DUAL`	Dual simplex
4	`CPX_ALG_BARRIER`	Barrier optimizer (no crossover)
4	`CPX_ALG_FEASOPT`	Feasopt
4	`CPX_ALG_MIP`	Mixed integer optimizer

Example

```
method = CPXgetmethod (env, lp);
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns one of the possible values summarized in the trable.

Global function CPXgetbase

`int` **`CPXgetbase`**`(CPXCENVptr env, CPXCLPptr lp, int * cstat, int * rstat)`

Definition file: cplex.h

The routine `CPXgetbase` accesses the basis resident in a CPLEX problem object. Either of the arguments `cstat` or `rstat` may be NULL if only one set of status values is needed.

Table 1: Values of elements of cstat

`CPX_AT_LOWER`	`0`	variable at lower bound
`CPX_BASIC`	`1`	variable is basic
`CPX_AT_UPPER`	`2`	variable at upper bound
`CPX_FREE_SUPER`	`3`	variable free and nonbasic

Table 2: Values of elements of rstat in rows other than ranged rows

`CPX_AT_LOWER`	`0`	associated slack, surplus, or artificial variable is nonbasic at value 0.0 (zero)
`CPX_BASIC`	`1`	associated slack, surplus, or artificial variable is basic

Table 3: Values of elements of rstat for ranged rows

`CPX_AT_LOWER`	`0`	associated slack, surplus, or artificial variable is nonbasic at its lower bound
`CPX_BASIC`	`1`	associated slack, surplus, or artificial variable is basic
`CPX_AT_UPPER`	`2`	associated slack, surplus, or artificial variable is nonbasic at upper bound

Example

```
status = CPXgetbase (env, lp, cstat, rstat);
```

See also the example `lpex2.c` in the examples distributed with the product.

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

lp A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

cstat An array to receive the basis status of the columns in the CPLEX problem object. The length of the array must be no less than the number of columns in the matrix. The array element `cstat[i]` has the meaning specified in Table 1.

rstat An array to receive the basis status of the artificial, slack, or surplus variable associated with each row in the constraint matrix. The length of the array must be no less than the number of rows in the CPLEX problem object. For rows other than ranged rows, the array element `rstat[i]` has the meaning specified in Table 2. For ranged rows, the array element `rstat[i]` has the meaning specified in Table 3.

Returns:

The routine returns zero if a basis exists. It returns nonzero if no solution exists or any other type of error occurs.

Global function CPXmsgstr

```
int CPXmsgstr(CPXCHANNELptr channel, const char * msg_str)
```

Definition file: cplex.h

The routine `CPXmsgstr` sends a character string to a CPLEX message channel. It is provided as an alternative to `CPXmsg`, which due to its variable-length argument list, cannot be used in some environments, such as Visual Basic.

Example

```
CPXmsgstr (p, q);
```

Parameters:

channel	The pointer to the channel receiving the message.
msg_str	A pointer to a string that should be sent to the message channel.

Returns:

The routine returns the number of characters in the string msg.

Global function CPXgetdeletenodecallbackfunc

```
void CPXgetdeletenodecallbackfunc(CPXCENVptr env, void(CPXPUBLIC
**deletecallback_p)(CALLBACK_DELETENODE_ARGS), void ** cbhandle_p)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetdeletenodecallbackfunc` accesses the user-written callback to be called during MIP optimization when a node is to be deleted. Nodes are deleted when a branch is carried out from that node, when the node relaxation is infeasible, or when the node relaxation objective value is worse than the cutoff. This callback can be used to delete user data associated with a node.

Example

```
CPXgetdeletenodecallbackfunc(env,
                             &current_callback,
                             &current_cbdata);
```

See also *Advanced MIP Control Interface* in the *CPLEX User's Manual*.

For documentation of callback arguments, see the routine `CPXsetdeletenodecallbackfunc`.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`deletenodecallback_p`

The address of the pointer to the current user-written delete-node callback. If no callback has been set, the pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the user's private pointer.

See Also: CPXsetdeletenodecallbackfunc, CPXbranchcallbackbranchbds, CPXbranchcallbackbranchconstraints, CPXbranchcallbackbranchgeneral

Returns:

This routine does not return a result.

Global function CPXgetcallbackglobalub

```
int CPXgetcallbackglobalub(CPXCENVptr env, void * cbdata, int wherefrom, double *
ub, int begin, int end)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbackglobalub` retrieves the best known global upper bound values during MIP optimization from within a user-written callback. The global upper bounds are tightened after a new incumbent is found, so the values returned by `CPXgetcallbacknodex` may violate these bounds at nodes where new incumbents have been found. The values are from the original problem if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`; otherwise, they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

Example

```
status = CPXgetcallbackglobalub (env, cbdata, wherefrom,
                                 gub, 0, cols-1);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting from where the user-written callback was called. The argument must be the value of `wherefrom` passed to the user-written callback.
ub	An array to receive the values of the global upper bound values. This array must be of length at least (`end - begin + 1`). If successful, `ub[0]` through `ub[end-begin]` contain the global upper bound values.
begin	An integer specifying the beginning of the range of upper bound values to be returned.
end	An integer specifying the end of the range of upper bound values to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcopybase

```
int CPXcopybase(CPXCENVptr env, CPXLPptr lp, const int * cstat, const int * rstat)
```

Definition file: cplex.h

The routine `CPXcopybase` copies a basis into a CPLEX problem object. It is not necessary to copy a basis prior to optimizing an LP problem, but a good initial basis can increase the speed of optimization significantly. A basis does not need to be primal or dual feasible to be used by the optimizer.

Note

The basis is ignored by the optimizer if `CPX_PARAM_ADVIND` is set to zero.

Table 1: Values of basis status for columns in cstat[j]

`CPX_AT_LOWER`	0	variable at lower bound
`CPX_BASIC`	1	variable is basic
`CPX_AT_UPPER`	2	variable at upper bound
`CPX_FREE_SUPER`	3	variable free and nonbasic

Table 2: Values of basis status for rows other than ranged rows in rstat[j]

`CPX_AT_LOWER`	0	associated slack, surplus, or artificial variable is nonbasic at value 0.0 (zero)
`CPX_BASIC`	1	associated slack, surplus, or artificial variable is basic

Table 3: Values of basis status for ranged rows in rstat[j]

`CPX_AT_LOWER`	0	associated slack, surplus, or artificial variable is nonbasic at its lower bound
`CPX_BASIC`	1	associated slack, surplus, or artificial variable is basic
`CPX_AT_UPPER`	2	associated slack, surplus, or artificial variable is nonbasic at its upper bound

Example

```
status = CPXcopybase (env, lp, cstat, rstat);
```

See Also: CPXreadcopybase

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

lp A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

cstat An array containing the basis status of the columns in the constraint matrix. The length of the array is equal to the number of columns in the problem object. Possible values of the basis status of columns appear in Table 1.

rstat An array containing the basis status of the slack, or surplus, or artificial variable associated with each row in the constraint matrix. The length of the array is equal to the number of rows in the CPLEX problem object. For rows other than ranged rows, the array element `rstat[i]` has the meaning in Table 2. For ranged rows, the array element `rstat[i]` has the meaning in Table 3.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXaddusercuts

```
int CPXaddusercuts(CPXCENVptr env, CPXLPptr lp, int rcnt, int nzcnt, const double *
rhs, const char * sense, const int * rmatbeg, const int * rmatind, const double *
rmatval, char ** rowname)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXaddusercuts` adds constraints to the list of constraints that should be added to the LP subproblem of a MIP optimization if they are violated. CPLEX handles addition of the constraints and makes sure that all integer solutions satisfy all the constraints. The constraints are added to those specified in prior calls to `CPXaddusercuts`.

The constraints must be cuts that are implied by the constraint matrix. The CPLEX parameter `CPX_PARAM_PRELINEAR` should be set to `CPX_OFF` (0).

Use `CPXfreeusercuts` to clear the list of cuts.

The arguments of `CPXaddusercuts` are the same as those of `CPXaddrows`, with the exception that new columns may not be specified, so there are no `ccnt` and `colname` arguments. Furthermore, unlike `CPXaddrows`, `CPXaddusercuts` does not accept a NULL pointer for the array of righthand side values or senses.

Example

```
status = CPXaddusercuts (env, lp, cutcnt, cutnzcnt, cutrhs,
                         cutsense, cutbeg, cutind, cutval, NULL);
```

See also `admipex4.c` in the standard distribution.

Values of sense

`sense[i]`	`= 'L'`	<= constraint
`sense[i]`	`= 'E'`	= constraint
`sense[i]`	`= 'G'`	>= constraint

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
rcnt	An integer that specifies the number of new rows to be added to the constraint matrix.
nzcnt	An integer that specifies the number of nonzero constraint coefficients to be added to the constraint matrix. This specifies the length of the arrays `rmatind` and `rmatval`.
rhs	An array of length `rcnt` containing the righthand side term for each constraint to be added to the CPLEX problem object.
sense	An array of length `rcnt` containing the sense of each constraint to be added to the CPLEX problem object. Possible values of this argument appear in the table.
rmatbeg	An array used with `rmatind` and `rmatval` to define the rows to be added.
rmatind	An array used with `rmatbeg` and `rmatval` to define the rows to be added.

rmatval An array used with `rmatbeg` and `rmatind` to define the rows to be added. The format is similar to the format used to describe the constraint matrix in the routine `CPXcopylp` (see description of `matbeg`, `matcnt`, `matind`, and `matval` in that routine), but the nonzero coefficients are grouped by row instead of column in the array `rmatval`. The nonzero elements of every row must be stored in sequential locations in this array from position `rmatbeg[i]` to `rmatbeg[i+1]-1` (or from `rmatbeg[i]` to `nzcnt -1` if `i=rcnt-1`). Each entry, `rmatind[i]`, specifies the column index of the corresponding coefficient, `rmatval[i]`. Unlike `CPXcopylp`, all rows must be contiguous, and `rmatbeg[0]` must be 0.

rowname An array containing pointers to character strings that represent the names of the user cuts. May be NULL, in which case the new user cuts are assigned default names if the user cuts already resident in the CPLEX problem object have names; otherwise, no names are associated with the user cuts. If row names are passed to `CPXaddusercuts` but existing user cuts have no names assigned, default names are created for them.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcopydnorms

```
int CPXcopydnorms(CPXCENVptr env, CPXLPptr lp, const double * norm, const int *
head, int len)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXcopydnorms` copies the dual steepest-edge norms to the specified LP problem object. The argument `head` is an array of column or row indices corresponding to the array of norms. Column indices are indexed with nonnegative values. Row indices are indexed with negative values offset by 1 (one). For example, if `head[0] = -5`, then `norm[0]` is associated with `row 4`.

See Also: CPXcopypnorms, CPXgetdnorms

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX LP problem object, as returned by `CPXcreateprob`.
norm	An array containing values to be used in a subsequent call to `CPXdualopt`, with a setting of `CPX_PARAM_DPRIIND` equal to 2, as the initial values for the dual steepest-edge norms of the corresponding basic variables specified in `head[]`. The array must be of length at least equal to the value of the argument `len`. If any indices in `head[]` are not basic, the corresponding values in `norm[]` are ignored.
head	An array containing the indices of the basic variables for which norms have been specified in `norm[]`. The array must be of length at least equal to the value of the argument `len`.
len	An integer that specifies the number of entries in `norm[]` and `head[]`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcheckcopylp

```
int CPXcheckcopylp(CPXCENVptr env, CPXCLPptr lp, int numcols, int numrows, int
objsen, const double * obj, const double * rhs, const char * sense, const int *
matbeg, const int * matcnt, const int * matind, const double * matval, const double
* lb, const double * ub, const double * rngval)
```

Definition file: cplex.h

The routine `CPXcheckcopylp` validates the arguments of the corresponding `CPXcopylp` routine. This data checking routine is found in source format in the file `check.c` which is provided with the standard CPLEX distribution. To call this routine, you must compile and link `check.c` with your program as well as the CPLEX Callable Library.

The `CPXcheckcopylp` routine has the same argument list as the `CPXcopylp` routine. The second argument, `lp`, is technically a pointer to a constant LP object of type `CPXCLPptr` rather than type `CPXLPptr`, as this routine will not modify the problem. For most user applications, this distinction is unimportant.

Example

```
status = CPXcheckcopylp (env, lp, numcols, numrows, objsen, obj,
                         rhs, sense, matbeg, matcnt, matind,
                         matval, lb, ub, rngval);
```

Returns:

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.

Global function CPXwriteprob

```
int CPXwriteprob(CPXCENVptr env, CPXCLPptr lp, const char * filename_str, const
char * filetype_str)
```

Definition file: cplex.h

The routine `CPXwriteprob` writes a CPLEX problem object to a file in one of the formats in the table. These formats are documented in the *CPLEX File Formats Reference Manual* and examples of their use appear in the *CPLEX User's Manual*.

File formats

SAV	Binary matrix and basis file
MPS	MPS format
LP	CPLEX LP format with names modified to conform to LP format
REW	MPS format, with all names changed to generic names
RMP	MPS format, with all names changed to generic names
RLP	LP format, with all names changed to generic names

When this routine is invoked, the current problem is written to a file. If the file name ends with one of the following extensions, a compressed file is written.

- `.bz2` for files compressed with BZip2.
- `.gz` for files compressed with GNU Zip.

Microsoft Windows does not support writing compressed files with this API.

Example

```
status = CPXwriteprob (env, lp, "myprob.sav", NULL);
```

See also the example `lpex1.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the problem is to be written, unless otherwise specified with the `filetype` argument. If the file name ends with `.gz` or `.bz2`, a compressed file is written in accordance with the selected file type.
filetype_str	A character string containing the type of the file, which can be one of the values in the table. May be NULL, in which case the type is inferred from the file name. The string is not case sensitive.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXmstwritesolnpoolall

```
int CPXmstwritesolnpoolall(CPXCENVptr env, CPXCLPptr lp, const char * filename_str)
```

Definition file: cplex.h

The routine `CPXmstwritesolnpoolall` writes MIP starts for all of the members of the solution pool to a file in MST format.

The MST format is an XML format and is documented in the stylesheet `solution.xsl` and schema `solution.xsd` in the `include` directory of the CPLEX distribution. *CPLEX File Formats Reference Manual* also documents this format briefly.

This routine is deprecated. Use CPXwritemipstarts instead.

See Also: CPXmstwrite, CPXwritemipstarts

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the MIP start information should be written.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXmstwritesolnpool

```
int CPXmstwritesolnpool(CPXCENVptr env, CPXCLPptr lp, int soln, const char *
filename_str)
```

Definition file: cplex.h

The routine `CPXmstwritesolnpool` writes a MIP start, using either the current MIP start or a MIP start from the solution pool, to a file in MST format.

The MST format is an XML format and is documented in the stylesheet `solution.xsl` and schema `solution.xsd` in the `include` directory of the CPLEX distribution. *CPLEX File Formats Reference Manual* also documents this format briefly.

This routine is deprecated. Use CPXwritemipstarts instead.

See Also: CPXmstwrite, CPXwritemipstarts

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX problem object as returned by `CPXcreateprob`.
soln	An integer specifying the index of the solution pool MIP start which should be written. A value of -1 specifies that the current MIP start should be used instead of a solution pool member.
filename_str	A character string containing the name of the file to which the MIP start information should be written.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETextract

`int` **`CPXNETextract`**`(CPXCENVptr env, CPXNETptr net, CPXCLPptr lp, int * colmap, int * rowmap)`

Definition file: cplex.h

The routine `CPXNETextract` finds an embedded network in the LP stored in a CPLEX problem object and copies it as a network to the network problem object, `net`. The extraction algorithm is controlled by the parameter `CPX_PARAM_NETFIND`.

If the CPLEX problem object has a basis, an attempt is made to copy the basis to the network object. However, this may fail if the status values corresponding to the rows and columns of the subnetworks do not form a basis. Even if the entire LP is a network, it may not be possible to load the basis to the network object if none of the slack or artificial variables are basic.

Example

```
status = CPXNETextract (env, net, lp, colmap, rowmap);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
colmap	If not NULL, after completion `colmap[i]` contains the index of the LP column that has been mapped to arc `i`. If `colmap[i] < 0`, arc i corresponds to the slack variable for row `-colmap[i]-1`. The size of `colmap` must be at least `CPXgetnumcols(env, lp) + CPXgetnumrows(env, lp)`.
rowmap	If not NULL, after completion `rowmap[i]` contains the index of the LP row that has been mapped to node `i`. If `rowmap[i] < 0`, node i is a dummy node that has no corresponding row in the LP. The size of `rowmap` must be least `CPXgetnumrows(env, lp) + 1`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsetbranchcallbackfunc

```
int CPXsetbranchcallbackfunc(CPXENVptr env, int(CPXPUBLIC
*branchcallback)(CALLBACK_BRANCH_ARGS), void * cbhandle)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXsetbranchcallbackfunc` sets and modifies the user-written callback routine to be called after a branch has been selected but before the branch is carried out during MIP optimization. In the callback routine, the CPLEX-selected branch can be changed to a user-selected branch.

Example

```
status = CPXsetbranchcallbackfunc (env, mybranchfunc, mydata);
```

See also the example `admipex1.c` in the standard distribution.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`branchcallback`

A pointer to a user-written branch callback. If the callback is set to NULL, no callback can be called during optimization.

`cbhandle`

A pointer to user private data. This pointer is passed to the callback.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle,
              int        type,
              int        sos,
              int        nodecnt,
              int        bdcnt,
              double     *nodeest,
              int        *nodebeg,
              int        *indices,
              char       *lu,
              int        *bd,
              int        *useraction_p);
```

The call to the branch callback occurs after a branch has been selected but before the branch is carried out. This function is written by the user. On entry to the callback, the CPLEX-selected branch is defined in the arguments. The arguments to the callback specify a list of changes to make to the bounds of variables when child nodes are created. One, two, or zero child nodes can be created, so one, two, or zero lists of changes are specified in the arguments. The first branch specified is considered first. The callback is called with zero lists of bound changes

when the solution at the node is integer feasible. CPLEX occasionally elects to branch by changing a number of bounds on variables or by adding constraints to the node subproblem; the branch type is then `CPX_TYPE_ANY`. The details of the constraints added for a `CPX_TYPE_ANY` branch are not available to the user.

You can implement custom branching strategies by calling the CPLEX routine `CPXbranchcallbackbranchbds`, `CPXbranchcallbackbranchconstraints`, or `CPXbranchcallbackbranchgeneral` and setting the `useraction` argument to `CPX_CALLBACK_SET`. Then CPLEX will carry out these branches instead of the CPLEX-selected branches.

Branch variables are expressed in terms of the original problem if the parameter `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, branch variables are in terms of the presolved problem.

If you set the parameter `CPX_PARAM_MIPCBREDLP` to `CPX_OFF`, you must also disable dual and nonlinear presolve reductions. To do so, set the parameter `CPX_PARAM_REDUCE` to 1 (one), and set the parameter `CPX_PARAM_PRELINEAR` to 0 (zero).

Callback return value

The callback returns zero if successful and nonzero if an error occurs.

Callback arguments

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`cbdata`

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

`wherefrom`

An integer value reporting where in the optimization this function was called. It will have the value `CPX_CALLBACK_MIP_BRANCH`.

`cbhandle`

A pointer to user-private data.

`int type`

An integer that specifies the type of branch. This table summarizes possible values.

Branch Types

Symbolic Constant	Value	Branch
`CPX_TYPE_VAR`	`'0'`	variable branch
`CPX_TYPE_SOS1`	`'1'`	SOS1 branch
`CPX_TYPE_SOS2`	`'2'`	SOS2 branch
`CPX_TYPE_ANY`	`'A'`	multiple bound changes and/or constraints will be used for branching

`sos`

An integer that specifies the special ordered set (SOS) used for this branch. A value of `-1` specifies that this branch is not an SOS-type branch.

`nodecnt`

An integer that specifies the number of nodes CPLEX will create from this branch. Possible values are:

- `0` (zero), or
- `1`, or
- `2`.

If the argument is `0`, the node will be fathomed unless user-specified branches are made; that is, no child nodes are created and the node itself is discarded.

`bdcnt`

An integer that specifies the number of bound changes defined in the arrays `indices`, `lu`, and `bd` that define the CPLEX-selected branch.

`nodeest`

An array with `nodecnt` entries that contains estimates of the integer objective-function value that will be attained from the created node.

`nodebeg`

An array with `nodecnt` entries. The i-th entry is the index into the arrays `indices`, `lu`, and `bd` of the first bound changed for the ith node.

`indices`

Together with `lu` and `bd`, this array defines the bound changes for each of the created nodes. The entry `indices[i]` is the index for the variable.

`lu`

Together with `indices` and `bd`, this array defines the bound changes for each of the created nodes. The entry `lu[i]` is one of the three possible values specifying which bound to change:

- `'L'` for lower bound, or
- `'U'` for upper bound, or
- `'B'` for both bounds.

`bd`

Together with `indices` and `lu`, this array defines the bound changes for each of the created nodes. The entry `bd[i]` specifies the new value of the bound.

`useraction_p`

A pointer to an integer specifying the action for CPLEX to take at the completion of the user callback. The table summarizes the possible actions.

Actions to be Taken After a User-Written Branch Callback

Value	Symbolic Constant	Action
`0`	`CPX_CALLBACK_DEFAULT`	Use CPLEX-selected branch
`1`	`CPX_CALLBACK_FAIL`	Exit optimization
`2`	`CPX_CALLBACK_SET`	Use user-selected branch, as defined by calls to `CPXbranchcallbackbranchbds`

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsolwritesolnpool

```
int CPXsolwritesolnpool(CPXCENVptr env, CPXCLPptr lp, int soln, const char *
filename_str)
```

Definition file: cplex.h

The routine `CPXsolwrite` writes a solution file, using either the incumbent solution or a solution from the solution pool, for the selected CPLEX problem object. The routine writes files in SOL format, which is an XML format.

The SOL format is documented in the stylesheet `solution.xsl` and schema `solution.xsd` in the `include` directory of the CPLEX distribution. *CPLEX File Formats Reference Manual* also documents this format briefly.

Example

```
status = CPXsolwritesolnpool (env, lp, 1, "myfile.sol");
```

See Also: CPXsolwrite, CPXsolwritesolnpoolall

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
soln	An integer specifying the index of the solution pool member which should be written. A value of -1 specifies that the incumbent solution should be used instead of a solution pool member.
filename_str	A character string containing the name of the file to which the solution should be written.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXqpopt

`int` **`CPXqpopt`**`(CPXCENVptr env, CPXLPptr lp)`

Definition file: cplex.h

The routine`CPXqpopt` may be used, at any time after a continuous quadratic program has been created, to find a solution to that problem using one of the CPLEX quadratic optimizers. The parameter `CPX_PARAM_QPMETHOD` controls the choice of optimizer (Dual Simplex, Primal Simplex, or Barrier). With the default setting of this parameter (that is, `Automatic`) CPLEX invokes the barrier method because it is fastest on a wide range of problems.

Example

```
status = CPXqpopt (env, lp);
```

See Also: CPXgetmethod

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (`CPXERR_NO_MEMORY`) or encountering invalid data in the CPLEX problem object (`CPXERR_NO_PROBLEM`). Exceeding a user-specified CPLEX limit, or proving the model infeasible or unbounded are not considered errors. Note that a zero return value does not necessarily mean that a solution exists. Use the query routines `CPXsolninfo`, `CPXgetstat`, and `CPXsolution` to obtain further information about the status of the optimization.

Global function CPXgetmipcallbackfunc

```
int CPXgetmipcallbackfunc(CPXCENVptr env, int(CPXPUBLIC **callback_p)(CPXCENVptr,
void *, int, void *), void ** cbhandle_p)
```

Definition file: cplex.h

The routine `CPXgetmipcallbackfunc` accesses the user-written callback routine to be called prior to solving each subproblem in the branch-and-cut tree during the optimization of a mixed integer program.

This routine works in the same way as the routine `CPXgetlpcallbackfunc`. It enables the user to create a separate callback function to be called during the solution of mixed integer programming problems. The prototype for the callback function is identical to that of `CPXgetlpcallbackfunc`.

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`callback_p`

The address of the pointer to the current user-written callback function. If no callback function has been set, the pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the user's private pointer.

Example

```
status = CPXgetmipcallbackfunc (env, mycallback, NULL);
```

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle);
```

This is the user-written callback routine.

Callback return value

A nonzero terminates the optimization.

Callback arguments

`env`

A pointer to the CPLEX environment that was passed into the associated optimization routine.

`cbdata`

A pointer passed from the optimization routine to the user-written callback function that identifies the LP problem being optimized. The only purpose for the `cbdata` pointer is to pass it to the routine `CPXgetcallbackinfo`.

`wherefrom`

An integer value reporting from which optimization algorithm the user-written callback function was called. Possible values and their meaning appear in this table.

Indicators of algorithm that called user-written callback

Value	Symbolic Constant	Meaning
101	CPX_CALLBACK_MIP	From mipopt
107	CPX_CALLBACK_MIP_PROBE	From probing or clique merging
108	CPX_CALLBACK_MIP_FRACCUT	From Gomory fractional cuts
109	CPX_CALLBACK_MIP_DISJCUT	From disjunctive cuts
110	CPX_CALLBACK_MIP_FLOWMIR	From Mixed Integer Rounding cuts

`cbhandle`

Pointer to user private data, as passed to `CPXsetmipcallbackfunc`.

See Also: CPXgetcallbackinfo

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXtuneparamprobset

```
int CPXtuneparamprobset(CPXENVptr env, int filecnt, char ** filename, char **
filetype, int intcnt, const int * intind, const int * intval, int dblcnt, const int
* dblind, const double * dblval, int strcnt, const int * strind, char ** strval,
int * tunestat_p)
```

Definition file: cplex.h

The routine `CPXtuneparamprobset` tunes the parameters of the CPLEX environment for improved optimizer performance for a set of problems. Tuning is carried out by making a number of trial runs with a variety parameter settings. Parameters and associated values which should not be changed by the tuning process (known as the fixed parameters) can be specified as arguments.

This routine does not apply to network models, nor to quadratically constrained programming problems (QCP).

After `CPXtuneparamprobset` has finished, the environment will contain the combined fixed and tuned parameter settings, which the user can query or write to a file.

All callbacks, except the tuning callback, will be ignored. Tuning will monitor the value set by `CPXsetterminate` and terminate when this value is set.

A few of the parameter settings in the environment control the tuning process. They are specified in the table below; other parameter settings in the environment are ignored.

Parameter	Use
`CPX_PARAM_TILIM`	Limits the total time spent tuning
`CPX_PARAM_TUNINGTILIM`	Limits the time of each trial run
`CPX_PARAM_TUNINGMEASURE`	Controls the tuning evaluation measure
`CPX_PARAM_TUNINGDISPLAY`	Controls the level of the tuning display
`CPX_PARAM_SCRIND`	Controls screen output

The value `tunestat` is 0 (zero) when tuning has completed and nonzero when it has not. The two nonzero statuses are `CPX_TUNE_ABORT`, which will be set when the terminate value passed to `CPXsetterminate` is set, and `CPX_TUNE_TILIM`, which will be set when the time limit specified by `CPX_PARAM_TILIM` is reached. Tuning will set any parameters which have been chosen even when tuning is not completed.

```
status = CPXtuneparamprobset (env, filecnt, filenames, filetypes,
                              icnt, inum, ival, dcnt, dnum, dval,
                              0, NULL, NULL, &tunestat);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
filecnt	An integer that specifies the number of problem files.
filename	An array of length `filecnt` containing problem file names.
filetype	An array of length `filecnt` containing problem file types, as documented in `CPXreadcopyprob`. May be NULL; then CPLEX discerns file types from the file extensions of the file names.
intcnt	An integer that specifies the number of integer parameters to be fixed during tuning. This argument specifies the length of the arrays `intnum` and `intval`.
intval	An array containing the values for the parameters listed in `intnum`. May be NULL if `intcnt` is 0 (zero).
dblcnt	An integer that specifies the number of double parameters to be fixed during tuning. This specifies the length of the arrays `dblnum` and `dblval`.

dblval	An array containing the values for the parameters listed in `dblnum`. May be NULL if `dblcnt` is 0 (zero).
strcnt	An integer that specifies the number of string parameters to be fixed during tuning. This specifies the length of the arrays `strnum` and `strval`.
strval	An array containing the values for the parameters listed in `strnum`. May be NULL if `strcnt` is 0 (zero).
tunestat_p	A pointer to an integer to receive the tuning status.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXreadcopyparam

`int` **`CPXreadcopyparam`**`(CPXENVptr env, const char * filename_str)`

Definition file: cplex.h

The routine `CPXreadcopyparam` reads parameter names and settings from the file specified by `filename_str` and copies them into CPLEX.

This routine reads and copies files in the PRM format, as created by `CPXwriteparam`. The PRM format is documented in the *CPLEX File Formats Reference Manual*.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
filename_str	Name of the file to read and copy into CPLEX.

Global function CPXgetnumsemicont

`int` **`CPXgetnumsemicont`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetnumsemicont` accesses the number of semi-continuous variables in a CPLEX problem object.

Example

```
numsc = CPXgetnumsemicont (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If the problem object or environment does not exist, `CPXgetnumsemicont` returns the value 0 (zero); otherwise, it returns the number of semi-continuous variables in the problem object.

Global function CPXNETcheckcopynet

```
int CPXNETcheckcopynet(CPXCENVptr env, CPXNETptr net, int objsen, int nnodes, const
double * supply, char ** nnames, int narcs, const int * fromnode, const int *
tonode, const double * low, const double * up, const double * obj, char ** aname)
```

Definition file: cplex.h

The routine `CPXNETcheckcopynet` performs a consistency check on the arguments passed to the routine `CPXNETcopynet`.

The `CPXNETcheckcopynet` routine has the same argument list as the `CPXNETcopynet` routine.

Example

```
status = CPXNETcheckcopynet (env, net, CPX_MAX, nnodes, supply,
                             nnames, narcs, fromnode, tonode,
                             lb, ub, obj, anames);
```

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetprotected

```
int CPXgetprotected(CPXCENVptr env, CPXCLPptr lp, int * cnt_p, int * indices, int
pspace, int * surplus_p)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetprotected` accesses the set of variables that cannot be aggregated out.

> **Note**
>
> If the value of `pspace` is 0, the negative of the value of `surplus_p` returned specifies the length needed for array `indices`.

Example

```
status = CPXgetprotected (env, lp, &protectcnt,
                          protectind, 10, &surplus);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
cnt_p	A pointer to an integer to contain the number of protected variables returned, that is, the true length of the array `indices`.
indices	The array to contain the indices of the protected variables.
pspace	An integer specifying the length of the array `indices`.
surplus_p	A pointer to an integer to contain the difference between `pspace` and the number of entries in `indices`. A nonnegative value of `surplus_p` specifies that the length of the arrays was sufficient. A negative value specifies that the length was insufficient and that the routine could not complete its task. In that case, the routine `CPXgetprotected` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the value of `surplus_p` specifies the amount of insufficient space in the arrays.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` specifies that insufficient space was available in the array `indices` to hold the protected variable indices.

Global function CPXgetrowname

```
int CPXgetrowname(CPXCENVptr env, CPXCLPptr lp, char ** name, char * namestore, int
storespace, int * surplus_p, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetrowname` accesses a range of row names or, equivalently, the constraint names of a CPLEX problem object. The beginning and end of the range, along with the length of the array in which the row names are to be returned, must be specified.

Note

If the value of `storespace` is 0, then the negative of the value of `surplus_p` returned specifies the total number of characters needed for the array `namestore`.

Example

```
status = CPXgetrowname (env, lp, cur_rowname, cur_rownamestore,
                        cur_storespace, &surplus, 0,
                        cur_numrows-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
name	An array of pointers to the row names stored in the array `namestore`. This array must be of length at least (`end - begin + 1`). The pointer to the name of row `i` is returned in `name[i-begin]`.
namestore	An array of characters where the specified row names are to be returned. May be NULL if `storespace` is 0.
storespace	An integer specifying the length of the array `namestore`. May be 0.
surplus_p	A pointer to an integer to contain the difference between `storespace` and the total amount of memory required to store the requested names. A nonnegative value of `surplus_p` specifies that `storespace` was sufficient. A negative value specifies that it was insufficient and that the routine could not complete its task. In that case, `CPXgetrowname` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `surplus_p` specifies the amount of insufficient space in the array `namestore`.
begin	An integer specifying the beginning of the range of row names to be returned.
end	An integer specifying the end of the range of row names to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` specifies that insufficient space was available in the `namestore` array to hold the names.

Global function CPXNETchgbds

`int` **`CPXNETchgbds`**`(CPXCENVptr env, CPXNETptr net, int cnt, const int * indices, const char * lu, const double * bd)`

Definition file: cplex.h

The routine `CPXNETchgbds` is used to change the upper, lower, or both bounds on the flow for a set of arcs in the network stored in a network problem object. The flow value of an arc can be fixed to a value by setting both bounds to that value.

Any solution information stored in the problem object is lost.

Example

```
status = CPXNETchgbds (env, net, cnt, index, lu, bd);
```

Indicators to change lower, upper bounds of flows through arcs

`lu[i] == 'L'`	The lower bound of arc `index[i]` is changed to `bd[i]`
`lu[i] == 'U'`	The upper bound of arc `index[i]` is changed to `bd[i]`
`lu[i] == 'B'`	Both bounds of arc `index[i]` are changed to `bd[i]`

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
cnt	Number of bounds to change.
indices	An array of arc indices that indicate the bounds to be changed. This array must have a length of at least `cnt`. All indices must be in the range [`0, narcs-1`].
lu	An array indicating which bounds to change. This array must have a length of at least `cnt`. The indicators appear in the table.
bd	An array of bound values. This array must have a length of at least `cnt`. Values greater than or equal to `CPX_INFBOUND` and less than or equal to `-CPX_INFBOUND` are considered infinity or -infinity, respectively.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXchgsense

```
int CPXchgsense(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices, const
char * sense)
```

Definition file: cplex.h

The routine `CPXchgsense` changes the sense of a set of linear constraints of a CPLEX problem object. When changing the sense of a row to ranged, `CPXchgsense` sets the corresponding range value to 0 (zero). The routine `CPXchgrngval` can then be used to change the range value.

Example

```
status = CPXchgsense (env, lp, cnt, indices, sense);
```

Values of sense

`sense[i]`	`= 'L'`	The new sense is <=
`sense[i]`	`= 'E'`	The new sense is =
`sense[i]`	`= 'G'`	The new sense is >=
`sense[i]`	`= 'R'`	The constraint is ranged

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt	An integer that specifies the total number of linear constraints to be changed, and thus represents the length of the arrays `indices` and `sense`.
indices	An array of length `cnt` containing the numeric indices of the rows corresponding to the linear constraints which are to have their senses changed.
sense	An array of length `cnt` containing characters that tell the new sense of the linear constraints specified in `indices`. Possible values appear in the table.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETgetx

```
int CPXNETgetx(CPXCENVptr env, CPXCNETptr net, double * x, int begin, int end)
```

Definition file: cplex.h

The routine `CPXNETgetx` is used to access solution values or, equivalently, flow values for a range of arcs stored in a network problem object.

For this routine to succeed, a solution must exist for the network problem object.

Example

```
status = CPXNETgetx (env, net, x, 10, 20);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
x	Array in which to write solution (or flow) values for requested arcs. If NULL is passed, no solution vector is returned. Otherwise, `x` must point to an array of size at least (`end-begin+1`).
begin	Index of the first arc for which a solution (or flow) value is to be obtained.
end	Index of the last arc for which a solution (or flow) value is to be obtained.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXpreaddrows

```
int CPXpreaddrows(CPXCENVptr env, CPXLPptr lp, int rcnt, int nzcnt, const double *
rhs, const char * sense, const int * rmatbeg, const int * rmatind, const double *
rmatval, char ** rowname)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXpreaddrows` adds rows to an LP problem object and its associated presolved LP problem object. The CPLEX parameter `CPX_PARAM_REDUCE` must be set to `CPX_PREREDUCE_PRIMALONLY (1)` or `CPX_PREREDUCE_NOPRIMALORDUAL (0)` at the time of the presolve in order to add rows and preserve the presolved problem. This routine should be used in place of `CPXaddrows)` when you want to preserve the presolved problem.

The arguments of `CPXpreaddrows` are the same as those of `CPXaddrows`, with the exception that new columns may not be added, so there are no `ccnt` and `colname` arguments. The new rows are added to both the original LP problem object and the associated presolved LP problem object.

Examples:

```
status = CPXpreaddrows (env, lp, rcnt, nzcnt, rhs, sense, rmatbeg, rmatind,
                        rmatval, newrowname);
```

See also the example `adpreex1.c` in the standard distribution.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsolwrite

```
int CPXsolwrite(CPXCENVptr env, CPXCLPptr lp, const char * filename_str)
```

Definition file: cplex.h

The routine `CPXsolwrite` writes a solution file for the selected CPLEX problem object. The routine writes files in SOL format, which is an XML format.

The SOL format is documented in the stylesheet `solution.xsl` and schema `solution.xsd` in the `include` directory of the CPLEX distribution. *CPLEX File Formats Reference Manual* also documents this format briefly.

Example

```
status = CPXsolwrite (env, lp, "myfile.sol");
```

See Also: CPXsolwritesolnpool, CPXsolwritesolnpoolall

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the solution should be written.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcheckcopyctype

```
int CPXcheckcopyctype(CPXCENVptr env, CPXCLPptr lp, const char * xctype)
```

Definition file: cplex.h

The routine `CPXcheckcopyctype` validates the arguments of the corresponding `CPXcopyctype` routine. This data checking routine is found in source format in the file `check.c` which is provided with the standard CPLEX distribution. To call this routine, you must compile and link `check.c` with your program as well as the CPLEX Callable Library.

The `CPXcheckcopyctype` routine has the same argument list as the `CPXcopyctype` routine. The second argument, `lp`, is technically a pointer to a constant LP object of type `CPXCLPptr` rather than type `CPXLPptr`, as this routine will not modify the problem. For most user applications, this distinction is unimportant.

Example

```
status = CPXcheckcopyctype (env, lp, ctype);
```

Returns:

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.

Global function CPXgetstat

```
int CPXgetstat(CPXCENVptr env, CPXCLPptr lp)
```

Definition file: cplex.h

The routine `CPXgetstat` accesses the solution status of the problem after an LP, QP, QCP, or MIP optimization, after `CPXfeasopt` and its extensions, after `CPXrefineconflict` and its extensions.

Example

```
lpstat = CPXgetstat (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns the solution status of the most recent optimization performed on the CPLEX problem object. Nonzero return values are shown in the group optim.cplex.solutionstatus. A return value of 0 (zero) specifies either an error condition or that a change to the most recently optimized problem may have invalidated the solution status. For status code `CPX_STAT_NUM_BEST`, the algorithm could not converge to the requested tolerances due to numeric difficulties.

The best solution found can be retrieved by the routine `CPXsolution`. Similarly, when an abort status is returned, the last solution computed before the algorithm aborted can be retrieved by `CPXsolution`.

Use the query routines `CPXsolninfo` and `CPXsolution` to obtain further information about the current solution of an LP, QP, or QCP.

Global function CPXtightenbds

```
int CPXtightenbds(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices, const
char * lu, const double * bd)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXtightenbds` changes the upper or lower bounds on a set of variables in a problem. Several bounds can be changed at once. Each bound is specified by the index of the variable associated with it. The value of a variable can be fixed at one value by setting both the upper and lower bounds to the same value.

In contrast to the Callable Library routine `CPXchgbds`, also used to change bounds, `CPXtightenbds`preserves more of the internal CPLEX data structures so it is more efficient for re-optimization, particularly when changes are made to bounds on basic variables.

Bound Indicators in the argument lu of CPXtightenbds

Value of lu[j]	Meaning for bd[j]
`U`	`bd[j]`is an upper bound
`L`	`bd[j]`is a lower bound
`B`	`bd[j]`is the lower and upper bound

Example

```
status = CPXtightenbds (env, lp, cnt, indices, lu, bd);
```

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
cnt	An integer specifying the total number of bounds to change. That is, `cnt` specifies the length of the arrays `indices`, `lu`, and `bd`.
indices	An array containing the numeric indices of the columns corresponding to the variables for which bounds will be changed. The allocated length of the array is `cnt`. Column j of the constraint matrix has the internal index `j - 1`.
lu	An array. This array contains characters specifying whether the corresponding entry in the array `bd` specifies the lower or upper bound on column `indices[j]`. The allocated length of the array is `cnt`. The table summarizes the values that entries in this array may assume.
bd	An array. This array contains the new values of the upper or lower bounds of the variables present in the array `indices`. The allocated length of the array is `cnt`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXembwrite

```
int CPXembwrite(CPXCENVptr env, CPXLPptr lp, const char * filename_str)
```

Definition file: cplex.h

The routine `CPXembwrite` writes out the network embedded in the selected problem object. MPS format is used. The specific network extracted depends on the current setting of the parameter `CPX_PARAM_NETFIND`.

Example

```
status = CPXembwrite (env, lp, "myfile.emb");
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the embedded network should be written.

Example

```
status = CPXembwrite (env, lp, "myfile.emb");
```

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetdsbcnt

`int` **`CPXgetdsbcnt`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetdsbcnt` accesses the number of dual super-basic variables in the current solution.

Example

```
dsbcnt = CPXgetdsbcnt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Example

```
dsbcnt = CPXgetdsbcnt (env, lp);
```

Returns:

If a solution exists, `CPXgetdsbcnt` returns the number of dual super-basic variables. If no solution exists, `CPXgetdsbcnt` returns the value 0 (zero).

Global function CPXchgprobtypesolnpool

`int` **`CPXchgprobtypesolnpool`**`(CPXCENVptr env, CPXLPptr lp, int type, int soln)`

Definition file: cplex.h

The routine `CPXchgprobtypesolnpool` changes the current problem, if it is a mixed integer problem, to a related fixed problem using a solution from the solution pool. The problem types that can be used appear in the table.

Table 1: Problem Types

Value	Symbolic Constant	Meaning
3	`CPXPROB_FIXEDMILP`	Problem with `ctype` information, integer variables fixed.
8	`CPXPROB_FIXEDMIQP`	Problem with quadratic data and `ctype` information, integer variables fixed.

A mixed integer problem (`CPXPROB_MILP`, `CPXPROB_MIQP`) can be changed to a fixed problem (`CPXPROB_FIXEDMILP`, `CPXPROB_FIXEDMIQP`) where bounds on integer variables are fixed to the values attained in the integer solution.

Example

```
status = CPXchgprobtypesolnpool (env, lp, 1, CPXPROB_FIXEDMILP);
```

See Also: CPXchgprobtype

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

lp A pointer to a CPLEX LP problem object as returned by `CPXcreateprob`.

type An integer specifying the target problem type.

soln An integer specifying the index of the solution pool member whose values are to be used. A value of -1 specifies that the incumbent solution should be used instead of a solution pool member.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetnodecallbackfunc

```
void CPXgetnodecallbackfunc(CPXCENVptr env, int(CPXPUBLIC
**nodecallback_p)(CALLBACK_NODE_ARGS), void ** cbhandle_p)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetnodecallbackfunc` accesses the user-written callback to be called during MIP optimization after CPLEX has selected a node to explore, but before this exploration is carried out. The callback routine can change the node selected by CPLEX to a node selected by the user.

For documentation of callback arguments, see the routine `CPXsetnodecallbackfunc`.

Example

```
CPXgetnodecallbackfunc(env, &current_callback, &current_handle);
```

See also the example `admipex1.c` in the standard distribution.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`nodecallback_p`

The address of the pointer to the current user-written node callback. If no callback has been set, the pointer will evaluate to NULL.

`cbhandle_p`

The address of a variable to hold the user's private pointer.

Returns:

This routine does not return a result.

Global function CPXgetobjval

```
int CPXgetobjval(CPXCENVptr env, CPXCLPptr lp, double * objval_p)
```

Definition file: cplex.h

The routine `CPXgetobjval` accesses the solution objective value.

Example

```
status = CPXgetobjval (env, lp, &objval);
```

See also the example `lpex2.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
objval_p	A pointer to a variable of type `double` where the objective value is stored.

Returns:

The routine returns zero if successful and nonzero if no solution exists.

Global function CPXgetpsbcnt

`int` **`CPXgetpsbcnt`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetpsbcnt` accesses the number of primal super-basic variables in the current solution.

Example

```
psbcnt = CPXgetpsbcnt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Example

```
psbcnt = CPXgetpsbcnt (env, lp);
```

Returns:

If a solution exists, `CPXgetpsbcnt` returns the number of primal super-basic variables. If no solution exists, `CPXgetpsbcnt` returns the value 0 (zero).

Global function CPXgetintparam

`int` **`CPXgetintparam`**`(CPXCENVptr env, int whichparam, int * value_p)`

Definition file: cplex.h

The routine `CPXgetintparam` obtains the current value of a CPLEX parameter of type `int`.

The *CPLEX Parameters Reference Manual* provides a list of parameters with their types, options, and default values.

Example

```
status = CPXgetintparam (env, CPX_PARAM_PREIND, &curpreind);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
whichparam	The symbolic constant (or reference number) of the parameter for which the value is to be obtained.
value_p	A pointer to an integer variable to hold the current value of the CPLEX parameter.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXputenv

```
int CPXputenv(const char * envsetting_str)
```

Definition file: cplex.h

The routine `CPXputenv` sets an environment variable to be used by CPLEX. Use it instead of the standard C Library `putenv` function to make sure your application ports properly to Windows. Be sure to allocate the memory dynamically for the string passed to `CPXputenv`.

As with the C `putenv` routine, the address of the character string goes directly into the environment. Therefore, the memory identified by the pointer must remain active throughout the remaining parts of the application where CPLEX runs. Since global or static variables are not thread safe, the team recommends dynamic memory allocation of the `envsetting` string.

Example

```
char *envstr = NULL;
envstr = (char *) malloc (256);
if ( envstr != NULL ) {
   strcpy (envstr,
           "ILOG_LICENSE_FILE=c:myapplicenseaccess.ilm");
   CPXputenv (envstr);
}
```

Parameters:

envsetting_str	A string containing an environment variable assignment. This argument typically sets the `ILOG_LICENSE_FILE` environment variable that customizes the location of the license key.

Returns:

The routine returns 0 (zero) when it executes successfully and -1 when it fails.

Global function CPXaddlazyconstraints

```
int CPXaddlazyconstraints(CPXCENVptr env, CPXLPptr lp, int rcnt, int nzcnt, const
double * rhs, const char * sense, const int * rmatbeg, const int * rmatind, const
double * rmatval, char ** rowname)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXaddlazyconstraints` adds constraints to the list of constraints that should be added to the LP subproblem of a MIP optimization if they are violated. CPLEX handles addition of the constraints and makes sure that all integer solutions satisfy all the constraints. The constraints are added to those specified in prior calls to `CPXaddlazyconstraints`.

Lazy constraints are constraints not specified in the constraint matrix of the MIP problem, but that must be not be violated in a solution. Using lazy constraints makes sense when there are a large number of constraints that must be satisfied at a solution, but are unlikely to be violated if they are left out.

The CPLEX parameter `CPX_PARAM_REDUCE` should be set to `CPX_PREREDUCE_NOPRIMALORDUAL (0)` or to `CPX_PREREDUCE_PRIMALONLY (1)` in order to turn off dual reductions.

Use `CPXfreelazyconstraints` to clear the list of lazy constraints.

The arguments of `CPXaddlazyconstraints` are the same as those of `CPXaddrows`, with the exception that new columns may not be specified, so there are no `ccnt` and `colname` arguments. Furthermore, unlike `CPXaddrows`, `CPXaddlazyconstraints` does not accept a NULL pointer for the array of righthand side values or senses.

Example

```
status = CPXaddlazyconstraints (env, lp, cnt, nzcnt, rhs, sense,
                                beg, ind, val, NULL);
```

Values of sense

`sense[i]`	`= 'L'`	<= constraint
`sense[i]`	`= 'E'`	= constraint
`sense[i]`	`= 'G'`	>= constraint

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
rcnt	An integer that specifies the number of new lazy constraints to be added.
nzcnt	An integer that specifies the number of nonzero constraint coefficients to be added to the constraint matrix. This specifies the length of the arrays `rmatind` and `rmatval`.
rhs	An array of length `rcnt` containing the righthand side (RHS) term for each lazy constraint to be added to the CPLEX problem object.
sense	An array of length `rcnt` containing the sense of each lazy constraint to be added to the CPLEX problem object. Possible values of this argument appear in the table.

rmatbeg An array used with `rmatind` and `rmatval` to define the lazy constraints to be added.

rmatind An array used with `rmatbeg` and `rmatval` to define the lazy constraints to be added.

rmatval An array used with `rmatbeg` and `rmatind` to define the lazy constraints to be added. The format is similar to the format used to describe the constraint matrix in the routine `CPXcopylp` (see description of `matbeg, matcnt`, `matind`, and `matval` in that routine), but the nonzero coefficients are grouped by row instead of column in the array `rmatval`. The nonzero elements of every lazy constraint must be stored in sequential locations in this array from position `rmatbeg[i]` to `rmatbeg[i+1]-1` (or from `rmatbeg[i]` to `nzcnt -1` if `i=rcnt-1`). Each entry, `rmatind[i]`, specifies the column index of the corresponding coefficient, `rmatval[i]`. Unlike `CPXcopylp`, all rows must be contiguous, and `rmatbeg[0]` must be 0 (zero).

rowname An array containing pointers to character strings that represent the names of the lazy constraints. May be NULL, in which case the new lazy constraints are assigned default names if the lazy constraints already resident in the CPLEX problem object have names; otherwise, no names are associated with the lazy constraints. If row names are passed to `CPXaddlazyconstraints` but existing lazy constraints have no names assigned, default names are created for them.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcheckcopylpwnames

```
int CPXcheckcopylpwnames(CPXCENVptr env, CPXCLPptr lp, int numcols, int numrows,
int objsen, const double * obj, const double * rhs, const char * sense, const int *
matbeg, const int * matcnt, const int * matind, const double * matval, const double
* lb, const double * ub, const double * rngval, char ** colname, char ** rowname)
```

Definition file: cplex.h

The routine `CPXcheckcopylpwnames` validates the arguments of the corresponding `CPXcopylpwnames` routine. This data checking routine is found in source format in the file `check.c` which is provided with the standard CPLEX distribution. To call this routine, you must compile and link `check.c` with your application as well as the CPLEX Callable Library.

The routine `CPXcheckcopylpwnames` has the same argument list as the routine `CPXcopylpwnames`. The second argument, `lp`, is technically a pointer to a constant LP object of type `CPXCLPptr` rather than type `CPXLPptr`, as this routine will not modify the problem. For most user applications, this distinction is unimportant.

Example

```
status = CPXcheckcopylpwnames (env,
                               lp,
                               numcols,
                               numrows,
                               objsen,
                               obj,
                               rhs,
                               sense,
                               matbeg,
                               matcnt,
                               matind,
                               matval,
                               lb,
                               ub,
                               rngval,
                               colname,
                               rowname);
```

Returns:

The routine returns nonzero if it detects an error in the data; it returns zero if it does not detect any data errors.

Global function CPXgetrngval

`int` **`CPXgetrngval`**`(CPXCENVptr env, CPXCLPptr lp, double * rngval, int begin, int end)`

Definition file: cplex.h

The routine `CPXgetrngval` accesses the RHS range coefficients for a set of constraints in a CPLEX problem object. The beginning and end of the set must be specified. `CPXgetrngval` checks if ranged constraints are present in the problem object.

Example

```
status = CPXgetrngval (env, lp, rngval, 0, cur_numrows-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
rngval	An array where RHS range coefficients are returned. This array must be of length at least (`end - begin + 1`). A value of 0 for any entry means that the corresponding row is not ranged.
begin	An integer specifying the beginning of the set of rows for which RHS range coefficients are returned.
end	An integer specifying the end of the set of rows for which RHS range coefficients are returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcloseCPLEX

```
int CPXcloseCPLEX(CPXENVptr * env_p)
```

Definition file: cplex.h

This routine frees all of the data structures associated with CPLEX and releases the license. It should be the last CPLEX routine called in any Callable Library application.

Example

```
status = CPXcloseCPLEX (&env);
```

See also `lpex1.c` in the *CPLEX User's Manual*.

Parameters:

env_p A pointer to a variable holding the pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetnodeint

`int` **`CPXgetnodeint`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetnodeint` accesses the node number of the best known integer solution.

Example

```
nodeint = CPXgetnodeint (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Example

```
nodeint = CPXgetnodeint (env, lp);
```

Returns:

If no solution, problem, or environment exists, CPXgetnodeint returns a value of -1; otherwise, `CPXgetnodeint` returns the node number.

Global function CPXchgmipstart

`int` **`CPXchgmipstart`**`(CPXCENVptr env, CPXLPptr lp, int cnt, const int * indices, const double * values)`

Definition file: cplex.h

This routine is **deprecated**. See `CPXchgmipstarts` instead to change values of several MIP starts.

The routine `CPXchgmipstart` modifies or extends the incumbent MIP start. If the existing incumbent MIP start has no value for the variable `x[j]`, for example, and the call to `CPXchgmipstart` specifies a start value, then the specified value is added to the incumbent MIP start. If the existing incumbent MIP start already has a value for `x[j]`, then the new value replaces the old. If the problem has no MIP start, `CPXchgmipstart` creates one. Start values may be specified for both integer and continuous variables.

See the routine `CPXcopymipstart` for more information about how CPLEX uses MIP start information.

Example

```
status = CPXchgmipstart (env, lp, cnt, indices, values);
```

See Also: CPXcopymipstart, CPXchgmipstarts

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt	An integer giving the number of entries in the list.
indices	An array of length `cnt` containing the numeric indices of the columns corresponding to the variables which are assigned starting values.
values	An array of length `cnt` containing the values to use for the starting integer solution. The entry `values[j]` is the value assigned to the variable `indices[j]`. An entry `values[j]` greater than or equal to `CPX_INFBOUND` specifies that no value is set for the variable `indices[j]`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXdelsetsos

`int` **`CPXdelsetsos`**`(CPXCENVptr env, CPXLPptr lp, int * delset)`

Definition file: cplex.h

The routine `CPXdelsetsos` deletes a group of special ordered sets (SOSs) from a CPLEX problem object.

> **Note**
>
> The `delstat` array must have at least `CPXgetnumsos(env,lp)` elements.

Example

```
status = CPXdelsetsos (env, lp, delstat);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
delset	An array specifying the SOSs to be deleted. The routine `CPXdelsetsos` deletes each SOS `j` for which `delstat[j] = 1`. The deletion of SOSs results in a renumbering of the remaining SOSs. After termination, `delstat[j]` is either -1 for SOSs that have been deleted or the new index number that has been assigned to the remaining SOSs.

> **Note**
>
> The `delstat` array must have at least `CPXgetnumsos(env,lp)` elements.

Example

```
status = CPXdelsetsos (env, lp, delstat);
```

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXordwrite

```
int CPXordwrite(CPXCENVptr env, CPXCLPptr lp, const char * filename_str)
```

Definition file: cplex.h

The routine `CPXordwrite` writes a priority order to an ORD file. If a priority order has been associated with the CPLEX problem object, or the parameter `CPX_PARAM_MIPORDTYPE` is nonzero, or a MIP feasible solution exists, this routine writes the priority order into a file.

Example

```
status = CPXordwrite (env, lp, "myfile.ord");
```

See also the example `mipex3.c` in the standard distribution.

See Also: CPXreadcopyorder

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the ORD information should be written.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetrhs

`int` **`CPXgetrhs`**`(CPXCENVptr env, CPXCLPptr lp, double * rhs, int begin, int end)`

Definition file: cplex.h

The routine `CPXgetrhs` accesses the righthand side coefficients for a range of constraints in a CPLEX problem object. The beginning and end of the range must be specified.

Example

```
status = CPXgetrhs (env, lp, rhs, 0, cur_numrows-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
rhs	An array where the specified righthand side coefficients are to be returned. This array must be of length at least (`end - begin + 1`). The righthand side of constraint `i` is returned in `rhs[i - begin]`.
begin	An integer specifying the beginning of the range of righthand side terms to be returned.
end	An integer specifying the end of the range of righthand side terms to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsolnpoolnummipstarts

`int` **`CPXgetsolnpoolnummipstarts`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

This routine is deprecated. Use `CPXgetnummipstarts` instead.

The routine `CPXgetsolnpoolnummipstarts` accesses the number of MIP starts stored in the solution pool of a CPLEX problem object.

Example

```
status = CPXgetsolnpoolnummipstarts (env, lp);
```

See Also: CPXgetnummipstarts

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If the CPLEX problem object or environment does not exist, `CPXgetsolnpoolnummipstarts` returns the value 0 (zero); otherwise, it returns the number of MIP starts.

Global function CPXgetlogfile

```
int CPXgetlogfile(CPXCENVptr env, CPXFILEptr * logfile_p)
```

Definition file: cplex.h

The routine `CPXgetlogfile` accesses the log file to which messages from all four CPLEX-defined channels are written.

Example

```
status = CPXgetlogfile (env, &logfile);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
logfile_p	The address of a `CPXFILEptr` variable. This routine sets `logfile_p` to be the file pointer for the current log file.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetcallbackindicatorinfo

```
int CPXgetcallbackindicatorinfo(CPXCENVptr env, void * cbdata, int wherefrom, int
iindex, int whichinfo, void * result_p)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbackindicatorinfo` accesses information about the indicator constraints of the presolved problem during MIP callbacks. When indicator constraints are present, CPLEX creates a presolved problem with indicator constraints in canonical form, regardless of the presolve settings.

```
Canonical Form
(implying variable = { 0 | 1 }) IMPLIES (implied variable) R rhs
```

In that canonical form, rhs stands for righthand side and R stands for one of these relations:

- less than or equal to
- greater than or equal to
- equal to

In the original problem, you may have indicator constraints in which the implied constraint has two or more variables. In contrast, in the canonical form, the implied constraint can have only one variable; moreover, its coefficient in the constraint must be 1 (one). For example, CPLEX transforms the indicator constraint

```
x = 0 -> 3y + z <= 0
```

into canonical form by introducing an implied variable `w`, like this:

```
w = 3y + z
x = 0 -> w <= 0
```

The argument `which_info` can assume one of the following values in a call to `CPXgetcallbackindicatorinfo`:

- `CPX_CALLBACK_INFO_IC_NUM` returns the number of indicator constraints.
- `CPX_CALLBACK_INFO_IC_IMPLYING_VAR` returns the index of the implying variable of the `iindex`-th indicator constraint. If the MIP callback parameter for the reduced LP (`CPX_PARAM_MIPCBREDLP`) is off (that is, set to `CPX_OFF`), the index is in terms of the original problem, and if the index = -1, then the variable has been created by presolve. Otherwise, the index is in terms of the presolved problem.
- `CPX_CALLBACK_INFO_IC_IMPLIED_VAR` returns the index of the implied variable of the `iindex`-th indicator constraint. If `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`, the index is in terms of the original problem, and if the index = -1, then the variable has been created by presolve. Otherwise, the index is in terms of the presolved problem.
- `CPX_CALLBACK_INFO_IC_SENSE` returns the sense of the `iindex`-th indicator constraint.
- `CPX_CALLBACK_INFO_IC_COMPL` returns 0 (zero) if the `iindex`-th indicator constraint is **not** complemented, and 1 (one) otherwise.
- `CPX_CALLBACK_INFO_IC_RHS` returns the righthand side of the `iindex`-th indicator constraint.
- `CPX_CALLBACK_INFO_IC_IS_FEASIBLE` returns 1 (one) if the implying variable is not 0 (zero) or 1 (one), or if the `iindex`-th indicator constraint is satisfied at the current node; otherwise, it returns 0 (zero).

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value that reports where the user-written callback was called from. This argument must be the value of `wherefrom` passed to the user-written callback.
iindex	An integer, the index of the indicator constraint.
result_p	A generic pointer to a variable of type `double` or `int`, representing the value returned by `whichinfo`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetprobname

```
int CPXgetprobname(CPXCENVptr env, CPXCLPptr lp, char * buf_str, int bufspace, int
* surplus_p)
```

Definition file: cplex.h

The routine `CPXgetprobname` accesses the name of the problem set via the call to `CPXcreateprob`.

> **Note**
>
> If the value of `bufspace` is 0, then the negative of the value of `surplus_p` returned specifies the total number of characters needed for the array `buf_str`.

Example

```
status = CPXgetprobname (env, lp, cur_probname, lenname,
                         &surplus);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
buf_str	A pointer to a buffer of size `bufspace`. May be NULL if `bufspace` is 0.
bufspace	An integer specifying the length of the array `buf_str`. May be 0.
surplus_p	A pointer to an integer to contain the difference between `bufspace` and the amount of memory required to store the problem name. A nonnegative value of `surplus_p` specifies that the length of the array `buf_str` was sufficient. A negative value specifies that the length of the array was insufficient and that the routine could not complete its task. In this case, `CPXgetprobname` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `surplus_p` specifies the amount of insufficient space in the array `buf_str`.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` specifies that insufficient space was available in the `buf_str` array to hold the problem name.

Global function CPXgetsolnpoolnumreplaced

`int` **`CPXgetsolnpoolnumreplaced`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetsolnpoolnumreplaced` accesses the number of solutions replaced in the solution pool.

Example

```
numrep = CPXgetsolnpoolnumreplaced (env, lp);
```

See also the example `populate.c` in the in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

If the CPLEX problem object or environment does not exist, `CPXgetsolnpoolnumreplaced` returns the value 0 (zero); otherwise, it returns the number of solutions which were replaced.

Global function CPXgetcallbacknodestat

```
int CPXgetcallbacknodestat(CPXCENVptr env, void * cbdata, int wherefrom, int *
nodestat_p)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbacknodestat` retrieves the optimization status of the subproblem at the current node from within a user-written callback during MIP optimization.

The optimization status will be either optimal or unbounded. An unbounded status can occur when some of the constraints are being treated as lazy constraints. When the node status is unbounded, then the function `CPXgetcallbacknodex` returns a ray that can be used to decide which lazy constraints need to be added to the subproblem.

This routine may be called only when the value of the `wherefrom` argument is `CPX_CALLBACK_MIP_CUT`.

Example

```
status = CPXgetcallbacknodestat (env, cbdata, wherefrom,
                                 &nodestatus);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting from where the user-written callback was called. The argument must be the value of `wherefrom` passed to the user-written callback.
nodestat_p	A pointer to an integer where the node subproblem optimization status is to be returned. The values of `*nodestat_p` may be `CPX_STAT_OPTIMAL` or `CPX_STAT_UNBOUNDED`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXNETdelset

```
int CPXNETdelset(CPXCENVptr env, CPXNETptr net, int * whichnodes, int * whicharcs)
```

Definition file: cplex.h

The routine `CPXNETdelset` is used to delete a set of nodes and arcs from the network stored in a network problem object. The remaining nodes and arcs are renumbered starting at zero; their order is preserved.

Any solution information stored in the problem object is lost.

Example

```
status = CPXNETdelset (env, net, whichnodes, whicharcs);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
whichnodes	Array of size at least `CPXNETgetnumnodes` that indicates the nodes to be deleted. If `whichnodes[i] == 1`, the node is deleted. For every node deleted, all arcs incident to it are deleted as well. After termination, `whichnode[j]` indicates either the position to which node with index `j` before deletion has been moved or, -1 if the node has been deleted. If NULL is passed, no nodes are deleted.
whicharcs	Array indicating the arc to be deleted. Every arc `i` in the network with `whicharcs[i] == 1` is deleted. After termination, `whicharc[j]` indicates either the position to which arc with index `j` before deletion has been moved or, -1 if the arc has been deleted. This array also contains the deletions due to removed nodes. If NULL is passed, the only arcs deleted are those that are incident to nodes that have been deleted.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXchgobjsen

`void` **`CPXchgobjsen`**`(CPXCENVptr env, CPXLPptr lp, int maxormin)`

Definition file: cplex.h

The routine `CPXchgobjsen` changes the sense of the optimization for a problem, to maximization or minimization.

> **Note**
>
> For problems with a quadratic objective function, changing the objective sense may make the problem unsolvable. Further changes to the quadratic coefficients may then be required to restore the convexity (concavity) of a minimization (maximization) problem.

Values of maxormin

`CPX_MIN`	(`1`)	new sense is minimize
`CPX_MAX`	(`-1`)	new sense is maximize

Example

```
CPXchgobjsen (env, lp, CPX_MAX);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
maxormin	An integer that specifies the new sense of the problem.

Returns:

This routine does not return a result.

Global function CPXchgprobtype

`int` **`CPXchgprobtype`**`(CPXCENVptr env, CPXLPptr lp, int type)`

Definition file: cplex.h

The routine `CPXchgprobtype` changes the current problem to a related problem. The problem types that can be used appear in the table.

Table 1: Problem Types

Value	Symbolic Constant	Meaning
0	CPXPROB_LP	Linear program, no ctype or quadratic data stored.
1	CPXPROB_MILP	Problem with ctype information.
3	CPXPROB_FIXEDMILP	Problem with ctype information, integer variables fixed.
5	CPXPROB_QP	Problem with quadratic data stored.
7	CPXPROB_MIQP	Problem with quadratic data and ctype information.
8	CPXPROB_FIXEDMIQP	Problem with quadratic data and ctype information, integer variables fixed.
10	CPXPROB_QCP	Problem with quadratic constraints.
11	CPXPROB_MIQCP	Problem with quadratic constraints and ctype information.

A mixed integer problem (CPXPROB_MILP, CPXPROB_MIQP, or CPXPROB_MIQCP) can be changed to a fixed problem (CPXPROB_FIXEDMILP, CPXPROB_FIXEDMIQP), or CPXPROB_FIXEDMIQCP, where bounds on integer variables are fixed to the values attained in the integer solution. A mixed integer problem (or its related fixed type) can also be changed to a continuous problem (CPXPROB_LPCPXPROB_QP, or CPXPROB_QCP), which causes any existing ctype values to be permanently discarded from the problem object.

The original mixed integer problem can be recovered from the fixed problem. If the current problem type is CPXPROB_FIXEDMILP, CPXPROB_FIXEDMIQP, or CPXPROB_FIXEDMIQCP, any calls to problem modification routines fail. To modify the problem object, the problem type should be changed to CPXPROB_MILP, CPXPROB_MIQP, or CPXPROB_MIQCP.

Changing a problem from a continuous type to a mixed integer type causes a ctype array to be created such that all variables are considered continuous. A problem of type CPXPROB_MILP, CPXPROB_MIQP, or CPXPROB_MIQCP can be solved only by the routine CPXmipopt, even if all of its variables are continuous.

A quadratic problem (CPXPROB_QP, CPXPROB_MIQP, CPXPROB_QCP, or CPXPROB_MIQCP) can be changed to a linear program (CPXPROB_LP), causing any existing quadratic information to be permanently discarded from the problem object. Changing a problem from a linear program (CPXPROB_LP or CPXPROB_MILP) to a quadratic program (CPXPROB_QP or CPXPROB_MIQP) causes an empty quadratic matrix to be created such that the problem is quadratic with the matrix `Q = 0`.

Example

```
status = CPXchgprobtype (env, lp, CPXPROB_MILP);
```

See Also: CPXchgprobtypesolnpool

Parameters:

env A pointer to the CPLEX environment as returned by CPXopenCPLEX.

lp A pointer to a CPLEX LP problem object as returned by CPXcreateprob.

type An integer specifying the desired problem type. See the previous discussion for possible values.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXchgmipstarts

```
int CPXchgmipstarts(CPXCENVptr env, CPXLPptr lp, int mcnt, const int *
mipstartindices, int nzcnt, const int * beg, const int * varindices, const double *
values, const int * effortlevel)
```

Definition file: cplex.h

The routine `CPXchgmipstarts` modifies or extends multiple MIP starts. If an existing MIP start has no value for the variable `x[j]`, for example, and the call to `CPXchgmipstarts` specifies a start value, then the specified value is added to the MIP start. If the existing MIP start already has a value for `x[j]`, then the new value replaces the old. If the MIP starts to be changed do not exist, `CPXchgmipstarts` will not create them and will return an error, `CPXERR_INDEX_RANGE`, instead. Start values may be specified for both integer and continuous variables.

See the routine `CPXaddmipstarts` for more information about how CPLEX uses MIP start information.

Example

```
status = CPXchgmipstarts (env, lp, mcnt, mipstartindices, nzcnt, beg, varindices, values, effortlevel);
```

See Also: CPXaddmipstarts

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
mcnt	An integer giving the number of MIP starts to be changed. This specifies the length of the array `mipstartindices`.
mipstartindices	An array of length `mcnt` containing the numeric indices of the MIP starts to be changed.
nzcnt	An integer giving the number of entries to be changed. This specifies the length of the arrays `varindices` and `values`.
beg	An array of length `mcnt` specifying which part of arrays `varindices` and `values` concern which MIP start to be changed. `beg[0]` must be 0 (zero). The elements specific to each MIP start `i` must be stored in sequential locations in arrays `varindices` and `values` from position `beg[i]` to `beg[i+1]-1` (or from `beg[i]` to `nzcnt -1` if `i=mcnt-1`).
varindices	An array of length `nzcnt` containing the numeric indices of the columns corresponding to the variables which are assigned starting values.
values	An array of length `nzcnt` containing the values to use for the MIP starts. The entry `values[j]` is the value assigned to the variable `indices[j]`. An entry `values[j]` greater than or equal to `CPX_INFBOUND` specifies that no value is set for the variable `indices[j]`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetcallbackseqinfo

```
int CPXgetcallbackseqinfo(CPXCENVptr env, void * cbdata, int wherefrom, int seqid,
int whichinfo, void * result_p)
```

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbackseqinfo` accesses information about nodes during the MIP optimization from within user-written callbacks. This routine may be called only when the value of its `wherefrom` argument is `CPX_CALLBACK_MIP_NODE`. The information accessed from this routine can also be accessed with the routine `CPXgetcallbacknodeinfo`. Nodes are not stored by sequence number but by node number, so using the routine `CPXgetcallbackseqinfo` can be much more time-consuming than using the routine `CPXgetcallbacknodeinfo`. A typical use of this routine is to obtain the node number of a node for which the sequence number is known and then use that node number to select the node with the node callback.

Note

This routine cannot retrieve information about nodes that have been moved to node files. (For more information about node files, see the *CPLEX User's Manual*.) If the argument `seqnum` refers to a node in a node file, `CPXgetcallbacknodeinfo` returns the value `CPXERR_NODE_ON_DISK`.

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting where the user-written callback was called from. This argument must be the value of `wherefrom` passed to the user-written callback.
seqid	The sequence number of the node for which information is requested.
whichinfo	An integer specifying which information is requested. For a summary of possible values, refer to the table titled *Information Requested for a User-Written Node Callback* in the description of `CPXgetcallbacknodeinfo`.
result_p	A generic pointer to a variable of type `double` or `int`. The variable represents the value returned by `whichinfo`. The column *C Type* in the table titled *Information Requested for a User-Written Node Callback* shows the type of various values returned by `whichinfo`.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The return value `CPXERR_NODE_ON_DISK` reports an attempt to access a node currently located in a node file on disk.

Global function CPXsetincumbentcallbackfunc

```
int CPXsetincumbentcallbackfunc(CPXENVptr env, int(CPXPUBLIC
*incumbentcallback)(CALLBACK_INCUMBENT_ARGS), void * cbhandle)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXsetincumbentcallbackfunc` sets and modifies the user-written callback routine to be called when an integer solution has been found but before this solution replaces the incumbent. This callback can be used to discard solutions that do not meet criteria beyond that of the mixed integer programming formulation.

Variables are in terms of the original problem if the parameter `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, variables are in terms of the presolved problem.

Example

```
status = CPXsetincumbentcallbackfunc (env, myincumbentcheck,
                                      mydata);
```

See also *Advanced MIP Control Interface* in the *CPLEX User's Manual*.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`incumbentcallback`

A pointer to a user-written incumbent callback. If the callback is set to NULL, no callback can be called during optimization.

`cbhandle`

A pointer to user private data. This pointer is passed to the callback.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle,
              double     objval,
              double     *x,
              int        *isfeas_p,
              int        *useraction_p);
```

The incumbent callback is called when CPLEX has found an integer solution, but before this solution replaces the incumbent integer solution.

Variables are in terms of the original problem if the parameter `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF` before the call to `CPXmipopt` that calls the callback. Otherwise, variables are in terms of the presolved problem.

Callback return value

The callback returns zero if successful and nonzero if an error occurs.

Callback arguments

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`cbdata`

A pointer passed from the optimization routine to the user-written callback that identifies the problem being optimized. The only purpose of this pointer is to pass it to the callback information routines.

`wherefrom`

An integer value reporting where in the optimization this function was called. It will have the value `CPX_CALLBACK_MIP_BRANCH`.

`cbhandle`

A pointer to user private data.

`objval`

A variable that contains the objective value of the integer solution.

`x`

An array that contains primal solution values for the integer solution.

`isfeas_p`

A pointer to an integer variable that determines whether or not CPLEX should use the integer solution specified in `x` to replace the current incumbent. A nonzero value states that the incumbent should be replaced by `x`; a zero value states that it should not.

`useraction_p`

A pointer to an integer to contain the specifier of the action to be taken on completion of the user callback. The table summarizes the possible values.

Actions to be Taken after a User-Written Incumbent Callback

Value	Symbolic Constant	Action
0	CPX_CALLBACK_DEFAULT	Proceed with optimization
1	CPX_CALLBACK_FAIL	Exit optimization
2	CPX_CALLBACK_SET	Proceed with optimization

See Also: CPXgetincumbentcallbackfunc

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcreateprob

```
CPXLPptr CPXcreateprob(CPXCENVptr env, int * status_p, const char * probname_str)
```

Definition file: cplex.h

The routine `CPXcreateprob` creates a CPLEX problem object in the CPLEX environment. The arguments to `CPXcreateprob` define an LP problem name. The problem that is created is an LP minimization problem with zero constraints, zero variables, and an empty constraint matrix. The CPLEX problem object exists until the routine `CPXfreeprob` is called.

To define the constraints, variables, and nonzero entries of the constraint matrix, any of the CPLEX LP problem modification routines may be used. In addition, any of the routines beginning with the prefix `CPXcopy` may be used to copy data into the CPLEX problem object. New constraints or new variables can be created with the routines `CPXnewrows` or `CPXnewcols`, respectively.

Example

```
lp = CPXcreateprob (env, &status, "myprob");
```

See also all the Callable Library examples (except those pertaining to networks) in the *CPLEX User's Manual*.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
status_p	A pointer to an integer used to return any error code produced by this routine.
probname_str	A character string that specifies the name of the problem being created.

Returns:

If successful, `CPXcreateprob` returns a pointer that can be passed to other CPLEX routines to identify the problem object that is created. If not successful, a NULL pointer is returned, and an error status is returned in the variable `*status_p`. If the routine is successful, `*status_p` is 0 (zero).

Global function CPXmbasewrite

```
int CPXmbasewrite(CPXCENVptr env, CPXCLPptr lp, const char * filename_str)
```

Definition file: cplex.h

The routine `CPXmbasewrite` writes the most current basis associated with a CPLEX problem object to a file. The file is saved in BAS format which corresponds to the industry standard MPS insert format for bases.

When `CPXmbasewrite` is invoked, the current basis is written to a file. This routine does not remove the basis from the problem object.

Example

```
status = CPXmbasewrite (env, lp, "myprob.bas");
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to the CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the basis should be written.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXaddsolnpoolrngfilter

```
int CPXaddsolnpoolrngfilter(CPXCENVptr env, CPXLPptr lp, double lb, double ub, int
nzcnt, const int * ind, const double * val, const char * lname_str)
```

Definition file: cplex.h

Adds a new range filter to the solution pool.

A *range filter* drives the search for multiple solutions toward new solutions that satisfy criteria specified as a ranged linear expression in the filter. A range filter sets a lower and an upper bound on a linear expression consisting of `nzcnt` variables designated by their indices in the argument `ind` and coefficient values designated in the argument `val`.

```
lower bound <= sum{val[i] times x[ind[i]]}  <= upper bound
```

A range filter applies to variables of any type (that is, binary, general integer, continuous).

Example

```
status = CPXaddsolnpoolrngfilter (env, lp, loval, hival,
                                  cnt, ind, val, NULL);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lb	The lower bound on the linear expression.
ub	The upper bound on the linear expression.
nzcnt	The number of variables in the linear expression.
ind	An array of variable indices that with `val` defines the linear expression.
val	An array of values that with `ind` defines the linear expression. The nonzero coefficients of the linear terms must be stored in sequential locations in the arrays `ind` and `val` from positions `0` to `num-1`. Each entry, `ind[i]`, specifies the variable index of the corresponding coefficient, `val[i]`.
lname_str	The name of the filter. May be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXqpdjfrompi

```
int CPXqpdjfrompi(CPXCENVptr env, CPXCLPptr lp, const double * pi, const double *
x, double * dj)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXqpdjfrompi` computes an array of reduced costs from an array of dual values and an array of primal values for a QP.

Example

```
status = CPXqpdjfrompi (env, lp, origpi, reducepi);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
pi	An array that contains dual solution (`pi`) values for a problem, as returned by such routines as `CPXqpuncrushpi` and `CPXcrushpi`. The length of the array must at least equal the number of rows in the LP problem object.
x	An array that contains primal solution (`x`) values for a problem, as returned by such routines as `CPXuncrushx` and `CPXcrushx`. The length of the array must at least equal the number of columns in the LP problem object.
dj	An array to receive the reduced cost values computed from the `pi` values for the problem object. The length of the array must at least equal the number of columns in the problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXrefineconflict

`int` **`CPXrefineconflict`**`(CPXCENVptr env, CPXLPptr lp, int * confnumrows_p, int * confnumcols_p)`

Definition file: cplex.h

The routine `CPXrefineconflict` identifies a minimal conflict for the infeasibility of the linear constraints and the variable bounds in the current problem. Since the conflict returned by this routine is minimal, removal of any member constraint or variable bound will remove that particular source of infeasibility. There may be other conflicts in the problem, so that repair of a conflict does not guarantee feasibility of the remaining problem.

To find a conflict by considering the quadratic constraints, indicator constraints, or special ordered sets, as well as the linear constraints and variable bounds, use `CPXrefineconflictext`.

When this routine returns, the value in `confnumrows_p` specifies the number of constraints participating in the conflict, and the value in `confnumcols_p` specifies the number of variables participating in the conflict. Use the routine `CPXgetconflict` to determine which constraints and variables participate in the conflict.

The parameters `CPX_PARAM_CUTUP`, `CPX_PARAM_CUTLO`, `CPX_PARAM_OBJULIM`, `CPX_PARAM_OBJLLIM` do **not** influence this routine. If you want to study infeasibilities introduced by those parameters, consider adding an objective function constraint to your model to enforce their effect before you invoke this routine.

Example

```
status = CPXrefineconflict (env, lp, NULL, NULL);
```

See Also: CPXgetconflict, CPXrefineconflictext, CPXclpwrite

Parameters:

env	A pointer to the CPLEX environment as returned by the routine `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
confnumrows_p	A pointer to an integer where the number of linear constraints in the conflict is returned.
confnumcols_p	A pointer to an integer where the number of variable bounds in the conflict is returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcheckvals

```
int CPXcheckvals(CPXCENVptr env, CPXCLPptr lp, int cnt, const int * rowind, const
int * colind, const double * values)
```

Definition file: cplex.h

The routine `CPXcheckvals` checks an array of indices and a corresponding array of values for input errors. The routine is useful for validating the arguments of problem modification routines such as `CPXchgcoeflist`, `CPXchgbds`, `CPXchgobj`, and `CPXchgrhs`. This data checking routine is found in source format in the file `check.c` which is provided with the standard CPLEX distribution. To call this routine, you must compile and link `check.c` with your program as well as the CPLEX Callable Library.

Example

Consider the following call to `CPXchgobj`:

```
status = CPXchgobj (env, lp, cnt, indices, values);
```

The arguments to this routine can be checked with a call to `CPXcheckvals` like this:

```
status = CPXcheckvals (env, lp, cnt, NULL, indices, values);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
cnt	The length of the indices and values arrays to be examined.
rowind	An array containing row indices. May be NULL.
colind	An array containing column indices. May be NULL.
values	An array of values. May be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetindconstrslack

`int` **`CPXgetindconstrslack`**`(CPXCENVptr env, CPXCLPptr lp, double * indslack, int begin, int end)`

Definition file: cplex.h

The routine `CPXgetindconstrslack` accesses the slack values for a range of indicator constraints. The beginning and end of the range must be specified. Note that an indicator constraint is considered inactive, and thus returns an infinite slack value, when the corresponding indicator binary takes a value less than the integrality tolerance (or greater than 1 minus the integrality tolerance if the indicator binary is complemented).

Example

```
status = CPXgetindconstrslack (env, lp, indslack, 0, CPXgetnumindconstrs(env,lp)-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
indslack	An array to receive the slack values for each of the constraints. This array must be of length at least (`end - begin + 1`). If successful, `indslack[0]` through `indslack[end-begin]` contain the values of the slacks.
begin	An integer specifying the beginning of the range of slack values to be returned.
end	An integer specifying the end of the range of slack values to be returned.

Returns:

The routine returns 0 (zero) if successful and nonzero if an error occurs.

Global function CPXgetsosname

```
int CPXgetsosname(CPXCENVptr env, CPXCLPptr lp, char ** name, char * namestore, int
storespace, int * surplus_p, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetsosname` accesses a range of special ordered set (SOS) names of a CPLEX problem object. The beginning and end of the range, along with the length of the array in which the SOS names are to be returned, must be specified.

Note

If the value of `storespace` is 0 (zero), then the negative of the value of `*surplus_p` returned specifies the total number of characters needed for the array `namestore`.

Example

```
status = CPXgetsosname (env, lp, cur_sosname, cur_sosnamestore,
                        cur_storespace, &surplus, 0,
                        cur_numsos-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
name	An array of pointers to the SOS names stored in the array `namestore`. This array must be of length at least (`end - begin+ 1`). The pointer to the name of SOS `i` is returned in `name[i-begin]`.
namestore	An array of characters where the requested SOS names are to be returned. May be NULL if `storespace` is 0 (zero).
storespace	An integer specifying the length of the array `namestore`. May be 0 (zero).
surplus_p	A pointer to an integer to contain the difference between `storespace` and the total amount of memory required to store the requested names. A nonnegative value of `*surplus_p` specifies that `storespace` was sufficient. A negative value reports that it was insufficient and that the routine could not complete its task. In that case, `CPXgetsosname` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `*surplus_p` specifies the amount of insufficient space in the array `namestore`.
begin	An integer specifying the beginning of the range of sos names to be returned.
end	An integer specifying the end of the range of sos names to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` reports that insufficient space was available in the `namestore` array to hold the names.

Global function CPXpresolve

`int` **`CPXpresolve`**`(CPXCENVptr env, CPXLPptr lp, int method)`

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXpresolve` performs LP or MIP presolve depending whether a problem object is an LP or a MIP. If the problem is already presolved, the existing presolved problem is freed, and a new presolved problem is created.

Example

```
status = CPXpresolve (env, lp, CPX_ALG_DUAL);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
method	An integer specifying the optimization algorithm to be used to solve the problem after the presolve is completed. Some presolve reductions are specific to an optimization algorithm, so specifying the algorithm makes sure that the problem is presolved for that algorithm, and that presolve does not have to be repeated when that optimization routine is called. Possible values are `CPX_ALG_NONE`, `CPX_ALG_PRIMAL`, `CPX_ALG_DUAL`, and `CPX_ALG_BARRIER` for LP; `CPX_ALG_NONE` should be used for MIP.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXreadcopysolnpoolfilters

```
int CPXreadcopysolnpoolfilters(CPXCENVptr env, CPXLPptr lp, const char *
filename_str)
```

Definition file: cplex.h

The routine `CPXreadcopysolnpoolfilters` reads solution pool filters from an FLT format file and copies the filters into a CPLEX problem object. This operation replaces all existing filters previously associated with the CPLEX problem object. This format is documented in the *CPLEX File Formats Reference Manual*.

Example

```
status = CPXreadcopysolnpoolfilters (env, lp, "myfilters.flt");
```

See Also: CPXfltwrite

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	The name of the file from which the filters should be read.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcrushpi

```
int CPXcrushpi(CPXCENVptr env, CPXCLPptr lp, const double * pi, double * prepi)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXcrushpi` crushes a dual solution for the original problem to a dual solution for the presolved problem.

Example

```
status = CPXcrushpi (env, lp, origpi, reducepi);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
pi	An array that contains dual solution (`pi`) values for the original problem, as returned by routines such as `CPXgetpi` or `CPXsolution`. The array must be of length at least the number of rows in the LP problem object.
prepi	An array to receive dual values corresponding to the presolved problem. The array must be of length at least the number of rows in the presolved problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetqconstr

```
int CPXgetqconstr(CPXCENVptr env, CPXCLPptr lp, int * linnzcnt_p, int *
quadnzcnt_p, double * rhs_p, char * sense_p, int * linind, double * linval, int
linspace, int * linsurplus_p, int * quadrow, int * quadcol, double * quadval, int
quadspace, int * quadsurplus_p, int which)
```

Definition file: cplex.h

The routine `CPXgetqconstr` is used to access a specified quadratic constraint on the variables of a CPLEX problem object. The length of the arrays in which the nonzero linear and quadratic coefficients of the constraint are to be returned must be specified.

Note

If the value of `linspace` is 0 (zero), then the negative of the value of `*linsurplus_p` returned indicates the length needed for the arrays `linind` and `linval`.

Note

If the value of `quadspace` is 0 (zero), then the negative of the value of `*quadsurplus_p` returned indicates the length needed for the arrays `quadrow, quadcol` and `quadval`.

Example

```
status = CPXgetqconstr (env, lp, &linnzcnt, &quadnzcnt,
                        &rhs, &sense, linind, linval,
                        linspace, &linsurplus, quadrow, quadcol, quadval,
                        quadspace, &quadsurplus, 0);
```

Parameters:

env	A pointer to the CPLEX environment as returned by the `CPXopenCPLEX` routine.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
linnzcnt_p	A pointer to an integer to contain the number of linear coefficients returned; that is, the true length of the arrays `linind` and `linval`.
quadnzcnt_p	A pointer to an integer to contain the number of quadratic coefficients returned; that is, the true length of the arrays `quadrow, quadcol` and `quadval`.
rhs_p	A pointer to a `double` containing the righthand-side value of the quadratic constraint.
sense_p	A pointer to a character indicating the sense of the constraint. Possible values are L for a <= constraint or G for a >= constraint.
linind	An array to contain the variable indices of the entries of `linval`. May be NULL if `linspace` is 0.
linval	An array to contain the linear coefficients of the specified constraint. May be NULL if `linspace` is 0.
linspace	An integer indicating the length of the arrays `linind` and `linval`. May be 0.
linsurplus_p	A pointer to an integer to contain the difference between `linspace` and the number of entries in each of the arrays `linind` and `linval`. A nonnegative value of `*linsurplus_p` indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, the routine `CPXgetqconstr` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of `*linsurplus_p` indicates the amount of insufficient space in the arrays. May be NULL if `linspace` is 0.
quadrow	An array to contain the variable indices of the entries of `quadval`. If the quadratic coefficients were stored in a matrix, `quadrow` would give the row indexes of the quadratic terms. May be NULL if `quadspace` is 0.
quadcol	An array to contain the variable indices of the entries of `quadval`. If the quadratic coefficients were stored in a matrix, `quadcol` would give the column indexes of the quadratic terms. May be

	NULL if `quadspace` is 0.
quadval	An array to contain the quadratic coefficients of the specified constraint. May be NULL if `quadspace` is 0.
quadspace	An integer indicating the length of the arrays `quadrow, quadcol` and `quadval`. May be 0.
quadsurplus_p	A pointer to an integer to contain the difference between `quadspace` and the number of entries in each of the arrays `quadrow, quadcol` and `quadval`. A nonnegative value of `*quadsurplus_p` indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, the routine `CPXgetqconstr` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of `*quadsurplus_p` indicates the amount of insufficient space in the arrays. May be NULL if `quadspace` is 0.
which	An integer indicating which quadratic constraint to return.

Returns:

The routine returns zero on success and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` indicates that insufficient space was available in either the arrays `linind` and `linval` or `quadrow, quadcol`, and `quadval` to hold the nonzero coefficients.

Global function CPXgetrows

```
int CPXgetrows(CPXCENVptr env, CPXCLPptr lp, int * nzcnt_p, int * rmatbeg, int *
rmatind, double * rmatval, int rmatspace, int * surplus_p, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetrows` accesses a range of rows of the constraint matrix, not including the objective function nor the bound constraints on the variables of a CPLEX problem object. The beginning and end of the range, along with the length of the arrays in which the nonzero entries of these rows are to be returned, must be specified.

Note

If the value of `rmatspace` is 0 then the negative of the value of `surplus_p` returned specifies the length needed for the arrays `rmatval` and `rmatind`.

Example

```
status = CPXgetrows (env, lp, &nzcnt, rmatbeg, rmatind, rmatval,
                     rmatspace, &surplus, 0, cur_numrows-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by the `CPXopenCPLEX` routine.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
nzcnt_p	A pointer to an integer to contain the number of nonzeros returned; that is, the true length of the arrays `rmatind` and `rmatval`.
rmatbeg	An array to contain indices specifying where each of the requested rows begins in the arrays `rmatval` and `rmatind`. Specifically, row i consists of the entries in `rmatval` and `rmatind` in the range from `rmatbeg[i - begin]` to `rmatbeg[(i + 1)-begin]-1`. (Row `end` consists of the entries from `rmatbeg[end - begin] to *nzcnt_p-1`.) This array must be of length at least (`end - begin + 1`).
rmatind	An array to contain the column indices of the entries of `rmatval`. May be NULL if `rmatspace` is 0.
rmatval	An array to contain the nonzero entries of the specified rows. May be NULL if `rmatspace` is 0.
rmatspace	An integer specifying the length of the arrays `rmatind` and `rmatval`. May be 0.
surplus_p	A pointer to an integer to contain the difference between `rmatspace` and the number of entries in each of the arrays `rmatind` and `rmatval`. A nonnegative value of `surplus_p` specifies that the length of the arrays was sufficient. A negative value specifies that the length was insufficient and that the routine could not complete its task. In this case, the routine `CPXgetrows` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of `surplus_p` specifies the amount of insufficient space in the arrays.
begin	An integer specifying the beginning of the range of rows to be returned.
end	An integer specifying the end of the range of rows to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` specifies that insufficient space was available in the arrays `rmatind` and `rmatval` to hold the nonzero coefficients.

Global function CPXstrcpy

`CPXCHARptr` **`CPXstrcpy`**`(char * dest_str, const char * src_str)`

Definition file: cplex.h

The routine `CPXstrcpy` copies strings. It is exactly the same as the standard C library routine `strcpy`. This routine is provided so that strings passed to the message function routines (see `CPXaddfuncdest`) can be copied by languages that do not allow dereferencing of pointers (for example, older versions of Visual Basic).

Example

```
CPXstrcpy (p, q);
```

Parameters:

dest_str	A pointer to the string to hold the copy of the string pointed to by `src_str`.
src_str	A pointer to a string to be copied to `dest_str`.

Returns:

The routine returns a pointer to the string being copied to.

Global function CPXgetindconstrindex

```
int CPXgetindconstrindex(CPXCENVptr env, CPXCLPptr lp, const char * lname_str, int
* index_p)
```

Definition file: cplex.h

The routine `CPXgetindconstrindex` searches for the index number of the specified indicator constraint in a CPLEX problem object.

Example

```
status = CPXgetindconstrindex (env, lp, "resource89", &indconstrindex);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lname_str	Name of an indicator constraint to search for.
index_p	A pointer to an integer to hold the index number of the indicator constraint with the name `lname_str`. If the routine is successful, `*index_p` contains the index number; otherwise, `*index_p` is undefined.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXaddqconstr

```
int CPXaddqconstr(CPXCENVptr env, CPXLPptr lp, int linnzcnt, int quadnzcnt, double
rhs, int sense, const int * linind, const double * linval, const int * quadrow,
const int * quadcol, const double * quadval, const char * lname_str)
```

Definition file: cplex.h

The routine `CPXaddqconstr` adds a quadratic constraint to a specified CPLEX problem object. This routine may be called any time after a call to `CPXcreateprob`.

Codes for sense of constraints in QCPs

`sense[i]`	`= 'L'`	<= constraint
`sense[i]`	`= 'G'`	>= constraint

Example

```
status = CPXaddqconstr (env, lp, linnzcnt, quadnzcnt, rhsval,
                        sense, linind, linval,
                        quadrow, quadcol, quadval, NULL);
```

See also the example `qcpex1.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
linnzcnt	An integer that indicates the number of nonzero constraint coefficients in the linear part of the constraint. This specifies the length of the arrays `linind` and `linval`.
quadnzcnt	An integer that indicates the number of nonzero constraint coefficients in the quadratic part of the constraint. This specifies the length of the arrays `quadrow, quadcol` and `quadval`.
rhs	The righthand side term for the constraint to be added.
sense	The sense of the constraint to be added. Note that quadratic constraints may only be less-than-or-equal-to or greater-than-or-equal-to constraints. See the discussion of QCP in the *CPLEX User's Manual*.
linind	An array that with `linval` defines the linear part of the quadratic constraint to be added.
linval	An array that with `linind` defines the linear part of the constraint to be added. The nonzero coefficients of the linear terms must be stored in sequential locations in the arrays `linind` and `linval` from positions `0` to `linnzcnt-1`. Each entry, `linind[i]`, indicates the variable index of the corresponding coefficient, `linval[i]`. May be NULL; then the constraint will have no linear terms.
quadrow	An array that with `quadcol` and `quadval` defines the quadratic part of the quadratic constraint to be added.
quadcol	An array that with `quadrow` and `quadval` defines the quadratic part of the quadratic constraint to be added.
quadval	An array that with `quadrow` and `quadcol` define the quadratic part of the constraint to be added. The nonzero coefficients of the quadratic terms must be stored in sequential locations in the arrays `quadrow, quadcol` and `quadval` from positions `0` to `quadnzcnt-1`. Each pair, `quadrow[i], quadcol[i]`, indicates the variable indices of the quadratic term, and `quadval[i]` the corresponding coefficient.
lname_str	The name of the constraint to be added. May be NULL, in which case the new constraint is assigned a default name if the quadratic constraints already resident in the CPLEX problem object have names; otherwise, no name is associated with the constraint.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetcallbacknodelb

`int` **`CPXgetcallbacknodelb`**`(CPXCENVptr env, void * cbdata, int wherefrom, double * lb, int begin, int end)`

Definition file: cplex.h

Note

This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetcallbacknodelb` retrieves the lower bound values for the subproblem at the current node during MIP optimization from within a user-written callback. The lower bounds are tightened after a new incumbent is found, so the values returned by `CPXgetcallbacknodex` may violate these bounds at nodes where new incumbents have been found. The values are from the original problem if `CPX_PARAM_MIPCBREDLP` is set to `CPX_OFF`; otherwise, they are from the presolved problem.

This routine may be called only when the value of the `wherefrom` argument is one of the following:

- `CPX_CALLBACK_MIP`,
- `CPX_CALLBACK_MIP_BRANCH`,
- `CPX_CALLBACK_MIP_INCUMBENT`,
- `CPX_CALLBACK_MIP_NODE`,
- `CPX_CALLBACK_MIP_HEURISTIC`,
- `CPX_CALLBACK_MIP_SOLVE`, or
- `CPX_CALLBACK_MIP_CUT`.

Example

```
status = CPXgetcallbacknodelb (env, cbdata, wherefrom,
                               lb, 0, cols-1);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cbdata	The pointer passed to the user-written callback. This argument must be the value of `cbdata` passed to the user-written callback.
wherefrom	An integer value reporting from where the user-written callback was called. The argument must be the value of `wherefrom` passed to the user-written callback.
lb	An array to receive the values of the lower bound values. This array must be of length at least (`end - begin + 1`). If successful, `lb[0]` through `lb[end-begin]` contain the lower bound values for the current subproblem.
begin	An integer specifying the beginning of the range of lower bounds to be returned.
end	An integer specifying the end of the range of lower bounds to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXsetnetcallbackfunc

```
int CPXsetnetcallbackfunc(CPXENVptr env, int(CPXPUBLIC *callback)(CPXCENVptr, void
*, int, void *), void * cbhandle)
```

Definition file: cplex.h

The routine `CPXsetnetcallbackfunc` sets the user-written callback routine to be called each time a log message is issued during the optimization of a network program. If the display log is turned off, the callback routine will still be called.

This routine works in the same way as the routine `CPXsetlpcallbackfunc`. It enables the user to create a separate callback function to be called during the solution of a network problem. The prototype for the callback function is identical to that of `CPXsetlpcallbackfunc`.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle);
```

This is the user-written callback routine.

Callback return value

A nonzero terminates the optimization.

Callback arguments

`env`

A pointer to the CPLEX environment that was passed into the associated optimization routine.

`cbdata`

A pointer passed from the optimization routine to the user-written callback function that identifies the problem being optimized. The only purpose for the `cbdata` pointer is to pass it to the routine `CPXgetcallbackinfo`.

`wherefrom`

An integer value specifying from which optimization algorithm the user-written callback function was called. Possible values and their meaning appear in the table.

Value	Symbolic Constant	Meaning
3	CPX_CALLBACK_NETWORK	From network simplex

`cbhandle`

Pointer to user private data, as passed to `CPXsetnetcallbackfunc`.

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`callback`

A pointer to a user-written callback function. Setting `callback` to NULL prevents any callback function from being called during optimization. The call to `callback` occurs after every log message is issued during optimization and periodically during the CPLEX presolve algorithms. This function is written by the user.

`cbhandle`

A pointer to user private data. This pointer is passed to the callback function.

Example

```
status = CPXsetnetcallbackfunc (env, myfunc, NULL);
```

See Also: CPXgetcallbackinfo, CPXsetlpcallbackfunc, CPXsetmipcallbackfunc

Returns:

If the operation is successful, the routine returns zero; if not, it returns nonzero to report an error.

Global function CPXgetquad

```
int CPXgetquad(CPXCENVptr env, CPXCLPptr lp, int * nzcnt_p, int * qmatbeg, int *
qmatind, double * qmatval, int qmatspace, int * surplus_p, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetquad` is used to access a range of columns of the matrix Q of a model with a quadratic objective function. The beginning and end of the range, along with the length of the arrays in which the nonzero entries of these columns are to be returned, must be specified.

Specifically, column `j` consists of the entries in `qmatval` and `qmatind` in the range from `qmatbeg[j - begin]` to `qmatbeg[(j + 1) - begin]-1`. (Column `end` consists of the entries from `qmatbeg[end - begin]` to `nzcnt_p-1`.) This array must be of length at least (`end - begin + 1`).

Note

If the value of `qmatspace` is zero, the negative of the value of `surplus_p` returned indicates the length needed for the arrays `qmatind` and `qmatval`.

Example

```
status = CPXgetquad (env, lp, &nzcnt, qmatbeg, qmatind,
                     qmatval, qmatspace, &surplus, 0,
                     cur_numquad-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
nzcnt_p	A pointer to an integer to contain the number of nonzeros returned; that is, the true length of the arrays `qmatind` and `qmatval`.
qmatbeg	An array to contain indices indicating where each of the requested columns of Q begins in the arrays `qmatval` and `qmatind`.
qmatind	An array to contain the row indices associated with the elements of `qmatval`. May be NULL if `qmatspace` is zero.
qmatval	An array to contain the nonzero coefficients of the specified columns. May be NULL if `qmatspace` is zero.
qmatspace	An integer indicating the length of the arrays `qmatind` and `qmatval`. May be zero.
surplus_p	A pointer to an integer to contain the difference between `qmatspace` and the number of entries in each of the arrays `qmatind` and `qmatval`. A nonnegative value of `*surplus_p` indicates that the length of the arrays was sufficient. A negative value indicates that the length was insufficient and that the routine could not complete its task. In this case, `CPXgetquad` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of `*surplus_p` indicates the amount of insufficient space in the arrays.
begin	An integer indicating the beginning of the range of columns to be returned.
end	An integer indicating the end of the range of columns to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` indicates that insufficient space was available in the arrays `qmatind` and `qmatval` to hold the nonzero coefficients.

Global function CPXNETgetobjval

`int` **`CPXNETgetobjval`**`(CPXCENVptr env, CPXCNETptr net, double * objval_p)`

Definition file: cplex.h

The routine `CPXNETgetobjval` returns the objective value of the solution stored in a network problem object.

If the current solution is not feasible, the value returned depends on the setting of the parameter `CPX_PARAM_NETDISPLAY`. If this parameter is set to `CPXNET_PENALIZED_OBJECTIVE (2)`, an objective function value is reported that includes penalty contributions for arcs on which the flow at termination violated the flow bounds on that arc.

Example

```
status = CPXNETgetobjval (env, net, &objval);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
objval_p	Pointer to where the objective value is written. If NULL is passed, no objective value is returned.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXstrongbranch

```
int CPXstrongbranch(CPXCENVptr env, CPXLPptr lp, const int * indices, int cnt,
double * downobj, double * upobj, int itlim)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXstrongbranch` computes information for selecting a branching variable in an integer-programming branch-and-cut search.

To describe this routine, let's assume that an LP has been solved and that the optimal solution is resident. Let `indices[]` be the list of variable indices for this problem and `cnt` be the length of that list. Then `indices[]` gives rise to `2*cnt` different LPs in which each of the listed variables in turn is fixed to the greatest integer value less than or equal to its value in the current optimal solution, and then each variable is fixed to the least integer value greater than or equal to its value in the current optimal solution. `CPXstrongbranch` performs at most `itlim` dual steepest-edge iterations on each of these `2*cnt` LPs, starting from the current optimal solution of the base LP. The objective values that these iterations yield are placed in the arrays `downobj[]` for the downward fix and `upobj[]` for the upward fix. If either of the fixings results in a problem which is dual unbounded (primal infeasible), the corresponding objective value is set to a large positive value for a minimization problem and a large negative value for a maximization problem. This value is system dependent, but it is usually of magnitude *1.0e+75*. Setting `CPX_PARAM_DPRIIND` to 2 may give more informative values for the arguments `downobj[]` and `upobj[]` for a given number of iterations `itlim`.

A user might use other routines of the Callable Library directly to build a function that computes the same values as `CPXstrongbranch`. However, `CPXstrongbranch` should be faster because it takes advantage of direct access to internal CPLEX data structures.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
indices	An array of integers. The length of the array must be at least `cnt`. As in other Callable Library routines, row variables in `indices[]` are specified by the negative of row index shifted down by one; that is, `-rowindex -1`.
cnt	An integer specifying the number of entries in `indices[]`.
downobj	An array containing objective values that are the result of the downward fix of branching variables in dual steepest-edge iterations carried out by `CPXstrongbranch`. The length of the array must be at least `cnt`.
upobj	An array containing objective values that are the result of the upward fix of branching variables in dual steepest-edge iterations carried out by `CPXstrongbranch`. The length of the array must be at least `cnt`.
itlim	An integer specifying the limit on the number of dual steepest-edge iterations carried out by `CPXstrongbranch` on each LP.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXaddindconstr

```
int CPXaddindconstr(CPXCENVptr env, CPXLPptr lp, int indvar, int complemented, int
nzcnt, double rhs, int sense, const int * linind, const double * linval, const char
* indname_str)
```

Definition file: cplex.h

The routine `CPXaddindconstr` adds an indicator constraint to the specified problem object. This routine may be called any time after a call to `CPXcreateprob`.

An indicator constraint is a linear constraint that is enforced only:

- when an associated binary variable takes a value of 1, or
- when an associated binary variable takes the value of 0 (zero) if the binary variable is complemented.

The linear constraint may be a less-than-or-equal-to constraint, a greater-than-or-equal-to constraint, or an equality constraint.

Codes for the sense of a linear constraint

`sense`	`= 'L'`	<= constraint
`sense`	`= 'G'`	>= constraint
`sense`	`= 'E'`	== constraint

Example

```
status = CPXaddindconstr (env, lp, indicator, complemented, nzcnt,
                          rhs, sense, ind, val, newindname);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
indvar	The binary variable that acts as the indicator for this constraint.
complemented	A Boolean value that specifies whether the indicator variable is complemented. The linear constraint must be satisfied when the indicator takes a value of 1 (one) if the indicator is not complemented, and similarly, the linear constraint must be satisfied when the indicator takes a value of 0 (zero) if the indicator is complemented.
nzcnt	An integer that specifies the number of nonzero coefficients in the linear portion of the indicator constraint. This argument gives the length of the arrays `linind` and `linval`.
rhs	The righthand side value for the linear portion of the indicator constraint.
sense	The sense of the linear portion of the indicator constraint. Specify 'L' for <= or 'G' for >= or 'E' for ==.
linind	An array that with `linval` defines the linear portion of the indicator constraint.
linval	An array that with `linind` defines the linear portion of the indicator constraint. The nonzero coefficients of the linear terms must be stored in sequential locations in the arrays `linind` and `linval` from positions `0` to `nzcnt-1`. Each entry, `linind[i]`, indicates the variable index of the corresponding coefficient, `linval[i]`.
indname_str	The name of the constraint to be added. May be NULL, in which case the new constraint is assigned a default name if the indicator constraints already resident in the CPLEX problem object have names; otherwise, no name is associated with the constraint.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXdualopt

`int` **`CPXdualopt`**`(CPXCENVptr env, CPXLPptr lp)`

Definition file: cplex.h

The routine `CPXdualopt` may be used at any time after a linear program has been created via a call to `CPXcreateprob` to find a solution to that problem using the dual simplex algorithm. When this function is called, the CPLEX dual simplex optimization routines attempt to optimize the specified problem. The results of the optimization are recorded in the CPLEX problem object.

Example

```
status = CPXdualopt (env, lp);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Returns:

The routine returns zero unless an error occurred during the optimization. Examples of errors include exhausting available memory (`CPXERR_NO_MEMORY`) or encountering invalid data in the CPLEX problem object (`CPXERR_NO_PROBLEM`).

Exceeding a user-specified CPLEX limit is not considered an error. Proving the problem infeasible or unbounded is not considered an error.

A zero return value does not necessarily mean that a solution exists. Use query routines `CPXsolninfo`, `CPXgetstat`, and `CPXsolution` to obtain further information about the status of the optimization.

Global function CPXNETgetsupply

`int` **`CPXNETgetsupply`**`(CPXCENVptr env, CPXCNETptr net, double * supply, int begin, int end)`

Definition file: cplex.h

The routine `CPXNETgetsupply` is used to obtain supply values for a range of nodes in the network stored in a CPLEX network problem object.

Example

```
status = CPXNETgetsupply (env, net, supply,
                          0, CPXNETgetnumnodes (env, net) - 1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
supply	Place where requested supply values are copied. If NULL is passed, no supply values are copied. Otherwise, the array must be of length at least (`end-begin+1`).
begin	Index of the first node for which a supply value is to be obtained.
end	Index of the last node for which a supply value is to be obtained.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXbasicpresolve

```
int CPXbasicpresolve(CPXCENVptr env, CPXLPptr lp, double * redlb, double * redub,
int * rstat)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXbasicpresolve` performs bound strengthening and detects redundant rows. `CPXbasicpresolve` does not create a presolved problem.

This routine cannot be used for quadratic programs.

Values for `rstat[i]`:

0 if row `i` is not redundant

-1 if row `i` is redundant

In the case of a semicontinous variable or a semi-integer variable, this routine produces the lower bound, **not** the semicontinous bound, **not** the semi-integer lower bound. If the strengthened bound is a value less than or equal to zero, the semicontinuity or semi-integrality persists. In contrast, if the strengthened bound is a value strictly greater than zero, then this routine has concluded that zero can be eliminated from the domain of the variable. It is thus possible to change the type of a semicontinuous variable to continous, or to change the type of a semi-integer variable to integer, without affecting the feasible region of the model.

Example

```
status = CPXbasicpresolve (env, lp, reducelb, reduceub, rowstat);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
redlb	An array to receive the strengthened lower bounds. The array must be of length at least the number of columns in the LP problem object. May be NULL.
redub	An array to receive the strengthened upper bounds. The array must be of length at least the number of columns in the LP problem object. May be NULL.
rstat	An array to receive the status of the row. The array must be of length at least the number of rows in the LP problem object. May be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXchgname

```
int CPXchgname(CPXCENVptr env, CPXLPptr lp, int key, int ij, const char *
newname_str)
```

Definition file: cplex.h

The routine `CPXchgname` changes the name of a constraint or the name of a variable in a CPLEX problem object. If this routine is performed on a problem object with no row or column names, default names are created before the change is made.

Example

```
status = CPXchgname (env, lp, 'c', 10, "name10");
```

Values of key

key = `'r'`	change row name
key = `'c'`	change column name

See Also: CPXdelnames

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
key	A character to specify whether a row name or a column name should be changed. Possible values appear in the table.
ij	An integer that specifies the numeric index of the column or row whose name is to be changed.
newname_str	A pointer to a character string containing the new name.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXpperwrite

```
int CPXpperwrite(CPXCENVptr env, CPXLPptr lp, const char * filename_str, double
epsilon)
```

Definition file: cplex.h

When solving degenerate linear programs with the primal simplex method, CPLEX may initiate a perturbation of the bounds of the problem in order to improve performance. The routine `CPXpperwrite` writes a similarly perturbed problem to a binary SAV format file.

Example

```
status = CPXpperwrite (env, lp, "myprob.ppe", epsilon);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	A character string containing the name of the file to which the perturbed problem should be written.
epsilon	The perturbation constant.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcopysos

```
int CPXcopysos(CPXCENVptr env, CPXLPptr lp, int numsos, int numsosnz, const char *
sostype, const int * sosbeg, const int * sosind, const double * soswt, char **
sosname)
```

Definition file: cplex.h

The routine `CPXcopysos` copies special ordered set (SOS) information to a problem object of type `CPXPROB_MILP`, `CPXPROB_MIQP`, or `CPXPROB_MIQCP`.

When you build or modify your problem with this routine, you can verify that the results are as you intended by calling `CPXcheckcopysos` during application development.

Table 1: Settings for sostype

`CPX_TYPE_SOS1`	'1'	Type 1
`CPX_TYPE_SOS2`	'2'	Type 2

Example

```
status = CPXcopysos (env,
                     lp,
                     numsos,
                     numsosnz,
                     sostype,
                     sosbeg,
                     sosind,
                     soswt,
                     NULL);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
numsos	The number of SOS sets. If `numsos` is equal to zero, `CPXcopysos` removes all the SOSs from the LP object.
numsosnz	The total number of members in all sets. `CPXcopysos` with `numsosnz` equal to zero removes all the SOSs from the LP object.
sostype	An array containing SOS type information for the sets. `sostype[i]` specifies the SOS type of set `i`, according tot the settings in Table 1. The length of this array must be at least `numsos`.
sosbeg	An array stating beginning indices as explained in `soswt`.
sosind	An array stating indices as explained in `soswt`.
soswt	Arrays declaring the indices and weights for the sets. For every set, the indices and weights must be stored in sequential locations in `sosind` and `soswt`, respectively, with `sosbeg[j]` containing the index of the beginning of set `j`. The weights must be unique in their array. For `j < numsos-1` the indices of set `j` must be stored in `sosind[sosbeg[j]],..., sosind[sosbeg[j+1]-1]` and the weights in `soswt[sosbeg[j],..., soswt[sosbeg[j+1]-1]`. For the last set, `j = numsos-1`, the indices must be stored in `sosind[sosbeg[numsos-1]],...,` `sosind[numsosnz-1]` and the corresponding weights in `soswt[sosbeg[numsos-1]] ...,` `soswt[numsosnz-1]`. Hence, `sosbeg` must be of length at least `numsos`, while `sosind` and `soswt` must be of length at least `numsosnz`.
sosname	An array containing pointers to character strings that represent the names of the SOSs. May be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetnetcallbackfunc

```
int CPXgetnetcallbackfunc(CPXCENVptr env, int(CPXPUBLIC **callback_p)(CPXCENVptr,
void *, int, void *), void ** cbhandle_p)
```

Definition file: cplex.h

The `CPXgetnetcallbackfunc` accesses the user-written callback routine to be called each time a log message is issued during the optimization of a network problem. If the display log is turned off, the callback routine is still called.

This routine works in the same way as the routine `CPXgetlpcallbackfunc`. It enables the user to create a separate callback function to be called during the solution of a network problem. The prototype for the callback function is identical to that of `CPXgetlpcallbackfunc`.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle);
```

This is the user-written callback routine.

Callback return value

A nonzero terminates the optimization.

Callback arguments

`env`

A pointer to the CPLEX environment that was passed into the associated optimization routine.

`cbdata`

A pointer passed from the optimization routine to the user-written callback function that identifies the problem being optimized. The only purpose for the `cbdata` pointer is to pass it to the routine `CPXgetcallbackinfo`.

`wherefrom`

An integer value specifying which optimization algorithm the user-written callback function was called from. Possible values and their meaning appear in the table.

Value	Symbolic Constant	Meaning
3	`CPX_CALLBACK_NETWORK`	From network simplex

`cbhandle`

Pointer to user private data, as passed to `CPXsetlpcallbackfunc`.

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`callback`

The address of the pointer to the current user-written callback function. If no callback function has been set, the pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the private pointer of the user.

Example

```
status = CPXgetnetcallbackfunc (env, mycallback, NULL);
```

See Also: CPXgetcallbackinfo

Returns:

A nonzero terminates the optimization.

Global function CPXfeasopt

```
int CPXfeasopt(CPXCENVptr env, CPXLPptr lp, const double * rhs, const double * rng,
const double * lb, const double * ub)
```

Definition file: cplex.h

The routine CPXfeasopt computes a minimum-cost relaxation of the righthand side values of constraints or bounds on variables in order to make an infeasible problem feasible. The routine also computes a relaxed solution vector that can be queried with CPXsolution, CPXgetcolinfeas for columns, CPXgetrowinfeas for rows, CPXgetsosinfeas for special ordered sets.

If CPXfeasopt finds a feasible solution, it returns the solution and the corresponding objective in terms of the original model.

This routine supports several options for the metric used to determine what constitutes a minimum-cost relaxation. These options are controlled by the parameter CPX_PARAM_FEASOPTMODE which can take the following values:

- CPX_FEASOPT_MIN_SUM 0
- CPX_FEASOPT_OPT_SUM 1
- CPX_FEASOPT_MIN_INF 2
- CPX_FEASOPT_OPT_INF 3
- CPX_FEASOPT_MIN_QUAD 4
- CPX_FEASOPT_OPT_QUAD 5

- It can minimize the weighted sum of the penalties for relaxations (denoted by SUM).
- It can minimize the weighted number of relaxed bounds and constraints (denoted by INF).
- It can minimize the weighted sum of the squared penalties of the relaxations (denoted by QUAD).

This routine can also optionally perform a secondary optimizaton (denoted by OPT in the name of the option), where it optimizes the original objective function over all possible relaxations for which the relaxation metric does not exceed the amount computed in the first phase. These options are controlled by the parameter CPX_PARAM_FEASOPTMODE. Thus, for example, the value CPX_FEASOPT_MIN_SUM denotes that CPXfeasopt should find a relaxation that minimizes the weighted sum of relaxations. Similarly, the value CPX_FEASOPT_OPT_INF specifies that CPXfeasopt should find a solution that optimizes the original objective function, choosing from among all possible relaxations that minimize the number of relaxed constraints and bounds.

If you use INF mode, the resulting feasopt problems will be MIPs even if your problem is continuous. Similarly, if you use QUAD mode, the feasopt problems will become quadratic even if your original problem is linear. This change in problem type can result in higher than expected solve times.

The user can specify preferences associated with relaxing a bound or righthand side value through input values of the rhs, rng, lb, and ub arguments. The input value denotes the user's willingness to relax a constraint or bound. More precisely, the reciprocal of the specified preference is used to weight the relaxation of that constraint or bound. For example, consider a preference of p on a constraint that is relaxed by 2 units. The penalty of this relaxation will be 1/p when minimizing the weighted number of infeasibilities; the penalty will be 2/p when minimizing the weighted sum of infeasibilities; and the penalty will be 4/p when minimizing the weighted sum of the squares of the infeasibilities. The user may specify a preference less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed.

To determine whether CPXfeasopt found relaxed values to make the problem feasible, call the routine CPXsolninfo for continuous problems or CPXgetstat for any problem type. CPXsolninfo will return a value of CPX_NO_SOLN for the argument solntype_p if CPXfeasopt could not find a feasible relaxation. Otherwise, it will return one of the following, depending on the value of CPX_PARAM_FEASOPTMODE:

- CPX_STAT_FEASIBLE_RELAXED_SUM
- CPX_STAT_OPTIMAL_RELAXED_SUM
- CPX_STAT_FEASIBLE_RELAXED_INF
- CPX_STAT_OPTIMAL_RELAXED_INF

- CPX_STAT_FEASIBLE_RELAXED_QUAD
- CPX_STAT_OPTIMAL_RELAXED_QUAD

For a MIP problem, the routine `CPXgetstat` will return a value of `CPXMIP_INFEASIBLE` or `CPX_STAT_INFEASIBLE` if it could not find a feasible relaxation. Otherwise, it will return one of the following, depending on the value of `CPX_PARAM_FEASOPTMODE`:

- CPXMIP_FEASIBLE_RELAXED_SUM
- CPXMIP_OPTIMAL_RELAXED_SUM
- CPXMIP_FEASIBLE_RELAXED_INF
- CPXMIP_OPTIMAL_RELAXED_INF
- CPXMIP_FEASIBLE_RELAXED_QUAD
- CPXMIP_OPTIMAL_RELAXED_QUAD

The routine `CPXfeasopt` accepts all problem types. However, it does not allow you to relax quadratic constraints nor indicator constraints; use the routine `CPXfeasoptext` for that purpose.

The parameters `CPX_PARAM_CUTUP`, `CPX_PARAM_CUTLO`, `CPX_PARAM_OBJULIM`, `CPX_PARAM_OBJLLIM` do **not** influence this routine. If you want to study infeasibilities introduced by those parameters, consider adding an objective function constraint to your model to enforce their effect before you invoke this routine.

Example

```
status = CPXfeasopt (env, lp, rhs, rng, lb, ub);
```

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`lp`

A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

`rhs`

An array of doubles of length at least equal to the number of rows in the problem. NULL may be specified if no rhs values are allowed to be relaxed. When not NULL, the array specifies the preference values that determine the cost of relaxing each constraint.

`rng`

An array of doubles of length at least equal to the number of rows in the problem. NULL may be specified if no range values are allowed to be relaxed or none are present in the active problem. When not NULL, the array specifies the preference values that determine the cost of relaxing each range.

`lb`

An array of doubles of length at least equal to the number of columns in the problem. NULL may be passed if no lower bound of any variable is allowed to be relaxed. When not NULL, the array specifies the preference values that determine the cost of relaxing each lower bound.

`ub`

An array of doubles of length at least equal to the number of columns in the problem. NULL may be passed if no upper bound of any variable is allowed to be relaxed. When not NULL, the array specifies the preference values that determine the cost of relaxing each upper bound.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsubstat

`int` **`CPXgetsubstat`**`(CPXCENVptr env, CPXCLPptr lp)`

Definition file: cplex.h

The routine `CPXgetsubstat` accesses the solution status of the last subproblem optimization, in the case of an error termination during mixed integer optimization.

Example

```
substatus = CPXgetsubstat (env, lp);
```

See Also: CPXgetsubmethod

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.

Example

```
substatus = CPXgetsubstat (env, lp);
```

Returns:

The routine returns zero if no solution exists. A nonzero return value reports that there was an error termination where a subproblem could not be solved to completion. The values returned are documented in the group optim.cplex.callable.solutionstatus in the reference manual of the API.

Global function CPXsetlpcallbackfunc

```
int CPXsetlpcallbackfunc(CPXENVptr env, int(CPXPUBLIC *callback)(CPXCENVptr, void
*, int, void *), void * cbhandle)
```

Definition file: cplex.h

The routine `CPXsetlpcallbackfunc` modifies the user-written callback routine to be called after each iteration during the optimization of a linear program, and also periodically during the CPLEX presolve algorithm.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle);
```

This is the user-written callback routine.

Callback return value

A nonzero terminates the optimization.

Callback arguments

`env`

A pointer to the CPLEX environment that was passed into the associated optimization routine.

`cbdata`

A pointer passed from the optimization routine to the user-written callback function that identifies the problem being optimized. The only purpose for the `cbdata` pointer is to pass it to the routine `CPXgetcallbackinfo`.

`wherefrom`

An integer value specifying from which optimization algorithm the user-written callback function was called. Possible values and their meaning appear in the table below.

Value	Symbolic Constant	Meaning
1	CPX_CALLBACK_PRIMAL	From primal simplex
2	CPX_CALLBACK_DUAL	From dual simplex
4	CPX_CALLBACK_PRIMAL_CROSSOVER	From primal crossover
5	CPX_CALLBACK_DUAL_CROSSOVER	From dual crossover
6	CPX_CALLBACK_BARRIER	From barrier
7	CPX_CALLBACK_PRESOLVE	From presolve
8	CPX_CALLBACK_QPBARRIER	From QP barrier
9	CPX_CALLBACK_QPSIMPLEX	From QP simplex

`cbhandle`

Pointer to user private data, as passed to `CPXsetlpcallbackfunc`.

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`myfunc`

A pointer to a user-written callback function. Setting `callback` to NULL prevents any callback function from being called during optimization. The call to `callback` occurs after every iteration during optimization and periodically during the CPLEX presolve algorithms. This function is written by the user, and is prototyped as documented here.

`cbhandle`

A pointer to user private data. This pointer is passed to the callback function.

Example

```
status = CPXsetlpcallbackfunc (env, myfunc, NULL);
```

See Also: CPXgetcallbackinfo, CPXsetmipcallbackfunc, CPXsetnetcallbackfunc

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXinfodblparam

```
int CPXinfodblparam(CPXCENVptr env, int whichparam, double * defvalue_p, double *
minvalue_p, double * maxvalue_p)
```

Definition file: cplex.h

The routine `CPXinfodblparam` obtains the default, minimum, and maximum values of a CPLEX parameter of type `double`.

> **Note**
>
> Values of zero obtained for both the minimum and maximum values of a parameter of type `double` mean that the parameter has no limit.

The *CPLEX Parameters Reference Manual* provides a list of parameters with their types, options, and default values.

Example

```
status = CPXinfodblparam (env, CPX_PARAM_TILIM, &default_tilim,
                          &min_tilim, &max_tilim);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
whichparam	The symbolic constant (or reference number) of the parameter value to be obtained.
defvalue_p	A pointer to a variable of type `double` to hold the default value of the CPLEX parameter. May be NULL.
minvalue_p	A pointer to a variable of type `double` to hold the minimum value of the CPLEX parameter. May be NULL.
maxvalue_p	A pointer to a variable of type `double` to hold the maximum value of the CPLEX parameter. May be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXreadcopybase

```
int CPXreadcopybase(CPXCENVptr env, CPXLPptr lp, const char * filename_str)
```

Definition file: cplex.h

The routine `CPXreadcopybase` reads a basis from a BAS file, and copies that basis into a CPLEX problem object. The parameter `CPX_PARAM_ADVIND` must be set to 1 (one), its default value, or 2 (two) in order for the basis to be used for starting a subsequent optimization.

Example

```
status = CPXreadcopybase (env, lp, "myprob.bas");
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	The name of the file from which the basis should be read.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXchgprobname

`int` **`CPXchgprobname`**`(CPXCENVptr env, CPXLPptr lp, const char * probname_str)`

Definition file: cplex.h

The routine `CPXchgprobname` changes the name of the current problem.

Example

```
status = CPXchgprobname (env, lp, probname);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
probname_str	The new name of the problem.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetctype

```
int CPXgetctype(CPXCENVptr env, CPXCLPptr lp, char * xctype, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetctype` accesses the types for a range of variables in a problem object. The beginning and end of the range must be specified.

Example

```
status = CPXgetctype (env, lp, ctype, 0, cur_numcols-1);
```

See Also: CPXcopyctype

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
xctype	An array where the specified types are to be returned. This array must be of length (`end - begin + 1`). The type of variable `j` is returned in `ctype[j-begin]`. See the routine `CPXcopyctype` for a list of possible values for the variables in `ctype`.
begin	An integer specifying the beginning of the range of types to be returned
end	An integer specifying the end of the range of types to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsolnpoolqconstrslack

`int` **`CPXgetsolnpoolqconstrslack`**`(CPXCENVptr env, CPXCLPptr lp, int soln, double * qcslack, int begin, int end)`

Definition file: cplex.h

The routine `CPXgetsolnpoolqconstrslack` accesses the slack values for a range of the quadratic constraints for a member of the solution pool of a quadratically constrained program (QCP). The beginning and end of the range must be specified. The slack values returned consist of the righthand side minus the constraint activity level.

Example

```
status = CPXgetsolnpoolqconstrslack (env, lp, 3, qcslack, 0, CPXgetnumqconstrs(env,lp)-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
soln	An integer specifying the index of the solution pool member for which to return slack values. A value of -1 specifies that the incumbent should be used instead of a solution pool member.
qcslack	An array to receive the values of the slack or surplus variables for each of the constraints. This array must be of length at least (`end - begin+1`). If successful, `qcslack[0]` through `qcslack[end-begin]` contain the values of the slacks.
begin	An integer specifying the beginning of the range of slack values to be returned.
end	An integer specifying the end of the range of slack values to be returned.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXmsg

```
int CPXPUBVARARGS CPXmsg(CPXCHANNELptr channel, const char * format, ...)
```

Definition file: cplex.h

The routine `CPXmsg` writes a message to a specified channel. Like the C function `printf`, it takes a variable number of arguments comprising the message to be written. The list of variables specified after the format string should be at least as long as the number of format codes in the format. The format string and variables are processed by the C library function `vsprintf` or a substitute on systems that do not have the `vsprintf` function.

The formatted string is limited to 1024 characters, and is usually output with the C function `fputs` to each output destination in the output destination list for a channel, except when a function has been specified by the routine `CPXaddfuncdest` as a destination.

The CPLEX Callable Library uses `CPXmsg` for all message output. The `CPXmsg` routine may also be used in applications to send messages to either CPLEX-defined or user-defined channels.

> **Note**
>
> `CPXmsg` is the only nonadvanced CPLEX routine not requiring the CPLEX environment as an argument.

Example

```
CPXmsg (mychannel, "The objective value was %f.n", objval);
```

See `lpex5.c` in the *CPLEX User's Manual*.

Parameters:

channel	The pointer to the channel receiving the message.
format	The format string controlling the message output. This string is used in a way identical to the format string in a `printf` statement.

Returns:

At completion, `CPXmsg` returns the number of characters in the formatted result string.

Global function CPXdelfuncdest

```
int CPXdelfuncdest(CPXCENVptr env, CPXCHANNELptr channel, void * handle,
void(CPXPUBLIC *msgfunction)(void *, const char *))
```

Definition file: cplex.h

The routine `CPXdelfuncdest` removes the function `msgfunction` from the list of message destinations associated with a channel. Use `CPXdelfuncdest` to remove functions that were added to the list using `CPXaddfuncdest`.

To illustrate, consider an application in which a developer wishes to trap CPLEX error messages and display them in a dialog box that prompts the user for an action. Use `CPXaddfuncdest` to add the address of a function to the list of message destinations associated with the `cpxerror` channel. Then write the `msgfunction` routine. It must contain the code that controls the dialog box. When `CPXmsg` is called with `cpxerror` as its first argument, it calls the `msgfunction` routine, which then displays the error message.

Note

The handle argument is a generic pointer that can be used to hold information needed by the `msgfunction` routine to avoid making such information global to all routines.

Example

```
void msgfunction (void *handle, char *msg_string)
{
    FILE *fp;
    fp = (FILE *)handle;
    fprintf (fp, "%s", msg_string);
}
status = CPXdelfuncdest (env, mychannel, fileptr, msgfunction);
```

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`channel`

The pointer to the channel to which the function destination is to be added.

`handle`

A void pointer that can be used in the `msgfunction` routine to direct the message to a file, the screen, or a memory location.

`msgfunction`

The pointer to the function to be called when a message is sent to a channel. For details about this callback function, see `CPXaddfuncdest`.

See Also: CPXaddfuncdest

Returns:

The routines return zero if successful and nonzero if an error occurs. Failure occurs when `msgfunction` is not in the message-destination list or the channel does not exist.

Global function CPXgetheuristiccallbackfunc

```
void CPXgetheuristiccallbackfunc(CPXCENVptr env, int(CPXPUBLIC
**heuristiccallback_p)(CALLBACK_HEURISTIC_ARGS), void ** cbhandle_p)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetheuristiccallbackfunc` accesses the user-written callback to be called by CPLEX during MIP optimization after the subproblem has been solved to optimality. That callback is not called when the subproblem is infeasible or cut off. The callback supplies CPLEX with heuristically-derived integer solutions.

Example

```
CPXgetheuristiccallbackfunc(env, &current_callback, &current_handle);
```

See also *Advanced MIP Control Interface* in the *CPLEX User's Manual*.

For documentation of callback arguments, see the routine `CPXsetheuristiccallbackfunc`.

Parameters

`env`

A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.

`heuristiccallback_p`

The address of the pointer to the current user-written heuristic callback. If no callback has been set, the pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the user's private pointer.

See Also: CPXsetheuristiccallbackfunc

Returns:

This routine does not return a result.

Global function CPXcopylp

```
int CPXcopylp(CPXCENVptr env, CPXLPptr lp, int numcols, int numrows, int objsense,
const double * objective, const double * rhs, const char * sense, const int *
matbeg, const int * matcnt, const int * matind, const double * matval, const double
* lb, const double * ub, const double * rngval)
```

Definition file: cplex.h

The routine `CPXcopylp` copies data that define an LP problem to a CPLEX problem object. The arguments to `CPXcopylp` define an objective function, the constraint matrix, the righthand side, and the bounds on the variables. Calling `CPXcopylp` destroys any existing data associated with the problem object.

The routine `CPXcopylp` does **not** copy names. The more comprehensive routine `CPXcopylpwnames` can be used in place of `CPXcopylp` to copy linear programs with associated names.

The arguments passed to `CPXcopylp` define a linear program. Since these arguments are copied into local arrays maintained by CPLEX, the LP problem data passed via `CPXcopylp` may be modified or freed after the call to `CPXcopylp` without affecting the state of the CPLEX problem object.

Table 1: Values of objsense

`objsense`	`= 1`	(`CPX_MIN`) minimize
`objsense`	`= -1`	(`CPX_MAX`) maximize

Table 2: Values of sense

`sense[i]`	`= 'L'`	<= constraint
`sense[i]`	`= 'E'`	= constraint
`sense[i]`	`= 'G'`	>= constraint
`sense[i]`	`= 'R'`	ranged constraint

The arrays `matbeg`, `matcnt`, `matind`, and `matval` are accessed as follows. Suppose that CPLEX wants to access the entries in some column `j`. These are assumed to be given by the array entries:

```
matval[matbeg[j]],.., matval[matbeg[j]+matcnt[j]-1]
```

The corresponding row indices are:

```
matind[matbeg[j]],.., matind[matbeg[j]+matcnt[j]-1]
```

Entries in `matind` are not required to be in row order. Duplicate entries in `matind` within a single column are not allowed. The length of the arrays `matbeg` and `matind` should be at least `numcols`. The length of arrays `matind` and `matval` should be at least `matbeg[numcols-1]+matcnt[numcols-1]`.

When you build or modify your problem with this routine, you can verify that the results are as you intended by calling `CPXcheckcopylp` during application development.

Example

```
status = CPXcopylp (env, lp, numcols, numrows, objsen, obj, rhs,
                    sense, matbeg, matcnt, matind, matval, lb,
                    ub, rngval);
```

See also the example `lpex1.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
numcols	An integer that specifies the number of columns in the constraint matrix, or equivalently, the number of variables in the problem object.
numrows	An integer that specifies the number of rows in the constraint matrix, not including the objective function or bounds on the variables.
objsense	An integer that specifies whether the problem is a minimization or maximization problem.
objective	An array of length at least `numcols` containing the objective function coefficients.
rhs	An array of length at least `numrows` containing the righthand side value for each constraint in the constraint matrix.
sense	An array of length at least `numrows` containing the sense of each constraint in the constraint matrix.
matbeg	An array that with `matval`, `matcnt`, and `matind` defines the constraint matrix.
matcnt	An array that with `matbeg`, `matval`, and `matind` defines the constraint matrix.
matind	An array that with `matbeg`, `matcnt`, and `matval` defines the constraint matrix.
matval	An array that with `matbeg`, `matcnt`, and `matind` defines the constraint matrix. CPLEX needs to know only the nonzero coefficients. These are grouped by column in the array `matval`. The nonzero elements of every column must be stored in sequential locations in this array with `matbeg[j]` containing the index of the beginning of column `j` and `matcnt[j]` containing the number of entries in column `j`. The components of `matbeg` must be in ascending order. For each k, `matind[k]` specifies the row number of the corresponding coefficient, `matval[k]`.
lb	An array of length at least `numcols` containing the lower bound on each of the variables. Any lower bound that is set to a value less than or equal to that of the constant `-CPX_INFBOUND` is treated as negative infinity. `CPX_INFBOUND` is defined in the header file `cplex.h`.
ub	An array of length at least `numcols` containing the upper bound on each of the variables. Any upper bound that is set to a value greater than or equal to that of the constant `CPX_INFBOUND` is treated as infinity. `CPX_INFBOUND` is defined in the header file `cplex.h`.
rngval	An array of length at least `numrows` containing the range value of each ranged constraint. Ranged rows are those designated by `'R'` in the `sense` array. If the row is not ranged, the `rngval` array entry is ignored. If `rngval[i]` > 0, then row `i` activity is in `[rhs[i],rhs[i]+rngval[i]]`, and if `rngval[i]` <= 0,then row `i` activity is in `[rhs[i]+rngval[i],rhs[i]]`. This argument may be NULL.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXcrushform

```
int CPXcrushform(CPXCENVptr env, CPXCLPptr lp, int len, const int * ind, const
double * val, int * plen_p, double * poffset_p, int * pind, double * pval)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXcrushform` crushes a linear formula of the original problem to a linear formula of the presolved problem.

Example

```
status = CPXcrushform (env, lp, len, ind, val,
                       &plen, &poffset, pind, pval);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
len	The number of entries in the arrays `ind` and `val`.
ind	An array to hold the column indices of coefficients in the array `val`.
val	The linear formula in terms of the original problem. Each entry, `ind[i]`, specifies the column index of the corresponding coefficient, `val[i]`.
plen_p	A pointer to an integer to receive the number of nonzero coefficients, that is, the true length of the arrays `pind` and `pval`.
poffset_p	A pointer to a double to contain the value of the linear formula corresponding to variables that have been removed in the presolved problem.
pind	An array to hold the column indices of coefficients in the presolved problem in the array `pval`.
pval	The linear formula in terms of the presolved problem. Each entry, `pind[i]`, specifies the column index in the presolved problem of the corresponding coefficient, `pval[i]`. The arrays `pind` and `pval` must be of length at least the number of columns in the presolved LP problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXwriteparam

```
int CPXwriteparam(CPXCENVptr env, const char * filename_str)
```

Definition file: cplex.h

The routine `CPXwriteparam` writes the name and current setting of CPLEX parameters that are not at their default setting in the environment specified by `env`.

This routine writes a file in a format suitable for reading by `CPXreadcopyparam`, so you can save current, nondefault parameter settings for re-use in a later session. The file is written in the PRM format which is documented in the *CPLEX File Formats Reference Manual*.

```
status = CPXwriteparam (env, "myparams.prm");
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
filename_str	A character string containing the name of the file to which the current set of modified parameter settings is to be written.

Global function CPXNETdelarcs

`int` **`CPXNETdelarcs`**`(CPXCENVptr env, CPXNETptr net, int begin, int end)`

Definition file: cplex.h

The routine `CPXNETdelarcs` is used to remove a range of arcs from the network stored in a network problem object. The remaining arcs are renumbered starting at zero; their order is preserved. If removing arcs disconnects some nodes from the rest of the network, the disconnected nodes remain part of the network.

Any solution information stored in the problem object is lost.

Example

```
status = CPXNETdelarcs (env, net, 10, 20);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
net	A pointer to a CPLEX network problem object as returned by `CPXNETcreateprob`.
begin	Index of the first arc to be deleted.
end	Index of the last arc to be deleted.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXgetcolname

```
int CPXgetcolname(CPXCENVptr env, CPXCLPptr lp, char ** name, char * namestore, int
storespace, int * surplus_p, int begin, int end)
```

Definition file: cplex.h

The routine `CPXgetcolname` accesses a range of column names or, equivalently, the variable names of a CPLEX problem object. The beginning and end of the range, along with the length of the array in which the column names are to be returned, must be specified.

Note

If the value of `storespace` is 0, the negative of the value of `surplus_p` returned specifies the total number of characters needed for the array `namestore`.

Example

```
status = CPXgetcolname (env, lp, cur_colname, cur_colnamestore,
                        cur_storespace, &surplus, 0,
                        cur_numcols-1);
```

See also the example `lpex7.c` in the *CPLEX User's Manual* and in the standard distribution.

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
name	An array of pointers to the column names stored in the array `namestore`. This array must be of length at least (`end - begin + 1`). The pointer to the name of column `j` is returned in `name[j-begin]`.
namestore	An array of characters where the specified column names are to be returned. May be NULL if `storespace` is 0.
storespace	An integer specifying the length of the array `namestore`. May be 0.
surplus_p	A pointer to an integer to contain the difference between `storespace` and the total amount of memory required to store the requested names. A nonnegative value of `surplus_p` specifies that `storespace` was sufficient. A negative value specifies that it was insufficient and that the routine could not complete its task. In that case, `CPXgetcolname` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of the variable `surplus_p` specifies the amount of insufficient space in the array `namestore`.
begin	An integer specifying the beginning of the range of column names to be returned.
end	An integer specifying the end of the range of column names to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` specifies that insufficient space was available in the `namestore` array to hold the names.

Global function CPXgetrowindex

```
int CPXgetrowindex(CPXCENVptr env, CPXCLPptr lp, const char * lname_str, int *
index_p)
```

Definition file: cplex.h

The routine `CPXgetrowindex` searches for the index number of the specified row in a CPLEX problem object.

Example

```
status = CPXgetrowindex (env, lp, "resource89", &rowindex);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lname_str	A row name to search for.
index_p	A pointer to an integer to hold the index number of the row with name `lname_str`. If the routine is successful, `*index_p` contains the index number; otherwise, `*index_p` is undefined.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXdelsetrows

`int` **`CPXdelsetrows`**`(CPXCENVptr env, CPXLPptr lp, int * delstat)`

Definition file: cplex.h

The routine `CPXdelsetrows` deletes a set of rows. Unlike the routine `CPXdelrows`, `CPXdelsetrows` does not require the rows to be in a contiguous range. After the deletion occurs, the remaining rows are indexed consecutively starting at 0, and in the same order as before the deletion.

Note

The `delstat` array must have at least `CPXgetnumrows(env,lp)` elements.

Example

```
status = CPXdelsetrows (env, lp, delstat);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
delstat	An array specifying the rows to be deleted. The routine `CPXdelsetrows` deletes each row `i` for which `delstat[i] = 1`. The deletion of rows results in a renumbering of the remaining rows. After termination, `delstat[i]` is either -1 for rows that have been deleted or the new index number that has been assigned to the remaining rows.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetcolindex

```
int CPXgetcolindex(CPXCENVptr env, CPXCLPptr lp, const char * lname_str, int *
index_p)
```

Definition file: cplex.h

The routine `CPXgetcolindex` searches for the index number of the specified column in a CPLEX problem object.

Example

```
status = CPXgetcolindex (env, lp, "power43", &colindex);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
lname_str	A column name to search for.
index_p	A pointer to an integer to hold the index number of the column with name `lname_str`. If the routine is successful, `*index_p` contains the index number; otherwise, `*index_p` is undefined.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXclpwrite

```
int CPXclpwrite(CPXCENVptr env, CPXCLPptr lp, const char * filename_str)
```

Definition file: cplex.h

After `CPXrefineconflict` or `CPXrefineconflictext` has been invoked on an infeasible problem to identify a minimal set of constraints that are in conflict, this routine will write an LP format file containing the identified conflict. The names will be modified to conform to LP format.

Example

```
status = CPXclpwrite (env, lp, "myfilename.clp");
```

Parameters:

env	A pointer to the CPLEX environment as returned by the routine `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
filename_str	Pointer to a character string naming the file.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetpnorms

```
int CPXgetpnorms(CPXCENVptr env, CPXCLPptr lp, double * cnorm, double * rnorm, int
* len_p)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetpnorms` returns the norms from the primal steepest-edge.

There is no comparable argument in this routine for `rnorm[]`. If the rows of the problem have changed since the norms were computed, they are generally no longer valid. However, if columns have been deleted, or if columns have been added, the norms for all remaining columns present before the deletions or additions remain valid.

See Also: CPXcopypnorms

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
cnorm	An array containing the primal steepest-edge norms for the normal, column variables. The array must be of length at least equal to the number of columns in the LP problem object.
rnorm	An array containing the primal steepest-edge norms for ranged variables and slacks. The array must be of length at least equal to the number of rows in the LP problem object.
len_p	A pointer to the number of entries in the array `cnorm[]`. When this routine is called, `*len_p` is equal to the number of columns in the LP problem object when optimization occurred. The routine `CPXcopypnorms` needs the value `*len_p`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsolnpoolslack

`int` **`CPXgetsolnpoolslack`**`(CPXCENVptr env, CPXCLPptr lp, int soln, double * slack, int begin, int end)`

Definition file: cplex.h

The routine `CPXgetsolnpoolslack` accesses the slack values for a range of linear constraints for a member of the solution pool. The beginning and end of the range must be specified. Except for ranged rows, the slack values returned consist of the righthand side minus the row activity level. For ranged rows, the value returned is the row activity level minus the righthand side, or, equivalently, the value of the internal structural variable that CPLEX creates to represent ranged rows.

Example

```
status = CPXgetsolnpoolslack (env, lp, 3, slack, 0, CPXgetnumrows(env,lp)-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
soln	An integer specifying the index of the solution pool member for which to return slack values. A value of -1 specifies that the incumbent should be used instead of a solution pool member.
slack	An array to receive the values of the slack or surplus variables for each of the constraints. This array must be of length at least (`end - begin + 1`). If successful, `slack[0]` through `slack[end-begin]` contain the values of the slacks.
begin	An integer specifying the beginning of the range of slack values to be returned.
end	An integer specifying the end of the range of slack values to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgettuningcallbackfunc

```
int CPXgettuningcallbackfunc(CPXCENVptr env, int(CPXPUBLIC
**callback_p)(CPXCENVptr, void *, int, void *), void ** cbhandle_p)
```

Definition file: cplex.h

The routine `CPXgettuningcallbackfunc` accesses the user-written callback routine to be called before each trial run during the tuning process.

Callback description

```
int callback (CPXCENVptr env,
              void       *cbdata,
              int        wherefrom,
              void       *cbhandle);
```

This is the user-written callback routine.

Callback return value

A nonzero terminates the tuning.

Callback arguments

A pointer to the CPLEX environment that was passed into the associated tuning routine.

`cbdata`

A pointer passed from the tuning routine to the user-written callback function that contains information about the tuning process. The only purpose for the `cbdata` pointer is to pass it to the routine `CPXgetcallbackinfo`.

`wherefrom`

An integer value specifying from which procedure the user-written callback function was called. This value will always be `CPX_CALLBACK_TUNING` for this callback.

`cbhandle`

Pointer to user private data, as passed to `CPXsettuningcallbackfunc`.

Parameters

`env`

A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

`callback_p`

The address of the pointer to the current user-written callback function. If no callback function has been set, the pointer evaluates to NULL.

`cbhandle_p`

The address of a variable to hold the user's private pointer.

Example

```
status = CPXgettuningcallbackfunc (env, mycallback, NULL);
```

See Also: CPXgetcallbackinfo

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXfclose

```
int CPXfclose(CPXFILEptr stream)
```

Definition file: cplex.h

The routine `CPXfclose` closes files that are used in conjunction with the routines `CPXaddfpdest`, `CPXdelfpdest`, and `CPXsetlogfile`. It is used in the same way as the standard C library function `fclose`. Files that are opened with the routine `CPXfopen` must be closed with the routine `CPXfclose`.

When to use this routine

Call this routine only **after** the message destinations that use the file pointer have been closed or deleted. Those destinations (such as log files) might be specified by routines such as `CPXaddfpdest`, `CPXdelfpdest`, and `CPXsetlogfile`.

Example

```
CPXfclose (fp);
```

See `lpex5.c` in the *CPLEX User's Manual*.

Parameters:

stream A pointer to a file opened by the routine `CPXfopen`.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The syntax is identical to the standard C library routine `fclose`.

Global function CPXversion

`CPXCCHARptr` **`CPXversion`**`(CPXCENVptr env)`

Definition file: cplex.h

The routine `CPXversion` returns a pointer to a string specifying the version of the CPLEX library linked with the application. The caller should not change the string returned by this function.

Example

```
printf ("CPLEX version is %s n", CPXversion (env));
```

Parameters:

env A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.

Returns:

The routine returns NULL if the environment does not exist and the pointer to a string otherwise.

Global function CPXuncrushx

```
int CPXuncrushx(CPXCENVptr env, CPXCLPptr lp, double * x, const double * prex)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXuncrushx` uncrushes a solution for the presolved problem to the solution for the original problem.

Example

```
status = CPXuncrushx (env, lp, x, prex);
```

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.
x	An array to receive the primal solution (`x`) values for the original problem as computed from primal values of the presolved problem object. The array must be of length at least the number of columns in the LP problem object.
prex	An array that contains primal solution (`x`) values for the presolved problem, as returned by routines such as `CPXgetx` and `CPXsolution` when applied to the presolved problem object. The array must be of length at least the number of columns in the presolved problem object.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetijrow

```
int CPXgetijrow(CPXCENVptr env, CPXCLPptr lp, int i, int j, int * row_p)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXgetijrow` returns the index of a specific basic variable as its position in the basis header. If the specified row indexes a constraint that is not basic, or if the specified column indexes a variable that is not basic, `CPXgetijrow` returns an error code and sets the value of its argument `*row_p` to `-1`. An error is also returned if both row and column indices are specified in the same call.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	The pointer to the CPLEX LP problem object, as returned by `CPXcreateprob`.
i	An integer specifying the index of a basic row; `CPXgetijrow` must find the position of this basic row in the basis header. A negative value in this argument specifies to `CPXgetijrow` not to seek a basic row.
j	An integer specifying the index of a basic column; `CPXgetijrow` must find the position of this basic column in the basis header. A negative value in this argument specifies to `CPXgetijrow` not to seek a basic column.
row_p	A pointer to an integer specifying the position in the basis header of the row `i` or column `j`. If `CPXgetijrow` encounters an error, and if `row_p` is not NULL, `*row_p` is set to `-1`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXchgcoef

```
int CPXchgcoef(CPXCENVptr env, CPXLPptr lp, int i, int j, double newvalue)
```

Definition file: cplex.h

The routine `CPXchgcoef` changes a single coefficient in the constraint matrix, linear objective coefficients, righthand side, or ranges of a CPLEX problem object. The coefficient is specified by its coordinates in the constraint matrix. When you change matrix coefficients from zero to nonzero values, be sure that the corresponding row and column indices exist in the problem, so that `-1 <= i < CPXgetnumrows(env,lp)` and `-2 <= j < CPXgetnumcols(env,lp)`.

Example

```
status = CPXchgcoef (env, lp, 10, 15, 23.2);
```

See Also: CPXchgobj, CPXchgrhs, CPXchgrngval

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
i	An integer that specifies the numeric index of the row in which the coefficient is located. The linear objective row is referenced with `i = -1`.
j	An integer that specifies the numeric index of the column in which the coefficient is located. The RHS column is referenced with `j = -1`. The range value column is referenced with `j = -2`. If `j = -2` is specified and row `i` is not a ranged row, an error status is returned.
newvalue	The new value for the coefficient being changed.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetqconstrslack

```
int CPXgetqconstrslack(CPXCENVptr env, CPXCLPptr lp, double * qcslack, int begin,
int end)
```

Definition file: cplex.h

The routine `CPXgetqconstrslack` is used to access the slack values for a range of the quadratic constraints of a quadratically constrained program. The beginning and end of the range must be specified. The slack values returned consist of the righthand side minus the constraint activity level.

Example

```
status = CPXgetqconstrslack (env, lp, qcslack, 0, CPXgetnumqconstrs(env,lp)-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
qcslack	An array to receive the values of the slack or surplus variables for each of the constraints. This array must be of length at least (`end - begin+1`). If successful, `qcslack[0]` through `qcslack[end-begin]` contain the values of the slacks.
begin	An integer indicating the beginning of the range of slack values to be returned.
end	An integer indicating the end of the range of slack values to be returned.

Returns:

The routine returns zero on success and nonzero if an error occurs.

Global function CPXunscaleprob

```
int CPXunscaleprob(CPXCENVptr env, CPXLPptr lp)
```

Definition file: cplex.h

> **Note**
>
> This is an advanced routine. Advanced routines typically demand a thorough understanding of the algorithms used by CPLEX. Thus they incur a higher risk of incorrect behavior in your application, behavior that can be difficult to debug. Therefore, the team encourages you to consider carefully whether you can accomplish the same task by means of other Callable Library routines instead.

The routine `CPXunscaleprob` removes any scaling that CPLEX has applied to the resident problem and its associated data. A side effect is that if there is a resident solution, any associated factorization is discarded and the solution itself is deactivated, meaning that it can no longer be accessed with a call to `CPXsolution`, nor by any other query routine. However, any starting point information for the current solution (such as an associated basis) is retained.

Parameters:

env	The pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX LP problem object, as returned by `CPXcreateprob`.

Returns:

The routine returns zero if successful and nonzero if an error occurs.

Global function CPXgetsos

```
int CPXgetsos(CPXCENVptr env, CPXCLPptr lp, int * numsosnz_p, char * sostype, int *
sosbeg, int * sosind, double * soswt, int sosspace, int * surplus_p, int begin, int
end)
```

Definition file: cplex.h

The routine `CPXgetsos` accesses the definitions of a range of special ordered sets (SOS) stored in a CPLEX problem object. The beginning and end of the range, along with the length of the array in which the definitions are to be returned, must be provided.

Note

If the value of `sosspace`is 0 (zero), then the negative of the value of `surplus_p` returned specifies the length needed for the arrays `sosind` and `soswt`.

Example

```
status = CPXgetsos (env, lp, &numsosnz, sostype, sosbeg, sosind,
                    soswt, sosspace, &surplus, 0, numsos-1);
```

Parameters:

env	A pointer to the CPLEX environment as returned by `CPXopenCPLEX`.
lp	A pointer to a CPLEX problem object as returned by `CPXcreateprob`.
numsosnz_p	A pointer to an integer to contain the number of set members returned; that is, the true length of the arrays `sosind` and `soswt`.
sostype	An array to contain the types of the requested SOSs. The type of set `k` is returned in `sostype[k-begin]`. This array must be of length at least (`end - begin+ 1`). The entry contains either `CPX_TYPE_SOS1` ('1') for type 1 or `CPX_TYPE_SOS2` ('2'), for type 2.
sosbeg	An array to contain indices specifying where each of the requested SOSs begins in the arrays `sosind` and `soswt`. Specifically, set `k` consists of the entries in `sosind` and `soswt` in the range from `sosbeg[k-begin]` to `sosbeg[(k+1) - begin] - 1`. (Set `end` consists of the entries from `sosbeg[end - begin]` to `numsosnz_p - 1`.) This array must be of length at least (`end - begin+ 1`).
sosind	An array to contain the variable indices of the SOS members. May be NULL if `sosspace` is 0 (zero).
soswt	An array to contain the reference values (weights) for SOS members. May be NULL if `sosspace` is 0 (zero). Weight `soswt[k]` corresponds to `sosind[k]`.
sosspace	An integer specifying the length of the arrays `sosind` and `soswt`. May be 0 (zero).
surplus_p	A pointer to an integer to contain the difference between `sosspace` and the number of entries in each of the arrays `sosind` and `soswt`. A nonnegative value of `surplus_p` reports that the length of the arrays was sufficient. A negative value reports that the length was insufficient and that the routine could not complete its task. In this case, the routine `CPXgetsos` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the negative value of `surplus_p` specifies the amount of insufficient space in the arrays.
begin	An integer specifying the beginning of the range of SOSs to be returned.
end	An integer specifying the end of the range of SOSs to be returned.

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` reports that insufficient space was available in the arrays `sosind` and `soswt` to hold the SOS definition.

Global function CPXgetchgparam

`int` **`CPXgetchgparam`**`(CPXCENVptr env, int * cnt_p, int * paramnum, int pspace, int * surplus_p)`

Definition file: cplex.h

The routine `CPXgetchgparam` returns an arrary of parameter numbers (unique identifiers) for parameters which are not set at their default values.

Parameters:

env	A pointer to the CPLEX environment, as returned by `CPXopenCPLEX`.
cnt_p	A pointer to an integer to contain the number of parameter numbers (unique identifiers) returned, that is, the true length of the array `paramnum`.
paramnum	The array to contain the numbers of the parameters with nondefault values.
pspace	An integer specifying the length of the array `paramnum`.
surplus_p	A pointer to an integer to contain the difference between `pspace` and the number of entries in `paramnum`. A nonnegative value of `surplus_p` specifies that the length of the arrays was sufficient. A negative value specifies that the length was insufficient and that the routine could not complete its task. In that case, the routine `CPXgetchgparam` returns the value `CPXERR_NEGATIVE_SURPLUS`, and the value of `surplus_p` specifies the amount of insufficiency (that is, how much more space is needed in the arrays).

Returns:

The routine returns zero if successful and nonzero if an error occurs. The value `CPXERR_NEGATIVE_SURPLUS` specifies that insufficient space was available in the array `paramnum` to hold the parameter numbers (unique identifiers) with nondefault values.

Macro CPX_DUAL_OBJ

Definition file: cplex.h

CPX_DUAL_OBJ

Concert Technology enum: DualObj.
Numeric meaning (`double`): To access the objective value relative to the dual barrier solution. This feature is available only for a **barrier** solution.

Integer meaning: not applicable

Macro CPX_EXACT_KAPPA

Definition file: cplex.h

CPX_EXACT_KAPPA

Concert Technology enum: ExactKappa.
Numeric meaning (`double`): To access the exact condition number of the scaled basis matrix. This feature is available only for a **simplex** solution

Integer meaning: not applicable

Macro CPX_KAPPA

Definition file: cplex.h

`CPX_KAPPA`

Concert Technology enum: Kappa.
Numeric meaning (`double`): To access the estimated condition number of the scaled basis matrix. This feature is available only for a **simplex** solution

Integer meaning: not applicable

Macro CPX_MAX_COMP_SLACK

Definition file: cplex.h

CPX_MAX_COMP_SLACK

Concert Technology enum: MaxCompSlack.
Numeric meaning (`double`): To access the maximum violation of the complementary slackness conditions for the unscaled problem. This feature is available only for a **barrier** solution

Integer meaning: To access the lowest index of a row or column with the largest violation of the complementary slackness conditions. An index (such as `*quality_p`) strictly less than zero denotes row (-i-1) or the slack variable for that row, in the case of columns. This feature is available only for a **barrier** solution.

Macro CPX_MAX_DUAL_INFEAS

Definition file: cplex.h

`CPX_MAX_DUAL_INFEAS`

Concert Technology enum: MaxDualInfeas.
Numeric meaning (`double`): To access the maximum of dual infeasibility or, equivalently, the maximum reduced-cost infeasibility for the unscaled problem

Integer meaning: To access the lowest index where the maximum dual infeasibility occurs for the unscaled problem

Macro CPX_MAX_DUAL_RESIDUAL

Definition file: cplex.h

CPX_MAX_DUAL_RESIDUAL

Concert Technology enum: MaxDualResidual.
Numeric meaning (`double`): To access maximum dual residual value. For a **simplex** solution, this is the maximum of the vector `|c-B'pi|`, and for a **barrier** solution, it is the maximum of the vector `|A'pi+rc-c|` for the unscaled problem

Integer meaning: To access the lowest index where the maximum dual residual occurs for the unscaled problem

Macro CPX_MAX_INDSLACK_INFEAS

Definition file: cplex.h

`CPX_MAX_INDSLACK_INFEAS`

Concert Technology enum: not applicable.
Numeric meaning (`double`): To access the maximum infeasibility of the indicator constraints, or equivalently, the maximum bound violation of the indicator constraint slacks.

Integer meaning: To acces the lowest index of the indicator constraints where the maximum indicator slack infeasibility occurs.

Can use a supplied primal solution.

Concert Technology does not distinguish indicator constraints from linear constraints in this respect.

Macro CPX_MAX_INT_INFEAS

Definition file: cplex.h

`CPX_MAX_INT_INFEAS`

Concert Technology enum: MaxIntInfeas.
Numeric meaning (`double`): To access the maximum of integer infeasibility for the unscaled problem

Integer meaning: To access the lowest index where the maximum integer infeasibility occurs for the unscaled problem

Can use a supplied primal solution.

Macro CPX_MAX_PI

Definition file: cplex.h

`CPX_MAX_PI`

Concert Technology enum: MaxPi.
Numeric meaning (`double`): To access the maximum absolute value in the dual solution vector for the unscaled problem

Integer meaning: To access the lowest index where the maximum pi value occurs for the unscaled problem

Macro CPX_MAX_PRIMAL_INFEAS

Definition file: cplex.h

`CPX_MAX_PRIMAL_INFEAS`

Concert Technology enum: MaxPrimalInfeas.
Numeric meaning (`double`): To access the maximum primal infeasibility or, equivalently, the maximum bound violation including slacks for the unscaled problem

Integer meaning: To access the lowest index of a column or row where the maximum primal infeasibility occurs for the unscaled problem. An index (such as `*quality_p`) strictly less than zero specifies that the maximum occurs at the slack variable for row (-i-1).

Can use a supplied primal solution.

Macro CPX_MAX_PRIMAL_RESIDUAL

Definition file: cplex.h

`CPX_MAX_PRIMAL_RESIDUAL`

Concert Technology enum: MaxPrimalResidual.
Numeric meaning (`double`): To access the maximum of the vector `|Ax-b|` for the unscaled problem

Integer meaning: To access the lowest index where the maximum primal residual occurs for the unscaled problem

Can use a supplied primal solution.

Macro CPX_MAX_QCPRIMAL_RESIDUAL

Definition file: cplex.h

`CPX_MAX_QCPRIMAL_RESIDUAL`

Concert Technology enum: MaxPrimalResidual.
Numeric meaning (`double`): To access the maximum residual `|x'Qx + dx - f|` over all the quadratic constraints in the unscaled problem.

Integer meaning: To access the lowest index over all the quadratic constraints where the maximum residual occurs in the unscaled problem.

Concert Technology does not distinguish quadratic constraints from linear constraints in this respect.

Macro CPX_MAX_QCSLACK

Definition file: cplex.h

`CPX_MAX_QCSLACK`

Concert Technology enum: not applicable.
Numeric meaning (`double`): To access the maximum absolute quadratic constraint slack value.

Integer meaning: To access the lowest index of the quadratic constraints where the maximum quadratic constraint slack values occcurs.

Can use a supplied primal solution.

Concert Technology does not distinguish quadratic constraints from linear constraints in this respect.

Macro CPX_MAX_QCSLACK_INFEAS

Definition file: cplex.h

`CPX_MAX_QCSLACK_INFEAS`

Concert Technology enum: not applicable.
Numeric meaning (`double`): To access the maximum infeasibility of the quadratic constraints, or equivalently, the maximum bound violation of the quadratic constraint slacks.

Integer meaning: To acces the lowest index of the quadratic constraints where the maximum quadratic slack infeasibility occurs.

Can use a supplied primal solution.

Concert Technology does not distinguish quadratic constraints from linear constraints in this respect.

Macro CPX_MAX_RED_COST

Definition file: cplex.h

CPX_MAX_RED_COST

Concert Technology enum: MaxRedCost.
Numeric meaning (`double`): To access the maximum absolute reduced cost value for the unscaled problem

Integer meaning: To access the lowest index where the maximum reduced cost value occurs for the unscaled problem

Macro CPX_MAX_SCALED_DUAL_INFEAS

Definition file: cplex.h

CPX_MAX_SCALED_DUAL_INFEAS

Concert Technology enum: MaxScaledDualInfeas.
Numeric meaning (`double`): To access the maximum of dual infeasibility or, equivalently, the maximum reduced-cost infeasibility for the scaled problem

Integer meaning: To access the lowest index where the maximum dual infeasibility occurs for the scaled problem

Macro CPX_MAX_SCALED_DUAL_RESIDUAL

Definition file: cplex.h

`CPX_MAX_SCALED_DUAL_RESIDUAL`

Concert Technology enum: MaxScaledDualResidual.
Numeric meaning (`double`): To access maximum dual residual value for the scaled problem

Integer meaning: To access the lowest index where the maximum dual residual occurs for the scaled problem

Macro CPX_MAX_SCALED_PI

Definition file: cplex.h

`CPX_MAX_SCALED_PI`

Concert Technology enum: MaxScaledPi.
Numeric meaning (`double`): To access the maximum absolute value in the dual solution vector for the scaled problem

Integer meaning: To access the lowest index where the maximum pi value occurs for the scaled problem

Macro CPX_MAX_SCALED_PRIMAL_INFEAS

Definition file: cplex.h

`CPX_MAX_SCALED_PRIMAL_INFEAS`

Concert Technology enum: MaxScaledPrimalInfeas.
Numeric meaning (`double`): To access the maximum primal infeasibility or, equivalently, the maximum bound violation including slacks for the scaled problem

Integer meaning: To access the lowest index of a column or row where the maximum primal infeasibility occurs for the scaled problem

Can use a supplied primal solution.

Macro CPX_MAX_SCALED_PRIMAL_RESIDUAL

Definition file: cplex.h

`CPX_MAX_SCALED_PRIMAL_RESIDUAL`

Concert Technology enum: MaxScaledPrimalResidual.
Numeric meaning (`double`): To access the maximum of the vector `|Ax-b|` for the scaled problem

Integer meaning: To access the lowest index where the maximum primal residual occurs for the scaled problem

Can use a supplied primal solution.

Macro CPX_MAX_SCALED_RED_COST

Definition file: cplex.h

CPX_MAX_SCALED_RED_COST

Concert Technology enum: MaxScaledRedCost.
Numeric meaning (`double`): To access the maximum absolute reduced cost value for the scaled problem

Integer meaning: To access the lowest index where the maximum reduced cost value occurs for the scaled problem

Macro CPX_MAX_SCALED_SLACK

Definition file: cplex.h

CPX_MAX_SCALED_SLACK

Concert Technology enum: MaxScaledSlack.
Numeric meaning (`double`): To access the maximum absolute slack value for the scaled problem

Integer meaning: To access the lowest index where the maximum slack value occurs for the scaled problem

Can use a supplied primal solution.

Macro CPX_MAX_SCALED_X

Definition file: cplex.h

`CPX_MAX_SCALED_X`

Concert Technology enum: MaxScaledX.
Numeric meaning (`double`): To access the maximum absolute value in the primal solution vector for the scaled problem

Integer meaning: To access the lowest index where the maximum x value occurs for the scaled problem

Can use a supplied primal solution.

Macro CPX_MAX_SLACK

Definition file: cplex.h

`CPX_MAX_SLACK`

Concert Technology enum: MaxSlack.
Numeric meaning (`double`): To access the maximum absolute slack value for the unscaled problem

Integer meaning: To access the lowest index where the maximum slack value occurs for the unscaled problem

Can use a supplied primal solution.

Macro CPX_MAX_X

Definition file: cplex.h

`CPX_MAX_X`

Concert Technology enum: MaxX.
Numeric meaning (`double`): To access the maximum absolute value in the primal solution vector for the unscaled problem

Integer meaning: To access the lowest index where the maximum x value occurs for the unscaled problem

Can use a supplied primal solution.

Macro CPX_OBJ_GAP

Definition file: cplex.h

CPX_OBJ_GAP

Concert Technology enum: ObjGap.
Numeric meaning (`double`): To access the objective value gap between the primal and dual objective value solution. This feature is available only for a **barrier** solution.

Integer meaning: not applicable

Macro CPX_PRIMAL_OBJ

Definition file: cplex.h

`CPX_PRIMAL_OBJ`

Concert Technology enum: PrimalObj.
Numeric meaning (`double`): To access the objective value relative to the primal barrier solution. This feature is available only for a **barrier** solution.

Integer meaning: not applicable

Macro CPX_SOLNPOOL_DIV

Definition file: cplex.h

`CPX_SOLNPOOL_DIV`

Solution replacement strategy when pool is full: keep most diverse solutions.

Macro CPX_SOLNPOOL_FIFO

Definition file: cplex.h

`CPX_SOLNPOOL_FIFO`

Solution replacement strategy when pool is full: first in first out.

Macro CPX_SOLNPOOL_FILTER_DIVERSITY

Definition file: cplex.h

CPX_SOLNPOOL_FILTER_DIVERSITY

Filter type: filter on diversity. If solution is incompatible with specified diversity measure, it will be discarded from the pool.

Macro CPX_SOLNPOOL_FILTER_RANGE

Definition file: cplex.h

`CPX_SOLNPOOL_FILTER_RANGE`

Filter type: filter on linear expression values. If linear expression of solution does not fall within specified bounds, it will be discarded from the pool.

Macro CPX_SOLNPOOL_OBJ

Definition file: cplex.h

`CPX_SOLNPOOL_OBJ`

Solution replacement strategy when pool is full: keep solutions with best objective value.

Macro CPX_STAT_ABORT_DUAL_OBJ_LIM

Definition file: cplex.h

CPX_STAT_ABORT_DUAL_OBJ_LIM

22 (Barrier only) enum: AbortDualObjLim
Stopped due to a limit on the dual objective

Macro CPX_STAT_ABORT_IT_LIM

Definition file: cplex.h

CPX_STAT_ABORT_IT_LIM

10 (Simplex or Barrier) enum: AbortItLim
Stopped due to limit on number of iterations.

Macro CPX_STAT_ABORT_OBJ_LIM

Definition file: cplex.h

CPX_STAT_ABORT_OBJ_LIM

12 (Simplex or Barrier) enum: AbortObjLim
Stopped due to an objective limit.

Macro CPX_STAT_ABORT_PRIM_OBJ_LIM

Definition file: cplex.h

CPX_STAT_ABORT_PRIM_OBJ_LIM

21 (Barrier only) enum: AbortPrimObjLim
Stopped due to a limit on the primal objective

Macro CPX_STAT_ABORT_TIME_LIM

Definition file: cplex.h

CPX_STAT_ABORT_TIME_LIM

11 (Simplex or Barrier) enum: AbortTimeLim
Stopped due to a time limit.

Macro CPX_STAT_ABORT_USER

Definition file: cplex.h

CPX_STAT_ABORT_USER

13 (Simplex or Barrier) enum: AbortUser
Stopped due to a request from the user.

Macro CPX_STAT_CONFLICT_ABORT_CONTRADICTION

Definition file: cplex.h

CPX_STAT_CONFLICT_ABORT_CONTRADICTION

32 (conflict refiner) enum: ConflictAbortContradiction
The conflict refiner concluded contradictory feasibility for the same set of constraints due to numeric problems. A conflict is available, but it is not minimal.

Macro CPX_STAT_CONFLICT_ABORT_IT_LIM

Definition file: cplex.h

CPX_STAT_CONFLICT_ABORT_IT_LIM

34 (conflict refiner) enum: ConflictAbortItLim
The conflict refiner terminated because of an iteration limit. A conflict is available, but it is not minimal.

Macro CPX_STAT_CONFLICT_ABORT_MEM_LIM

Definition file: cplex.h

CPX_STAT_CONFLICT_ABORT_MEM_LIM

37 (conflict refiner) enum: ConflictAbortMemLim
The conflict refiner terminated because of a memory limit. A conflict is available, but it is not minimal.

Macro CPX_STAT_CONFLICT_ABORT_NODE_LIM

Definition file: cplex.h

CPX_STAT_CONFLICT_ABORT_NODE_LIM

35 (conflict refiner) enum: ConflictAbortNodeLim
The conflict refiner terminated because of a node limit. A conflict is available, but it is not minimal.

Macro CPX_STAT_CONFLICT_ABORT_OBJ_LIM

Definition file: cplex.h

CPX_STAT_CONFLICT_ABORT_OBJ_LIM

36 (conflict refiner) enum: ConflictAbortObjLim
The conflict refiner terminated because of an objective limit. A conflict is available, but it is not minimal.

Macro CPX_STAT_CONFLICT_ABORT_TIME_LIM

Definition file: cplex.h

CPX_STAT_CONFLICT_ABORT_TIME_LIM

33 (conflict refiner) enum: ConflictAbortTimeLim
The conflict refiner terminated because of a time limit. A conflict is available, but it is not minimal.

Macro CPX_STAT_CONFLICT_ABORT_USER

Definition file: cplex.h

CPX_STAT_CONFLICT_ABORT_USER

38 (conflict refiner) enum: ConflictAbortUser
The conflict refiner terminated because a user terminated the application. A conflict is available, but it is not minimal.

Macro CPX_STAT_CONFLICT_FEASIBLE

Definition file: cplex.h

CPX_STAT_CONFLICT_FEASIBLE

30 (conflict refiner) enum: ConflictFeasible
The problem appears to be feasible; no conflict is available.

Macro CPX_STAT_CONFLICT_MINIMAL

Definition file: cplex.h

CPX_STAT_CONFLICT_MINIMAL

31 (conflict refiner) enum: ConflictMinimal
The conflict refiner found a minimal conflict.

Macro CPX_STAT_FEASIBLE

Definition file: cplex.h

CPX_STAT_FEASIBLE

23 (Simplex or Barrier) enum: Feasible
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) on a continuous problem. The problem under consideration was found to be feasible after phase 1 of FeasOpt. A feasible solution is available.

Macro CPX_STAT_FEASIBLE_RELAXED_INF

Definition file: cplex.h

CPX_STAT_FEASIBLE_RELAXED_INF

16 (Simplex or Barrier) enum: FeasibleRelaxedInf
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) with the parameter `CPX_PARAM_FEASOPTMODE` (or `FeasOptMode`) set to `CPX_FEASOPT_MIN_INF` (or `MinInf`) on a continuous problem. A relaxation was successfully found and a feasible solution for the problem (if relaxed according to that relaxation) was installed. The relaxation is minimal.

Macro CPX_STAT_FEASIBLE_RELAXED_QUAD

Definition file: cplex.h

CPX_STAT_FEASIBLE_RELAXED_QUAD

18 (Simplex or Barrier) enum: FeasibleRelaxedQuad
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) with the parameter `CPX_PARAM_FEASOPTMODE` (or `FeasOptMode`) set to `CPX_FEASOPT_MIN_QUAD` (or `MinQuad`) on a continuous problem. A relaxation was successfully found and a feasible solution for the problem (if relaxed according to that relaxation) was installed. The relaxation is minimal.

Macro CPX_STAT_FEASIBLE_RELAXED_SUM

Definition file: cplex.h

CPX_STAT_FEASIBLE_RELAXED_SUM

14 (Simplex or Barrier) enum: FeasibleRelaxedSum
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) with the parameter `CPX_PARAM_FEASOPTMODE` (or `FeasOptMode`) set to `CPX_FEASOPT_MIN_SUM` (or `MinSum`) on a continuous problem. A relaxation was successfully found and a feasible solution for the problem. (if relaxed according to that relaxation) was installed. The relaxation is minimal.

Macro CPX_STAT_INFEASIBLE

Definition file: cplex.h

CPX_STAT_INFEASIBLE

3 (Simplex or Barrier) enum: Infeasible
Problem has been proven infeasible; see the topic *Interpreting Solution Quality* in the *CPLEX User's Manual* for more details.

Macro CPX_STAT_INForUNBD

Definition file: cplex.h

`CPX_STAT_INForUNBD`

4 (Simplex or Barrier) enum: InfOrUnbd
Problem has been proven either infeasible or unbounded; see the topic *Effect of Preprocessing on Feasibility* in the *CPLEX User's Manual* for more detail.

Macro CPX_STAT_NUM_BEST

Definition file: cplex.h

CPX_STAT_NUM_BEST

6 (Simplex or Barrier) enum: NumBest
Solution is available, but not proved optimal, due to numeric difficulties during optimization.

Macro CPX_STAT_OPTIMAL

Definition file: cplex.h

CPX_STAT_OPTIMAL

1 (Simplex or Barrier) enum: Optimal
Optimal solution is available.

Macro CPX_STAT_OPTIMAL_FACE_UNBOUNDED

Definition file: cplex.h

CPX_STAT_OPTIMAL_FACE_UNBOUNDED

20 (Barrier only) enum: OptimalFaceUnbounded
Model has an unbounded optimal face

Macro CPX_STAT_OPTIMAL_INFEAS

Definition file: cplex.h

CPX_STAT_OPTIMAL_INFEAS

5 (Simplex or Barrier) enum: OptimalInfeas
Optimal solution is available, but with infeasibilities after unscaling.

Macro CPX_STAT_OPTIMAL_RELAXED_INF

Definition file: cplex.h

CPX_STAT_OPTIMAL_RELAXED_INF

17 (Simplex or Barrier) enum: OptimalRelaxedInf
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) with the parameter `CPX_PARAM_FEASOPTMODE` (or `FeasOptMode`) set to `CPX_FEASOPT_OPT_INF` (or `OptInf`) on a continuous problem. A relaxation was successfully found and a feasible solution for the problem (if relaxed according to that relaxation) was installed. The relaxation is optimal.

Macro CPX_STAT_OPTIMAL_RELAXED_QUAD

Definition file: cplex.h

CPX_STAT_OPTIMAL_RELAXED_QUAD

19 (Simplex or Barrier) enum: OptimalRelaxedQuad
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) with the parameter `CPX_PARAM_FEASOPTMODE` (or `FeasOptMode`) set to `CPX_FEASOPT_OPT_QUAD` (or `OptQuad`) on a continuous problem. A relaxation was successfully found and a feasible solution for the problem (if relaxed according to that relaxation) was installed. The relaxation is optimal.

Macro CPX_STAT_OPTIMAL_RELAXED_SUM

Definition file: cplex.h

CPX_STAT_OPTIMAL_RELAXED_SUM

15 (Simplex or Barrier) enum: OptimalRelaxedSum
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) with the parameter `CPX_PARAM_FEASOPTMODE` (or `FeasOptMode`) set to `CPX_FEASOPT_OPT_SUM` (or `OptSum`) on a continuous problem. A relaxation was successfully found and a feasible solution for the problem (if relaxed according to that relaxation) was installed. The relaxation is optimal.

Macro CPX_STAT_UNBOUNDED

Definition file: cplex.h

CPX_STAT_UNBOUNDED

2 (Simplex or Barrier) enum: Unbounded
Problem has an unbounded ray; see the concept *Unboundedness* for more information about infeasibility and unboundedness as a solution status.

Macro CPX_SUM_COMP_SLACK

Definition file: cplex.h

CPX_SUM_COMP_SLACK

Concert Technology enum: SumCompSlack.
Numeric meaning (`double`): To access the sum of the violations of the complementary slackness conditions for the unscaled problem. This feature is available only for a **barrier** solution.

Integer meaning: not applicable

Macro CPX_SUM_DUAL_INFEAS

Definition file: cplex.h

`CPX_SUM_DUAL_INFEAS`

Concert Technology enum: SumDualInfeas.
Numeric meaning (`double`): To access the sum of dual infeasibilities or, equivalently, the sum of reduced-cost bound violations for the unscaled problem

Integer meaning: not applicable

Macro CPX_SUM_DUAL_RESIDUAL

Definition file: cplex.h

CPX_SUM_DUAL_RESIDUAL

Concert Technology enum: SumDualResidual.
Numeric meaning (`double`): To access the sum of the absolute values of the dual residual vector for the unscaled problem

Integer meaning: not applicable

Macro CPX_SUM_INDSLACK_INFEAS

Definition file: cplex.h

`CPX_SUM_INDSLACK_INFEAS`

Concert Technology enum: not applicable.
Numeric meaning (`double`): To access the sum of the infeasibilities of the indicator constraints.

Integer meaning: not applicable

Can use a supplied primal solution.

Concert Technology does not distinguish indicator constraints from linear constraints in this respect.

Macro CPX_SUM_INT_INFEAS

Definition file: cplex.h

`CPX_SUM_INT_INFEAS`

Concert Technology enum: SumIntInfeas.
Numeric meaning (`double`): To access the sum of integer infeasibilities for the unscaled problem

Integer meaning: not applicable

Can use a supplied primal solution.

Macro CPX_SUM_PI

Definition file: cplex.h

`CPX_SUM_PI`

Concert Technology enum: SumPi.
Numeric meaning (`double`): To access the sum of the absolute values in the dual solution vector for the unscaled problem

Integer meaning: not applicable

Macro CPX_SUM_PRIMAL_INFEAS

Definition file: cplex.h

CPX_SUM_PRIMAL_INFEAS

Concert Technology enum: SumPrimalInfeas.
Numeric meaning (`double`): To access the sum of primal infeasibilities or, equivalently, the sum of bound violations for the unscaled problem.

Integer meaning: not applicable

Can use a supplied primal solution.

Macro CPX_SUM_PRIMAL_RESIDUAL

Definition file: cplex.h

`CPX_SUM_PRIMAL_RESIDUAL`

Concert Technology enum: SumPrimalResidual.
Numeric meaning (`double`): To access the sum of the elements of vector `|Ax-b|` for the unscaled problem

Integer meaning: not applicable

Can use a supplied primal solution.

Macro CPX_SUM_QCPRIMAL_RESIDUAL

Definition file: cplex.h

`CPX_SUM_QCPRIMAL_RESIDUAL`

Concert Technology enum: SumPrimalResidual.
Numeric meaning (`double`): To access the sum of the residuals `|x'Qx + dx - f|` for the unscaled quadratic constraints.

Integer meaning: not applicable

Concert Technology does not distinguish quadratic constraints from linear constraints in this respect.

Macro CPX_SUM_QCSLACK

Definition file: cplex.h

CPX_SUM_QCSLACK

Concert Technology enum: SumSlack.
Numeric meaning (`double`): To access the sum of the absolute quadratic constraint slack values.

Integer meaning: not applicable

Can use a supplied primal solution.

Concert Technology does not distinguish quadratic constraints from linear constraints in this respect.

Macro CPX_SUM_QCSLACK_INFEAS

Definition file: cplex.h

CPX_SUM_QCSLACK_INFEAS

Concert Technology enum: not applicable.
Numeric meaning (`double`): To access the sum of the infeasibilities of the quadratic constraints.

Integer meaning: not applicable

Can use a supplied primal solution.

Concert Technology does not distinguish quadratic constraints from linear constraints in this respect.

Macro CPX_SUM_RED_COST

Definition file: cplex.h

`CPX_SUM_RED_COST`

Concert Technology enum: SumRedCost.
Numeric meaning (`double`): To access the sum of the absolute reduced cost values for the unscaled problem

Integer meaning: not applicable

Macro CPX_SUM_SCALED_DUAL_INFEAS

Definition file: cplex.h

CPX_SUM_SCALED_DUAL_INFEAS

Concert Technology enum: SumScaledDualInfeas.
Numeric meaning (`double`): To access the sum of dual infeasibilities or, equivalently, the sum of reduced-cost bound violations for the scaled problem

Integer meaning: not applicable

Macro CPX_SUM_SCALED_DUAL_RESIDUAL

Definition file: cplex.h

`CPX_SUM_SCALED_DUAL_RESIDUAL`

Concert Technology enum: SumScaledDualResidual.
Numeric meaning (`double`): To access the sum of the absolute values of the dual residual vector for the scaled problem

Integer meaning: not applicable

Macro CPX_SUM_SCALED_PI

Definition file: cplex.h

CPX_SUM_SCALED_PI

Concert Technology enum: SumScaledPi.
Numeric meaning (`double`): To access the sum of the absolute values in the dual solution vector for the scaled problem

Integer meaning: not applicable

Macro CPX_SUM_SCALED_PRIMAL_INFEAS

Definition file: cplex.h

`CPX_SUM_SCALED_PRIMAL_INFEAS`

Concert Technology enum: SumScaledPrimalInfeas.
Numeric meaning (`double`): To access the sum of primal infeasibilities or, equivalently, the sum of bound violations for the scaled problem

Integer meaning: not applicable

Can use a supplied primal solution.

Macro CPX_SUM_SCALED_PRIMAL_RESIDUAL

Definition file: cplex.h

CPX_SUM_SCALED_PRIMAL_RESIDUAL

Concert Technology enum: SumScaledPrimalResidual.
Numeric meaning (`double`): To access the sum of the elements of vector `|Ax-b|` for the unscaled problem

Integer meaning: not applicable

Can use a supplied primal solution.

Macro CPX_SUM_SCALED_RED_COST

Definition file: cplex.h

CPX_SUM_SCALED_RED_COST

Concert Technology enum: SumScaledRedCost.
Numeric meaning (`double`): To access the sum of the absolute reduced cost values for the unscaled problem

Integer meaning: not applicable

Macro CPX_SUM_SCALED_SLACK

Definition file: cplex.h

CPX_SUM_SCALED_SLACK

Concert Technology enum: SumScaledSlack.
Numeric meaning (`double`): To access the sum of the absolute slack values for the scaled problem

Integer meaning: not applicable

Can use a supplied primal solution.

Macro CPX_SUM_SCALED_X

Definition file: cplex.h

`CPX_SUM_SCALED_X`

Concert Technology enum: SumScaledX.
Numeric meaning (`double`): To access the sum of the absolute values in the primal solution vector for the scaled problem

Integer meaning: not applicable

Can use a supplied primal solution.

Macro CPX_SUM_SLACK

Definition file: cplex.h

`CPX_SUM_SLACK`

Concert Technology enum: SumSlack.
Numeric meaning (`double`): To access the sum of the absolute slack values for the unscaled problem

Integer meaning: not applicable

Can use a supplied primal solution.

Macro CPX_SUM_X

Definition file: cplex.h

`CPX_SUM_X`

Concert Technology enum: SumX.
Numeric meaning (`double`): To access the sum of the absolute values in the primal solution vector for the unscaled problem

Integer meaning: not applicable

Can use a supplied primal solution.

Macro CPXERR_ABORT_STRONGBRANCH

Definition file: cplex.h

CPXERR_ABORT_STRONGBRANCH

1263 Strong branching aborted
Strong branching, for variable selection, could not proceed because a subproblem optimization was aborted.

Macro CPXERR_ADJ_SIGN_QUAD

Definition file: cplex.h

CPXERR_ADJ_SIGN_QUAD

1606 Lines %d,%d: Adjacent sign and quadratic character
The previous line ended with a + or - so the subsequent line must start with a variable name rather than an one of the reserved quadratic characters []*^.

Macro CPXERR_ADJ_SIGN_SENSE

Definition file: cplex.h

CPXERR_ADJ_SIGN_SENSE

1604 Lines %d,%d: Adjacent sign and sense
A sense specifier erroneously follows an arithmetic operator.

Macro CPXERR_ADJ_SIGNS

Definition file: cplex.h

CPXERR_ADJ_SIGNS

1602 Lines %d,%d: Adjacent signs
The previous line ended with a + (plus) or - (minus) so the next line must start with a variable name rather than an operator.

Macro CPXERR_ALGNOTLICENSED

Definition file: cplex.h

CPXERR_ALGNOTLICENSED

32024 Licensing problem: Optimization algorithm not licensed
The license is not configured for this optimization algorithm. For example, this error occurs when anyone tries to invoke the CPLEX Barrier Optimizer with a license key that does not permit this algorithm. Check the options field of the license key to see the CPLEX features that are enabled.

Macro CPXERR_ARC_INDEX_RANGE

Definition file: cplex.h

CPXERR_ARC_INDEX_RANGE

1231 Arc index %d out of range
The specified arc index is negative or greater than or equal to the number of arcs in the network.

Macro CPXERR_ARRAY_BAD_SOS_TYPE

Definition file: cplex.h

CPXERR_ARRAY_BAD_SOS_TYPE

3009 Illegal sostype entry %d
Only sostype values of 1 or 2 are legal.

Macro CPXERR_ARRAY_NOT_ASCENDING

Definition file: cplex.h

CPXERR_ARRAY_NOT_ASCENDING

1226 Array entry %d not ascending
Entries in matbeg or sosbeg arrays must be ascending.

Macro CPXERR_ARRAY_TOO_LONG

Definition file: cplex.h

CPXERR_ARRAY_TOO_LONG

1208 Array length too long
The number of norm values passed to `CPXcopypnorms` exceeds the number of columns, or the number of norm values passed to `CPXcopydnorms` exceeds the number of rows.

Macro CPXERR_BAD_ARGUMENT

Definition file: cplex.h

CPXERR_BAD_ARGUMENT

1003 Bad argument to Callable Library routine.
An invalid argument was passed.

Macro CPXERR_BAD_BOUND_SENSE

Definition file: cplex.h

CPXERR_BAD_BOUND_SENSE

1622 Line %d: Invalid bound sense
An invalid bounds sense marker appears in the LP file. Acceptable bound senses are <, >, =, or free.

Macro CPXERR_BAD_BOUND_TYPE

Definition file: cplex.h

CPXERR_BAD_BOUND_TYPE

1457 Line %d: Unrecognized bound type '%s'
An unrecognized bounds sense specifier appears in the MPS file. Acceptable bound senses are BV, LI, UI, UP, LO, FX, FR, MI, PL, and SC.

Macro CPXERR_BAD_CHAR

Definition file: cplex.h

CPXERR_BAD_CHAR

1537 Illegal character
That character is not allowed. See specifications of the NET or MIN format.

Macro CPXERR_BAD_CTYPE

Definition file: cplex.h

CPXERR_BAD_CTYPE

3021 Illegal ctype entry %d
An illegal `ctype` character has been passed to `CPXchgctype`. Use one of these: C, B, I, S, or N.

Macro CPXERR_BAD_DIRECTION

Definition file: cplex.h

CPXERR_BAD_DIRECTION

3012 Line %d: Unrecognized direction '%c%c'
Only UP and DN are accepted as branching directions beginning in column 2 of an ORD file.

Macro CPXERR_BAD_EXPO_RANGE

Definition file: cplex.h

CPXERR_BAD_EXPO_RANGE

1435 Line %d: Exponent '%s' out of range
An exponent on the specified line is greater than the largest permitted for your computer system.

Macro CPXERR_BAD_EXPONENT

Definition file: cplex.h

CPXERR_BAD_EXPONENT

1618 Line %d: Exponent '%s' not %s with number
The characters following an exponent on the specified line are not numbers.

Macro CPXERR_BAD_FILETYPE

Definition file: cplex.h

CPXERR_BAD_FILETYPE

1424 Invalid filetype
An invalid file type has been passed to a routine requiring a file type.

Macro CPXERR_BAD_ID

Definition file: cplex.h

`CPXERR_BAD_ID`

1617 Line %d: '%s' not valid identifier
An illegal variable or row name exists on the specified line.

Macro CPXERR_BAD_INDCONSTR

Definition file: cplex.h

CPXERR_BAD_INDCONSTR

1439 Line %d: Illegal indicator constraint
Indicator constraints are not allowed in the objective, nor in lazy constraints, nor in user cuts sections. The indicator variable may only be compared against values of 0 (zero) and 1 (one). The MPS format requires that the indicator type be "IF" and that indicator constraints be of type 'E', 'L', or 'G'.

Macro CPXERR_BAD_INDICATOR

Definition file: cplex.h

CPXERR_BAD_INDICATOR

1551 Line %d: Unrecognized basis marker '%s'
An invalid basis marker appears in the BAS file.

Macro CPXERR_BAD_LAZY_UCUT

Definition file: cplex.h

CPXERR_BAD_LAZY_UCUT

1438 Line %d: Illegal lazy constraint or user cut
MPS reader does not allow 'E', 'N', or 'R' in lazy constraints or user cuts.

Macro CPXERR_BAD_LUB

Definition file: cplex.h

CPXERR_BAD_LUB

1229 Illegal bound change specified by entry %d
The bound change specifier must be L, U, or B.

Macro CPXERR_BAD_METHOD

Definition file: cplex.h

CPXERR_BAD_METHOD

1292 Invalid choice of optimization method
Unknown method selected for `CPXhybnetopt` or `CPXhybbaropt`. Select `CPX_ALG_PRIMAL` or `CPX_ALG_DUAL`.

Macro CPXERR_BAD_NUMBER

Definition file: cplex.h

CPXERR_BAD_NUMBER

1434 Line %d: Couldn't convert '%s' to a number
CPLEX was unable to interpret a string as a number on the specified line.

Macro CPXERR_BAD_OBJ_SENSE

Definition file: cplex.h

CPXERR_BAD_OBJ_SENSE

1487 Line %d: Unrecognized objective sense '%s'
There is an OBJSENSE line in an MPS problem file, but CPLEX can not locate the MIN or MAX objective sense statement. Check the MPS file for correct syntax. See the File Formats Manual for a description of MPS format.

Macro CPXERR_BAD_PARAM_NAME

Definition file: cplex.h

CPXERR_BAD_PARAM_NAME

1028 Bad parameter name to CPLEX parameter routine
The parameter name does not exist.

Macro CPXERR_BAD_PARAM_NUM

Definition file: cplex.h

CPXERR_BAD_PARAM_NUM

1013 Bad parameter number to CPLEX parameter routine
The CPLEX parameter number does not exist.

Macro CPXERR_BAD_PIVOT

Definition file: cplex.h

CPXERR_BAD_PIVOT

1267 Illegal pivot
This error occurs if illegal or bad simplex pivots are attempted. Examples are attempts to remove nonbasic variables from the basis or selection of a zero column to enter the basis. Also, this error code may be generated if a pivot would yield a numerically unstable or singular basis.

Macro CPXERR_BAD_PRIORITY

Definition file: cplex.h

`CPXERR_BAD_PRIORITY`

3006 Negative priority entry %d
Priority orders must be positive integer values.

Macro CPXERR_BAD_PROB_TYPE

Definition file: cplex.h

CPXERR_BAD_PROB_TYPE

1022 Unknown problem type. Problem not changed
`CPXchgprobtype` could not change the problem type since an unknown type was specified.

Macro CPXERR_BAD_ROW_ID

Definition file: cplex.h

`CPXERR_BAD_ROW_ID`

1532 Incorrect row identifier
Selected row does not exist.

Macro CPXERR_BAD_SECTION_BOUNDS

Definition file: cplex.h

CPXERR_BAD_SECTION_BOUNDS

1473 Line %d: Unrecognized section marker. Expecting RANGES, BOUNDS, QMATRIX, or ENDATA
An unrecognized MPS file section marker occurred after the COLUMNS section of the MPS file.

Macro CPXERR_BAD_SECTION_ENDATA

Definition file: cplex.h

CPXERR_BAD_SECTION_ENDATA

1462 Line %d: Unrecognized section marker. Expecting ENDATA
An unrecognized MPS file section marker occurred after the COLUMNS section of the MPS file.

Macro CPXERR_BAD_SECTION_QMATRIX

Definition file: cplex.h

CPXERR_BAD_SECTION_QMATRIX

1475 Line %d: Unrecognized section marker. Expecting QMATRIX or ENDATA
An unrecognized MPS file section marker occurred after the RHS or BOUNDS section of the MPS file

Macro CPXERR_BAD_SENSE

Definition file: cplex.h

CPXERR_BAD_SENSE

1215 Illegal sense entry %d
Legal sense symbols are L, G, E, and R.

Macro CPXERR_BAD_SOS_TYPE

Definition file: cplex.h

CPXERR_BAD_SOS_TYPE

1442 Line %d: Unrecognized SOS type: %c%c
Only SOS Types S1 or S2 can be specified within an SOS or MPS file.

Macro CPXERR_BAD_STATUS

Definition file: cplex.h

`CPXERR_BAD_STATUS`

1253 Invalid status entry %d for basis specification
The basis status values are out of range.

Macro CPXERR_BADPRODUCT

Definition file: cplex.h

CPXERR_BADPRODUCT

32023 Licensing problem: License not valid for this product
The license is not configured to support the particular part of the product you are trying to use. For example, this error occurs when when anyone tries to run the Interactive Optimizer with a license configured only for runtime application deployment (for example, ILM key type RTNODE). In such a case, either locate a different ILM key that has been provided to you, or try compiling and running one of the simple examples found in the distribution, such as lpex1.c and its language variants, instead of using the Interactive Optimizer. Contact your support representative for assistance if you are unable to determine the correct ILM key for your purposes.

Macro CPXERR_BAS_FILE_SHORT

Definition file: cplex.h

CPXERR_BAS_FILE_SHORT

1550 Basis missing some basic variables
Number of basic variables is less than the number of rows.

Macro CPXERR_BAS_FILE_SIZE

Definition file: cplex.h

CPXERR_BAS_FILE_SIZE

1555 %d %s basic variable(s)
Number of basic variables doesn't match the problem. Check the `CPXcopybase` call.

Macro CPXERR_CALLBACK

Definition file: cplex.h

`CPXERR_CALLBACK`

1006 Error during callback
An error condition occurred during the callback, as, for example, when a MIP problem is being solved, if a callback asks for information that is not available from CPLEX.

Macro CPXERR_CANT_CLOSE_CHILD

Definition file: cplex.h

CPXERR_CANT_CLOSE_CHILD

1021 Cannot close a child environment
It is not permitted to call `CPXcloseCPLEX` for a child environment.

Macro CPXERR_CHILD_OF_CHILD

Definition file: cplex.h

CPXERR_CHILD_OF_CHILD

1019 Cannot clone a cloned environment
`CPXparenv` cannot be called from a child thread.

Macro CPXERR_COL_INDEX_RANGE

Definition file: cplex.h

CPXERR_COL_INDEX_RANGE

1201 Column index %d out of range
The specified column index is negative or greater than or equal to the number of columns in the currently loaded problem.

Macro CPXERR_COL_REPEAT_PRINT

Definition file: cplex.h

CPXERR_COL_REPEAT_PRINT

1478 %d Column repeats messages not printed
The MPS problem or REV file contains duplicate column entries. Inspect and edit the file.

Macro CPXERR_COL_REPEATS

Definition file: cplex.h

CPXERR_COL_REPEATS

1446 Column '%s' repeats
The MPS file contains duplicate column entries. Inspect and edit the file.

Macro CPXERR_COL_ROW_REPEATS

Definition file: cplex.h

CPXERR_COL_ROW_REPEATS

1443 Column '%s' has repeated row '%s'
The specified column appears more than once in a row. Check the MPS file for duplicate entries.

Macro CPXERR_COL_UNKNOWN

Definition file: cplex.h

CPXERR_COL_UNKNOWN

1449 Line %d: '%s' is not a column name
The MPS file specifies a column name that does not exist.

Macro CPXERR_CONFLICT_UNSTABLE

Definition file: cplex.h

CPXERR_CONFLICT_UNSTABLE

1720 Infeasibility not reproduced.
Computation failed because a previously detected infeasibility could not be reproduced. A conflict exists and can be queried, but it is not minimal.

Macro CPXERR_COUNT_OVERLAP

Definition file: cplex.h

CPXERR_COUNT_OVERLAP

1228 Count entry %d specifies overlapping entries
Entries in the matcnt array are such that the specified items overlap.

Macro CPXERR_COUNT_RANGE

Definition file: cplex.h

CPXERR_COUNT_RANGE

1227 Count entry %d negative or larger than allowed
Entries in matcnt arrays must be nonnegative or less than the number of items possible (columns or rows, for example).

Macro CPXERR_DBL_MAX

Definition file: cplex.h

CPXERR_DBL_MAX

1233 Numeric entry %d is larger than allowed maximum of %g
Data checking detected a number too large.

Macro CPXERR_DECOMPRESSION

Definition file: cplex.h

`CPXERR_DECOMPRESSION`

1027 Decompression of unpresolved problem failed
CPLEX was unable to restore the original problem, due, for example, to insufficient memory.

Macro CPXERR_DUP_ENTRY

Definition file: cplex.h

CPXERR_DUP_ENTRY

1222 Duplicate entry or entries
One or more duplicate entries for a (row, column) pair were found. To identify which pair or pairs caused this error message, use one of the routines in `check.c`.

Macro CPXERR_EXTRA_BV_BOUND

Definition file: cplex.h

CPXERR_EXTRA_BV_BOUND

1456 Line %d: 'BV' bound type illegal when prior bound given
Check the MPS file for bound values which conflict with this type specification.

Macro CPXERR_EXTRA_FR_BOUND

Definition file: cplex.h

CPXERR_EXTRA_FR_BOUND

1455 Line %d: 'FR' bound type illegal when prior bound given
A column with an upper or lower bound previously assigned has an illegal FR bound assignment. Since the FR bound type has neither an upper nor lower bound, no other bound type can be specified. Check the MPS file.

Macro CPXERR_EXTRA_FX_BOUND

Definition file: cplex.h

CPXERR_EXTRA_FX_BOUND

1454 Line %d: 'FX' bound type illegal when prior bound given
A column with either an upper or lower bound previously assigned has an illegal FX bound assignment. Since the FX bound type fixes both upper and lower bounds, no additional bounds can be specified. Check the MPS file.

Macro CPXERR_EXTRA_INTEND

Definition file: cplex.h

CPXERR_EXTRA_INTEND

1481 Line %d: 'INTEND' found while not reading integers
Integer markers are incorrectly positioned in the MPS file.

Macro CPXERR_EXTRA_INTORG

Definition file: cplex.h

CPXERR_EXTRA_INTORG

1480 Line %d: 'INTORG' found while reading integers
Integer markers are incorrectly positioned in the MPS file.

Macro CPXERR_EXTRA_SOSEND

Definition file: cplex.h

CPXERR_EXTRA_SOSEND

1483 Line %d: 'SOSEND' found while not reading a SOS
SOS markers are incorrectly positioned in the MPS file.

Macro CPXERR_EXTRA_SOSORG

Definition file: cplex.h

CPXERR_EXTRA_SOSORG

1482 Line %d: 'SOSORG' found while reading a SOS
SOS markers are incorrectly positioned in the MPS file.

Macro CPXERR_FAIL_OPEN_READ

Definition file: cplex.h

CPXERR_FAIL_OPEN_READ

1423 Could not open file '%s' for reading
CPLEX could not read the specified file. Check the file specification.

Macro CPXERR_FAIL_OPEN_WRITE

Definition file: cplex.h

CPXERR_FAIL_OPEN_WRITE

1422 Could not open file '%s' for writing
CPLEX could not create the specified file. Check the file specification.

Macro CPXERR_FILE_ENTRIES

Definition file: cplex.h

`CPXERR_FILE_ENTRIES`

1553 Line %d: Wrong number of entries
The BAS or VEC or FLT file contains a line with too many or too few entries.

Macro CPXERR_FILE_FORMAT

Definition file: cplex.h

`CPXERR_FILE_FORMAT`

1563 File '%s' has an incompatible format. Try setting reverse flag
When a binary file that has been produced on a different computer system is being read, reversing the setting of the byte order may allow reading.

Macro CPXERR_FILTER_VARIABLE_TYPE

Definition file: cplex.h

CPXERR_FILTER_VARIABLE_TYPE

3414 Diversity filter has non-binary variable(s)
Only binary variables are allowed in diversity filters.

Macro CPXERR_ILL_DEFINED_PWL

Definition file: cplex.h

CPXERR_ILL_DEFINED_PWL

1213
A piecewise linear function has been defined with a discontinuity at the anchor point. Such a definition does not fully specify the piecewise linear function.

Macro CPXERR_ILOG_LICENSE

Definition file: cplex.h

CPXERR_ILOG_LICENSE

32201 ILM Error %d
A licensing error has occurred. Check the environment variable `ILOG_LICENSE_FILE`. For more information, consult the troubleshooting section of the *ILOG License Manager User's Guide and Reference Manual*.

Macro CPXERR_IN_INFOCALLBACK

Definition file: cplex.h

CPXERR_IN_INFOCALLBACK

1804 Calling routines not allowed in informational callback
CPLEX encountered an error in an informational callback, when the user-written callback attempted to invoke a routine other than the routines `CPXgetcallbackinfo` or `CPXgetcallbackincumbent` allowed in informational callbacks.

Macro CPXERR_INDEX_NOT_BASIC

Definition file: cplex.h

CPXERR_INDEX_NOT_BASIC

1251 Index must correspond to a basic variable
The requested variable is not basic.

Macro CPXERR_INDEX_RANGE

Definition file: cplex.h

`CPXERR_INDEX_RANGE`

1200 Index is outside range of valid values
Selected index is too large or small.

Macro CPXERR_INDEX_RANGE_HIGH

Definition file: cplex.h

CPXERR_INDEX_RANGE_HIGH

1206 %s: 'end' value %d is greater than %d
The index in the query routine is too large. The symbol %s represents a string, %d a number.

Macro CPXERR_INDEX_RANGE_LOW

Definition file: cplex.h

CPXERR_INDEX_RANGE_LOW

1205 %s: 'begin' value %d is less than %d
The index in the query routine is too small. The symbol %s represents a string, %d a number.

Macro CPXERR_INT_TOO_BIG

Definition file: cplex.h

CPXERR_INT_TOO_BIG

3018 Magnitude of variable %s: %g exceeds integer limit %d
`CPXmipopt` tried to branch on the specified integer variable at a value larger than representable in the branch-and-cut tree. Check the problem formulation.

Macro CPXERR_INT_TOO_BIG_INPUT

Definition file: cplex.h

CPXERR_INT_TOO_BIG_INPUT

1463 Line %d: Magnitude exceeds integer limit %d
A number has been read that is greater than the largest integer value that can be represented by the computer.

Macro CPXERR_INVALID_NUMBER

Definition file: cplex.h

CPXERR_INVALID_NUMBER

1650 Number not representable in exponential notation
The number to be printed is not representable.

Macro CPXERR_LIMITS_TOO_BIG

Definition file: cplex.h

CPXERR_LIMITS_TOO_BIG

1012 Problem size limits too large
One of the problem dimensions or read limits requires an array length beyond the architectural maximum of the computer.

Macro CPXERR_LINE_TOO_LONG

Definition file: cplex.h

CPXERR_LINE_TOO_LONG

1465 Line %d: Line longer than limit of %d characters
The length of the input line was beyond the size CPLEX can process.

Macro CPXERR_LO_BOUND_REPEATS

Definition file: cplex.h

CPXERR_LO_BOUND_REPEATS

1459 Line %d: Repeated lower bound
The lower bound for a column is repeated within the problem file on the specified line. Two individual lower bounds could exist. Alternatively, an MI bound and individual lower bound could be in conflict. Check the MPS file.

Macro CPXERR_LP_NOT_IN_ENVIRONMENT

Definition file: cplex.h

CPXERR_LP_NOT_IN_ENVIRONMENT

1806 Problem is not member of this environment
CPLEX encountered an error caused by an LP pointer attempting to access an environment other than the environment where the problem problem was created.

Macro CPXERR_MIPSEARCH_WITH_CALLBACKS

Definition file: cplex.h

CPXERR_MIPSEARCH_WITH_CALLBACKS

1805 MIP dynamic search incompatible with control callbacks
CPLEX encountered an error caused by a control callback invoked during dynamic search in MIP optimization.

Macro CPXERR_MISS_SOS_TYPE

Definition file: cplex.h

CPXERR_MISS_SOS_TYPE

3301 Line %d: Missing SOS type
An SOS type has not been specified.

Macro CPXERR_MSG_NO_CHANNEL

Definition file: cplex.h

CPXERR_MSG_NO_CHANNEL

1051 No channel pointer supplied to message routine
The message routine needs a pointer to a channel.

Macro CPXERR_MSG_NO_FILEPTR

Definition file: cplex.h

CPXERR_MSG_NO_FILEPTR

1052 No file pointer found for message routine
The message routine needs a pointer to a file.

Macro CPXERR_MSG_NO_FUNCTION

Definition file: cplex.h

CPXERR_MSG_NO_FUNCTION

1053 No function pointer found for message routine
The message routine needs a pointer to a function.

Macro CPXERR_NAME_CREATION

Definition file: cplex.h

CPXERR_NAME_CREATION

1209 Unable to create default names
The current names of rows or columns don't allow the creation of default names.

Macro CPXERR_NAME_NOT_FOUND

Definition file: cplex.h

CPXERR_NAME_NOT_FOUND

1210 Name not found
Name does not exist. Check the arguments of `CPXgetcolindex` or `CPXgetrowindex`.

Macro CPXERR_NAME_TOO_LONG

Definition file: cplex.h

CPXERR_NAME_TOO_LONG

1464 Line %d: Identifier/name too long to process
The length of the identifier or name was beyond the size CPLEX can process.

Macro CPXERR_NAN

Definition file: cplex.h

CPXERR_NAN

1225 Numeric entry %d is not a double precision number (NAN)
The value is not a number.

Macro CPXERR_NEED_OPT_SOLN

Definition file: cplex.h

CPXERR_NEED_OPT_SOLN

1252 Optimal solution required
An optimal solution must exist before the requested operation can be performed.

Macro CPXERR_NEGATIVE_SURPLUS

Definition file: cplex.h

`CPXERR_NEGATIVE_SURPLUS`

1207 Insufficient array length
The array is too short to hold the requested data.

Macro CPXERR_NET_DATA

Definition file: cplex.h

CPXERR_NET_DATA

1530 Inconsistent network file
Check the NET format file for errors.

Macro CPXERR_NET_FILE_SHORT

Definition file: cplex.h

CPXERR_NET_FILE_SHORT

1538 Unexpected end of network file
Check the NET format file for errors.

Macro CPXERR_NO_BARRIER_SOLN

Definition file: cplex.h

CPXERR_NO_BARRIER_SOLN

1223 No barrier solution exists
The requested operation requires the existence of a barrier solution.

Macro CPXERR_NO_BASIC_SOLN

Definition file: cplex.h

CPXERR_NO_BASIC_SOLN

1261 No basic solution exists
The requested operation requires the existence of a basic solution. Apply primal or dual simplex or crossover.

Macro CPXERR_NO_BASIS

Definition file: cplex.h

`CPXERR_NO_BASIS`

1262 No basis exists
The requested operation requires the existence of a basis.

Macro CPXERR_NO_BOUND_SENSE

Definition file: cplex.h

CPXERR_NO_BOUND_SENSE

1621 Line %d: No bound sense
The sense marker is missing from the specified line.

Macro CPXERR_NO_BOUND_TYPE

Definition file: cplex.h

CPXERR_NO_BOUND_TYPE

1460 Line %d: Bound type missing
No bound type could be found for the specified column bound on the specified line. Check the MPS file.

Macro CPXERR_NO_COLUMNS_SECTION

Definition file: cplex.h

`CPXERR_NO_COLUMNS_SECTION`

1472 Line %d: No COLUMNS section
The required COLUMNS section is missing from the MPS file. Check the file.

Macro CPXERR_NO_CONFLICT

Definition file: cplex.h

CPXERR_NO_CONFLICT

1719 No conflict is available.
Either a conflict has not been computed or the computation failed. For example, computation may fail because the problem is feasible and thus does not contain conflicting constraints.

Macro CPXERR_NO_DUAL_SOLN

Definition file: cplex.h

CPXERR_NO_DUAL_SOLN

1232 No dual solution exists
There is no dual solution available, so there is no quality information about the dual either.

Macro CPXERR_NO_ENDATA

Definition file: cplex.h

`CPXERR_NO_ENDATA`

1552 ENDATA missing
BAS files must have an ENDATA record as the last line of the file.

Macro CPXERR_NO_ENVIRONMENT

Definition file: cplex.h

CPXERR_NO_ENVIRONMENT

1002 No environment
Be sure to pass a valid environment pointer to the routines.

Macro CPXERR_NO_FILENAME

Definition file: cplex.h

`CPXERR_NO_FILENAME`

1421 File name not specified
A filename must be specified for the requested operation to succeed.

Macro CPXERR_NO_ID

Definition file: cplex.h

`CPXERR_NO_ID`

1616 Line %d: Expected identifier, found '%c'
Instead of the expected identifier CPLEX found the character shown in the error message.

Macro CPXERR_NO_ID_FIRST

Definition file: cplex.h

`CPXERR_NO_ID_FIRST`

1609 Line %d: Expected identifier first
A variable name is missing on the specified line.

Macro CPXERR_NO_INT_X

Definition file: cplex.h

`CPXERR_NO_INT_X`

3023 Integer feasible solution values are unavailable
When the incumbent for the problem has been provided by a MIP Start or by an advanced callback function working on the original problem, the incumbent solution values are not available for the reduced problem.

Macro CPXERR_NO_LU_FACTOR

Definition file: cplex.h

CPXERR_NO_LU_FACTOR

1258 No LU factorization exists
The requested item requires the presence of factoring. You may need to optimize with a 0 (zero) iteration limit to factor.

Macro CPXERR_NO_MEMORY

Definition file: cplex.h

CPXERR_NO_MEMORY

1001 Out of memory
The computer has insufficient memory available to complete the selected operation. Downsize problem or increase the amount of physical memory available. Depending on the command, several memory-conserving corrections can be made.

Macro CPXERR_NO_MIPSTART

Definition file: cplex.h

CPXERR_NO_MIPSTART

3020 No MIP start exists
`CPXgetmipstart` failed because no MIP start data is available for the problem.

Macro CPXERR_NO_NAME_SECTION

Definition file: cplex.h

CPXERR_NO_NAME_SECTION

1441 Line %d: No NAME section
The NAME section required in an MPS file is missing.

Macro CPXERR_NO_NAMES

Definition file: cplex.h

CPXERR_NO_NAMES

1219 No names exist
The requested operation is successful only if names have been assigned. Typically, this failure occurs when a file is being read, such as an ORD file, when no names were assigned during the prior call to `CPXreadcopyprob`.

Macro CPXERR_NO_NORMS

Definition file: cplex.h

CPXERR_NO_NORMS

1264 No norms available
Norms are not present. Change pricing, and call the optimization routine.

Macro CPXERR_NO_NUMBER

Definition file: cplex.h

CPXERR_NO_NUMBER

1615 Line %d: Expected number, found '%c'
Some character other than a number, as required, appears on the specified line.

Macro CPXERR_NO_NUMBER_BOUND

Definition file: cplex.h

CPXERR_NO_NUMBER_BOUND

1623 Line %d: Missing bound number
The bound data is missing from the LP file. CPLEX expected a number where no number was found.

Macro CPXERR_NO_NUMBER_FIRST

Definition file: cplex.h

CPXERR_NO_NUMBER_FIRST

1611 Line %d: Expected number first
Some character other than a number, as required, appears on the specified line.

Macro CPXERR_NO_OBJ_SENSE

Definition file: cplex.h

CPXERR_NO_OBJ_SENSE

1436 Max or Min missing
The sense of the objective function (Max maximization or Min minimization) is missing from the LP file. No problem has been read as a consequence.

Macro CPXERR_NO_OBJECTIVE

Definition file: cplex.h

`CPXERR_NO_OBJECTIVE`

1476 Line %d: No objective row found
No free row was found in the MPS file. Check the file. At least one free row must be present. Free rows have an N sense beginning in column 2.

Macro CPXERR_NO_OP_OR_SENSE

Definition file: cplex.h

CPXERR_NO_OP_OR_SENSE

1608 Line %d: Expected '+','-' or sense, found '%c'
Some character other than a + or - operator, as required, appears on the specified line.

Macro CPXERR_NO_OPERATOR

Definition file: cplex.h

CPXERR_NO_OPERATOR

1607 Line %d: Expected '+' or '-', found '%c'
Some character other than + or - appears between variable names on the specified line.

Macro CPXERR_NO_ORDER

Definition file: cplex.h

CPXERR_NO_ORDER

3016 No priority order exists
The requested command cannot be executed because no priority order has been loaded.

Macro CPXERR_NO_PROBLEM

Definition file: cplex.h

`CPXERR_NO_PROBLEM`

1009 No problem exists
The requested command cannot be executed because no problem has been loaded.

Macro CPXERR_NO_QMATRIX_SECTION

Definition file: cplex.h

CPXERR_NO_QMATRIX_SECTION

1461 Line %d: No QMATRIX section
The required QMATRIX section for quadratic programs is missing from the QP file. Check the file.

Macro CPXERR_NO_QP_OPERATOR

Definition file: cplex.h

CPXERR_NO_QP_OPERATOR

1614 Line %d: Expected ^ or *
The ^ or * operator is missing from the QP term.

Macro CPXERR_NO_QUAD_EXP

Definition file: cplex.h

CPXERR_NO_QUAD_EXP

1612 Line %d: Expected quadratic exponent
An exponent of 2 is expected after the ^ operator.

Macro CPXERR_NO_RHS_COEFF

Definition file: cplex.h

CPXERR_NO_RHS_COEFF

1610 Line %d: Expected RHS coefficient
No RHS coefficient is present after the sense marker on the specified line.

Macro CPXERR_NO_RHS_IN_OBJ

Definition file: cplex.h

CPXERR_NO_RHS_IN_OBJ

1211 rhs has no coefficient in obj
You cannot make changes to the righthand side of an objective row because no coefficients exist.

Macro CPXERR_NO_RNGVAL

Definition file: cplex.h

CPXERR_NO_RNGVAL

1216 No range values
No ranges exist for this problem.

Macro CPXERR_NO_ROW_NAME

Definition file: cplex.h

CPXERR_NO_ROW_NAME

1486 Line %d: No row name
A row name is missing within the ROWS section.

Macro CPXERR_NO_ROW_SENSE

Definition file: cplex.h

CPXERR_NO_ROW_SENSE

1453 Line %d: No row sense
No sense for the row was found on the specified line.

Macro CPXERR_NO_ROWS_SECTION

Definition file: cplex.h

`CPXERR_NO_ROWS_SECTION`

1471 Line %d: No ROWS section
No ROW section was found in the MPS file.

Macro CPXERR_NO_SENSIT

Definition file: cplex.h

`CPXERR_NO_SENSIT`

1260 Sensitivity analysis not available for current status
Sensitivity information is not available because an optimal basic solution does not exist for the currently loaded problem. Optimize the problem, and check to make sure that it is not infeasible or unbounded.

Macro CPXERR_NO_SOLN

Definition file: cplex.h

CPXERR_NO_SOLN

1217 No solution exists
The requested command cannot be executed because no solution exists for the problem. Optimize the problem first.

Macro CPXERR_NO_SOLNPOOL

Definition file: cplex.h

CPXERR_NO_SOLNPOOL

3024 No solution pool exists
The requested command cannot be executed because no solution pool exists for the problem. Optimize the problem first. If you have changed the solution pool capacity parameter from its default value, note that it needs to take a positive value for the solution pool to exist.

Macro CPXERR_NO_SOS

Definition file: cplex.h

CPXERR_NO_SOS

3015 No user-defined SOSs exist
SOS information can be written to a file only if the SOS has already been defined. SOS Type 3 information (found by the SOSSCAN feature) cannot be written to an SOS file.

Macro CPXERR_NO_SOS_SEPARATOR

Definition file: cplex.h

CPXERR_NO_SOS_SEPARATOR

1627 Expected ':', found '%c'
The separator :: must follow the S1 or S2 declaration.

Macro CPXERR_NO_TREE

Definition file: cplex.h

`CPXERR_NO_TREE`

3412 Current problem has no tree
No tree exists until after the mixed integer optimization has begun.

Macro CPXERR_NO_VECTOR_SOLN

Definition file: cplex.h

CPXERR_NO_VECTOR_SOLN

1556 Vector solution does not exist
CPLEX could not write VEC file because no vector solution is available.

Macro CPXERR_NODE_INDEX_RANGE

Definition file: cplex.h

CPXERR_NODE_INDEX_RANGE

1230 Node index %d out of range
The specified node index is negative or greater than or equal to the number of nodes in the network.

Macro CPXERR_NODE_ON_DISK

Definition file: cplex.h

`CPXERR_NODE_ON_DISK`

3504 No callback info on disk/compressed nodes
Information about nodes stored in node files is not available through the advanced callback functions.

Macro CPXERR_NOT_DUAL_UNBOUNDED

Definition file: cplex.h

CPXERR_NOT_DUAL_UNBOUNDED

1265 Dual unbounded solution required
The called function requires that the LP stored in the problem object has been determined to be primal infeasible by the dual simplex algorithm.

Macro CPXERR_NOT_FIXED

Definition file: cplex.h

CPXERR_NOT_FIXED

1221 Only fixed variables are pivoted out
`CPXpivotout` can pivot out only fixed variables.

Macro CPXERR_NOT_FOR_MIP

Definition file: cplex.h

CPXERR_NOT_FOR_MIP

1017 Not available for mixed-integer problems
The requested operation can not be performed for mixed integer programs. Change the problem type.

Macro CPXERR_NOT_FOR_QCP

Definition file: cplex.h

CPXERR_NOT_FOR_QCP

1031 Not available for QCP
Function is not available for quadratically constrained problems.

Macro CPXERR_NOT_FOR_QP

Definition file: cplex.h

CPXERR_NOT_FOR_QP

1018 Not available for quadratic programs
The requested operation can not be performed for quadratic programs. Change the problem type.

Macro CPXERR_NOT_MILPCLASS

Definition file: cplex.h

`CPXERR_NOT_MILPCLASS`

1024 Not a MILP or fixed MILP
Function requires that problem type must be CPXPROB_MILP or CPXPROB_FIXEDMILP.

Macro CPXERR_NOT_MIN_COST_FLOW

Definition file: cplex.h

CPXERR_NOT_MIN_COST_FLOW

1531 Not a min-cost flow problem
Check the MIN format file for errors.

Macro CPXERR_NOT_MIP

Definition file: cplex.h

`CPXERR_NOT_MIP`

3003 Not a mixed-integer problem
The requested operation can be performed only on a mixed integer problem.

Macro CPXERR_NOT_MIQPCLASS

Definition file: cplex.h

`CPXERR_NOT_MIQPCLASS`

1029 Not a MIQP or fixed MIQP
Function requires that problem type be CPXPROB_MIQP or CPXPROB_FIXEDMIQP (that is, it has a quadratic objective).

Macro CPXERR_NOT_ONE_PROBLEM

Definition file: cplex.h

CPXERR_NOT_ONE_PROBLEM

1023 Not a single problem
No problem available, or problem is fixed, and the operation is inappropriate for this types of problem.

Macro CPXERR_NOT_QP

Definition file: cplex.h

`CPXERR_NOT_QP`

5004 Not a quadratic program
The requested operation can be performed only on a quadratic problem.

Macro CPXERR_NOT_SAV_FILE

Definition file: cplex.h

`CPXERR_NOT_SAV_FILE`

1560 File '%s' is not a SAV file
The selected file does not match the type specified.

Macro CPXERR_NOT_UNBOUNDED

Definition file: cplex.h

CPXERR_NOT_UNBOUNDED

1254 Unbounded solution required
The requested operation can be performed only on a problem determined to be unbounded.

Macro CPXERR_NULL_NAME

Definition file: cplex.h

CPXERR_NULL_NAME

1224 Null pointer %d in name array
Null pointers are not allowed in name arrays.

Macro CPXERR_NULL_POINTER

Definition file: cplex.h

CPXERR_NULL_POINTER

1004 Null pointer for required data
A value of NULL was passed to a routine where NULL is not allowed.

Macro CPXERR_ORDER_BAD_DIRECTION

Definition file: cplex.h

`CPXERR_ORDER_BAD_DIRECTION`

3007 Illegal direction entry %d
Legal direction entries are limited to the values CPX_BRANCH_GLOBAL, CPX_BRANCH_DOWN, and CPX_BRANCH_UP.

Macro CPXERR_PARAM_INCOMPATIBLE

Definition file: cplex.h

CPXERR_PARAM_INCOMPATIBLE

1807 Incompatible parameters
Incompatible parameters cannot be used together. In particular, `CPX_PARAM_POLISHTIME` cannot be used with any of the `CPX_PARAM_POLISHAFTER...` parameters.

Macro CPXERR_PARAM_TOO_BIG

Definition file: cplex.h

CPXERR_PARAM_TOO_BIG

1015 Parameter value too big
The value of the CPLEX parameter is outside the range of possible settings.

Macro CPXERR_PARAM_TOO_SMALL

Definition file: cplex.h

CPXERR_PARAM_TOO_SMALL

1014 Parameter value too small
The value of the CPLEX parameter is outside the range of possible settings.

Macro CPXERR_PRESLV_ABORT

Definition file: cplex.h

`CPXERR_PRESLV_ABORT`

1106 Aborted during presolve
The user halted preprocessing by means of a callback.

Macro CPXERR_PRESLV_BAD_PARAM

Definition file: cplex.h

CPXERR_PRESLV_BAD_PARAM

1122 Bad presolve parameter setting
Dual presolve reductions (CPX_PARAM_REDUCE) were specified in the presence of lazy constraints, or nonlinear reductions (CPX_PARAM_PRELINEAR) were specified in the presence of user cuts.

Macro CPXERR_PRESLV_BASIS_MEM

Definition file: cplex.h

CPXERR_PRESLV_BASIS_MEM

1107 Not enough memory to build basis for original LP
Insufficient memory exists to complete the uncrushing of the presolved problem.

Macro CPXERR_PRESLV_COPYORDER

Definition file: cplex.h

CPXERR_PRESLV_COPYORDER

1109 Can't copy priority order info from original MIP
The CPLEX call to `CPXcopyorder` failed.

Macro CPXERR_PRESLV_COPYSOS

Definition file: cplex.h

CPXERR_PRESLV_COPYSOS

1108 Can't copy SOS info from original MIP
The CPLEX call to `CPXcopysos` failed.

Macro CPXERR_PRESLV_CRUSHFORM

Definition file: cplex.h

CPXERR_PRESLV_CRUSHFORM

1121 Can't crush solution form
Presolve could not reduce the solution.

Macro CPXERR_PRESLV_DUAL

Definition file: cplex.h

`CPXERR_PRESLV_DUAL`

1119 The feature is not available for solving dual formulation
Certain presolve features are not compatible with its creating an explicit dual formulation.

Macro CPXERR_PRESLV_FAIL_BASIS

Definition file: cplex.h

CPXERR_PRESLV_FAIL_BASIS

1114 Could not load unpresolved basis for original LP
Most likely insufficient memory exists to complete the uncrushing of the presolved problem.

Macro CPXERR_PRESLV_INF

Definition file: cplex.h

`CPXERR_PRESLV_INF`

1117 Presolve determines problem is infeasible
The loaded problem contains blatant infeasibilities.

Macro CPXERR_PRESLV_INForUNBD

Definition file: cplex.h

`CPXERR_PRESLV_INForUNBD`

1101 Presolve determines problem is infeasible or unbounded
The loaded problem contains blatant infeasibilities or unboundedness.

Macro CPXERR_PRESLV_NO_BASIS

Definition file: cplex.h

`CPXERR_PRESLV_NO_BASIS`

1115 Failed to find basis in presolved LP
A basis could not be recovered during uncrushing, most likely due to lack of memory.

Macro CPXERR_PRESLV_NO_PROB

Definition file: cplex.h

CPXERR_PRESLV_NO_PROB

1103 No presolved problem created
Most likely insufficient memory exists to complete the loading of the presolved problem.

Macro CPXERR_PRESLV_SOLN_MIP

Definition file: cplex.h

CPXERR_PRESLV_SOLN_MIP

1110 Not enough memory to recover solution for original MIP
Most likely insufficient memory exists to complete the uncrushing of the presolved problem.

Macro CPXERR_PRESLV_SOLN_QP

Definition file: cplex.h

CPXERR_PRESLV_SOLN_QP

1111 Not enough memory to compute solution to original QP
Most likely insufficient memory exists to complete the uncrushing of the presolved problem.

Macro CPXERR_PRESLV_START_LP

Definition file: cplex.h

`CPXERR_PRESLV_START_LP`

1112 Not enough memory to build start for original LP
Most likely insufficient memory exists to complete the uncrushing of the presolved problem.

Macro CPXERR_PRESLV_TIME_LIM

Definition file: cplex.h

CPXERR_PRESLV_TIME_LIM

1123 Time limit exceeded during presolve
Time limit exceeded during preprocessing.

Macro CPXERR_PRESLV_UNBD

Definition file: cplex.h

`CPXERR_PRESLV_UNBD`

1118 Presolve determines problem is unbounded
The loaded problem contains blatant unboundedness.

Macro CPXERR_PRESLV_UNCRUSHFORM

Definition file: cplex.h

CPXERR_PRESLV_UNCRUSHFORM

1120 Can't uncrush solution form
Presolve could not create a full solution.

Macro CPXERR_PRIIND

Definition file: cplex.h

`CPXERR_PRIIND`

1257 Incorrect usage of pricing indicator
The value of the pricing indicator is out of range.

Macro CPXERR_PRM_DATA

Definition file: cplex.h

CPXERR_PRM_DATA

1660 Line %d: Not enough entries
There were illegal or missing values in a parameter file (.prm).

Macro CPXERR_PRM_HEADER

Definition file: cplex.h

CPXERR_PRM_HEADER

1661 Line %d: Missing or invalid header
Illegal or missing version number in the header of a parameter file (.prm).

Macro CPXERR_PTHREAD_CREATE

Definition file: cplex.h

CPXERR_PTHREAD_CREATE

3603 Could not create thread
An error occurred during a system call needed to initialize parallel MIP.

Macro CPXERR_PTHREAD_MUTEX_INIT

Definition file: cplex.h

CPXERR_PTHREAD_MUTEX_INIT

3601 Could not initialize mutex
An error occurred during a system call needed to initialize parallel MIP.

Macro CPXERR_Q_DIVISOR

Definition file: cplex.h

CPXERR_Q_DIVISOR

1619 Line %d: Missing or incorrect divisor for Q terms
Quadratic terms must be enclosed in square brackets and followed by a division sign with the divisor 2, that is, [ ]/2.

Macro CPXERR_Q_DUP_ENTRY

Definition file: cplex.h

CPXERR_Q_DUP_ENTRY

5011 Duplicate entry for pair '%s' and '%s'
There are duplicate entries for the quadratic term.

Macro CPXERR_Q_NOT_INDEF

Definition file: cplex.h

`CPXERR_Q_NOT_INDEF`

5014 Q is not indefinite
Function requires that the Q matrix be indefinite.

Macro CPXERR_Q_NOT_POS_DEF

Definition file: cplex.h

CPXERR_Q_NOT_POS_DEF

5002 Q in '%s' is not positive semi-definite
The Q matrix associated with the quadratic objective or with a quadratic constraint must be positive semi-definite (for minimizations). Check the appropriate quadratic term(s).

Macro CPXERR_Q_NOT_SYMMETRIC

Definition file: cplex.h

CPXERR_Q_NOT_SYMMETRIC

5012 Q is not symmetric
The Q matrix must be symmetric. Check off-diagonal elements. Look for either a missing or superfluous element.

Macro CPXERR_QCP_SENSE

Definition file: cplex.h

`CPXERR_QCP_SENSE`

6002 Illegal quadratic constraint sense
Legal sense symbols for quadratic constraints are L and G.

Macro CPXERR_QCP_SENSE_FILE

Definition file: cplex.h

CPXERR_QCP_SENSE_FILE

1437 Line %d: Illegal quadratic constraint sense
LP reader does not allow equality in quadratic constraints; MPS file format does not allow 'E', 'N', or 'R' in quadratic constraints.

Macro CPXERR_QUAD_EXP_NOT_2

Definition file: cplex.h

CPXERR_QUAD_EXP_NOT_2

1613 Line %d: Quadratic exponent must be 2
Only an exponent of 2 is allowed after the exponentiation operator ^.

Macro CPXERR_QUAD_IN_ROW

Definition file: cplex.h

CPXERR_QUAD_IN_ROW

1605 Line %d: Illegal quadratic term in a constraint
Quadratic terms are not allowed in indicator constraints, lazy constraints, or user cuts.

Macro CPXERR_RANGE_SECTION_ORDER

Definition file: cplex.h

CPXERR_RANGE_SECTION_ORDER

1474 Line %d: 'RANGES' section out of order
The RANGES section can appear only after the RHS section in an MPS file.

Macro CPXERR_RESTRICTED_VERSION

Definition file: cplex.h

`CPXERR_RESTRICTED_VERSION`

1016 Promotional version. Problem size limits exceeded
The current problem is too large for your version of CPLEX. Reduce the size of the problem.

Macro CPXERR_RHS_IN_OBJ

Definition file: cplex.h

`CPXERR_RHS_IN_OBJ`

1603 Line %d: RHS sense in objective
The objective row erroneously includes a sense specifier.

Macro CPXERR_RIM_REPEATS

Definition file: cplex.h

`CPXERR_RIM_REPEATS`

1447 Line %d: %s '%s' repeats
The MPS file contains duplicate names.

Macro CPXERR_RIM_ROW_REPEATS

Definition file: cplex.h

CPXERR_RIM_ROW_REPEATS

1444 %s '%s' has repeated row '%s'
The MPS file contains duplicate row names.

Macro CPXERR_RIMNZ_REPEATS

Definition file: cplex.h

CPXERR_RIMNZ_REPEATS

1479 Line %d: %s %s repeats
The MPS file contains duplicate entries in an extra rim vector.

Macro CPXERR_ROW_INDEX_RANGE

Definition file: cplex.h

CPXERR_ROW_INDEX_RANGE

1203 Row index %d out of range
The specified row index is negative or greater than or equal to the number of rows in the currently loaded problem.

Macro CPXERR_ROW_REPEAT_PRINT

Definition file: cplex.h

CPXERR_ROW_REPEAT_PRINT

1477 %d Row repeats messages not printed
The MPS problem or REV file contains duplicate row entries. Inspect and edit the file.

Macro CPXERR_ROW_REPEATS

Definition file: cplex.h

CPXERR_ROW_REPEATS

1445 Row '%s' repeats
The MPS file contains duplicate row entries. Inspect and edit the file.

Macro CPXERR_ROW_UNKNOWN

Definition file: cplex.h

CPXERR_ROW_UNKNOWN

1448 Line %d: '%s' is not a row name
The MPS file specifies a row name that does not exist.

Macro CPXERR_SAV_FILE_DATA

Definition file: cplex.h

CPXERR_SAV_FILE_DATA

1561 Not enough data in SAV file
The file is corrupted or was generated by an incompatible version of the software.

Macro CPXERR_SAV_FILE_WRITE

Definition file: cplex.h

CPXERR_SAV_FILE_WRITE

1562 Unable to write SAV file to disk
CPLEX could not open or write to the requested SAV file. Check the file designation and disk space.

Macro CPXERR_SBASE_ILLEGAL

Definition file: cplex.h

CPXERR_SBASE_ILLEGAL

1554 Superbases are not allowed
Basis or restart file contains superbasis that cannot be read.

Macro CPXERR_SBASE_INCOMPAT

Definition file: cplex.h

CPXERR_SBASE_INCOMPAT

1255 Incompatible with superbasis
The requested operation is incompatible with an existing superbasis.

Macro CPXERR_SINGULAR

Definition file: cplex.h

`CPXERR_SINGULAR`

1256 Basis singular
CPLEX cannot factor a singular basis. See the discussion of numeric difficulties in the *CPLEX User's Manual*.

Macro CPXERR_STR_PARAM_TOO_LONG

Definition file: cplex.h

CPXERR_STR_PARAM_TOO_LONG

1026 String parameter is too long
Length of the string was greater than 510.

Macro CPXERR_SUBPROB_SOLVE

Definition file: cplex.h

CPXERR_SUBPROB_SOLVE

3019 Failure to solve MIP subproblem
`CPXmipopt` failed to solve one of the subproblems in the branch-and-cut tree. This failure can be due to a limit (for example, an iteration limit) or due to numeric trouble. Check the log, or add a call to `CPXgetsubstat` in the Callable Library) for information about the cause.

Macro CPXERR_THREAD_FAILED

Definition file: cplex.h

CPXERR_THREAD_FAILED

1234 Creation of parallel thread failed.
Could not create one or more requested parallel threads.

Macro CPXERR_TILIM_CONDITION_NO

Definition file: cplex.h

CPXERR_TILIM_CONDITION_NO

1268 Time limit reached in computing condition number
Condition number computation was not completed due to a time limit.

Macro CPXERR_TILIM_STRONGBRANCH

Definition file: cplex.h

CPXERR_TILIM_STRONGBRANCH

1266 Time limit reached in strong branching
Strong branching was not completed due to a time limit.

Macro CPXERR_TOO_MANY_COEFFS

Definition file: cplex.h

CPXERR_TOO_MANY_COEFFS

1433 Too many coefficients
The problem contains more matrix coefficients than are allowed.

Macro CPXERR_TOO_MANY_COLS

Definition file: cplex.h

```
CPXERR_TOO_MANY_COLS
```

1432 Too many columns
The problem contains more columns than are allowed.

Macro CPXERR_TOO_MANY_RIMNZ

Definition file: cplex.h

CPXERR_TOO_MANY_RIMNZ

1485 Too many rim nonzeros
Reset the rim vector nonzero read limit to a larger number.

Macro CPXERR_TOO_MANY_RIMS

Definition file: cplex.h

CPXERR_TOO_MANY_RIMS

1484 Too many rim vectors
Reset the rim vector read limit to a larger number.

Macro CPXERR_TOO_MANY_ROWS

Definition file: cplex.h

CPXERR_TOO_MANY_ROWS

1431 Too many rows
The problem contains more rows than are allowed.

Macro CPXERR_TOO_MANY_THREADS

Definition file: cplex.h

CPXERR_TOO_MANY_THREADS

1020 Thread limit exceeded
The maximum number of cloned threads has been exceeded.

Macro CPXERR_TREE_MEMORY_LIMIT

Definition file: cplex.h

CPXERR_TREE_MEMORY_LIMIT

3413 Tree memory limit exceeded
The reading of the tree file has stopped because the tree memory limit has been reached.

Macro CPXERR_UNIQUE_WEIGHTS

Definition file: cplex.h

CPXERR_UNIQUE_WEIGHTS

3010 Set does not have unique weights
SOS weights must be unique.

Macro CPXERR_UNSUPPORTED_CONSTRAINT_TYPE

Definition file: cplex.h

CPXERR_UNSUPPORTED_CONSTRAINT_TYPE

1212 Unsupported constraint type was used.
CPLEX was unable to use the specified constraint type, or the constraint type identifier is invalid in a parameter passed to the routine `CPXrefineconflictext` or `CPXfeasoptext`.

Macro CPXERR_UP_BOUND_REPEATS

Definition file: cplex.h

CPXERR_UP_BOUND_REPEATS

1458 Line %d: Repeated upper bound
The upper bound for a column is repeated within the problem file on the specified line. Two individual upper bounds could exist. Alternatively, a PL bound and individual bound could be in conflict. Check the MPS file.

Macro CPXERR_WORK_FILE_OPEN

Definition file: cplex.h

CPXERR_WORK_FILE_OPEN

1801 Could not open temporary file
CPLEX was unable to access a temporary file in the directory specified by `CPX_PARAM_WORKDIR`.

Macro CPXERR_WORK_FILE_READ

Definition file: cplex.h

CPXERR_WORK_FILE_READ

1802 Failure on temporary file read
CPLEX was unable to read a temporary file in the directory specified by `CPX_PARAM_WORKDIR`.

Macro CPXERR_WORK_FILE_WRITE

Definition file: cplex.h

CPXERR_WORK_FILE_WRITE

1803 Failure on temporary file write
CPLEX was unable to write a temporary file in the directory specified by `CPX_PARAM_WORKDIR`.

Macro CPXERR_XMLPARSE

Definition file: cplex.h

CPXERR_XMLPARSE

1425 XML parsing error at line %d: %s
The parser was unable to parse the input file. Additional information about the reason is given in the message.

Macro CPXMIP_ABORT_FEAS

Definition file: cplex.h

`CPXMIP_ABORT_FEAS`

113 (MIP only) enum: AbortFeas
Stopped, but an integer solution exists

Macro CPXMIP_ABORT_INFEAS

Definition file: cplex.h

`CPXMIP_ABORT_INFEAS`

114 (MIP only) enum: AbortInfeas
Stopped; no integer solution

Macro CPXMIP_ABORT_RELAXED

Definition file: cplex.h

CPXMIP_ABORT_RELAXED

126 (MIP only) enum: AbortRelaxed
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`), when the algorithm terminates prematurely, for example after reaching a limit.

This status means that a relaxed solution is available and can be queried.

Macro CPXMIP_FAIL_FEAS

Definition file: cplex.h

`CPXMIP_FAIL_FEAS`

109 (MIP only) enum: FailFeas
Terminated because of an error, but integer solution exists

Macro CPXMIP_FAIL_FEAS_NO_TREE

Definition file: cplex.h

CPXMIP_FAIL_FEAS_NO_TREE

116 (MIP only) enum: FailFeasNoTree
Out of memory, no tree available, integer solution exists

Macro CPXMIP_FAIL_INFEAS

Definition file: cplex.h

`CPXMIP_FAIL_INFEAS`

110 (MIP only) enum: FailInfeas
Terminated because of an error; no integer solution

Macro CPXMIP_FAIL_INFEAS_NO_TREE

Definition file: cplex.h

CPXMIP_FAIL_INFEAS_NO_TREE

117 (MIP only) enum: FailInfeasNoTree
Out of memory, no tree available, no integer solution

Macro CPXMIP_FEASIBLE

Definition file: cplex.h

CPXMIP_FEASIBLE

127 (MIP only) enum: Feasible
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) on a MIP problem. The problem under consideration was found to be feasible after phase 1 of FeasOpt. A feasible solution is available. This status is also used in the status field of solution and mipstart files for solutions from the solution pool.

Macro CPXMIP_FEASIBLE_RELAXED_INF

Definition file: cplex.h

CPXMIP_FEASIBLE_RELAXED_INF

122 (MIP only) enum: FeasibleRelaxedInf
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) with the parameter `CPX_PARAM_FEASOPTMODE` (or `FeasOptMode`) set to `CPX_FEASOPT_MIN_INF` (or `MinInf`) on a mixed integer problem. A relaxation was successfully found and a feasible solution for the problem (if relaxed according to that relaxation) was installed. The relaxation is minimal.

Macro CPXMIP_FEASIBLE_RELAXED_QUAD

Definition file: cplex.h

CPXMIP_FEASIBLE_RELAXED_QUAD

124 (MIP only) enum: FeasibleRelaxedQuad
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) with the parameter `CPX_PARAM_FEASOPTMODE` (or `FeasOptMode`) set to `CPX_FEASOPT_MIN_QUAD` (or `MinQuad`) on a mixed integer problem. A relaxation was successfully found and a feasible solution for the problem (if relaxed according to that relaxation) was installed. The relaxation is minimal.

Macro CPXMIP_FEASIBLE_RELAXED_SUM

Definition file: cplex.h

CPXMIP_FEASIBLE_RELAXED_SUM

120 (MIP only) enum: FeasibleRelaxedSum
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) with the parameter `CPX_PARAM_FEASOPTMODE` (or `FeasOptMode`) set to `CPX_FEASOPT_MIN_SUM` (or `MinSum`) on a mixed integer problem. A relaxation was successfully found and a feasible solution for the problem (if relaxed according to that relaxation) was installed. The relaxation is minimal.

Macro CPXMIP_INFEASIBLE

Definition file: cplex.h

`CPXMIP_INFEASIBLE`

103 (MIP only) enum: Infeasible
Solution is integer infeasible

Macro CPXMIP_INForUNBD

Definition file: cplex.h

`CPXMIP_INForUNBD`

119 (MIP only) enum: InfOrUnbd
Problem has been proved either infeasible or unbounded

Macro CPXMIP_MEM_LIM_FEAS

Definition file: cplex.h

`CPXMIP_MEM_LIM_FEAS`

111 (MIP only) enum: MemLimFeas
Limit on tree memory has been reached, but an integer solution exists

Macro CPXMIP_MEM_LIM_INFEAS

Definition file: cplex.h

CPXMIP_MEM_LIM_INFEAS

112 (MIP only) enum: MemLimInfeas
Limit on tree memory has been reached; no integer solution

Macro CPXMIP_NODE_LIM_FEAS

Definition file: cplex.h

CPXMIP_NODE_LIM_FEAS

105 (MIP only) enum: NodeLimFeas
Node limit has been exceeded but integer solution exists

Macro CPXMIP_NODE_LIM_INFEAS

Definition file: cplex.h

CPXMIP_NODE_LIM_INFEAS

106 (MIP only) enum: NodeLimInfeas
Node limit has been reached; no integer solution

Macro CPXMIP_OPTIMAL

Definition file: cplex.h

`CPXMIP_OPTIMAL`

101 (MIP only) enum: Optimal
Optimal integer solution has been found

Macro CPXMIP_OPTIMAL_INFEAS

Definition file: cplex.h

CPXMIP_OPTIMAL_INFEAS

115 (MIP only) enum: OptimalInfeas
Problem is optimal with unscaled infeasibilities

Macro CPXMIP_OPTIMAL_POPULATED

Definition file: cplex.h

CPXMIP_OPTIMAL_POPULATED

129 (MIP only) enum: OptimalPopulated
This status occurs only after a call to the Callable Library routine `CPXpopulate` (or the Concert Technology method `populate`) on a MIP problem. Populate has completed the enumeration of all solutions it could enumerate.

Macro CPXMIP_OPTIMAL_POPULATED_TOL

Definition file: cplex.h

CPXMIP_OPTIMAL_POPULATED_TOL

130 (MIP only) enum: OptimalPopulatedTol
This status occurs only after a call to the Callable Library routine `CPXpopulate` (or the Concert Technology method `populate`) on a MIP problem. Populate has completed the enumeration of all solutions it could enumerate whose objective value fit the tolerance specified by the parameters `CPX_PARAM_SOLNPOOLAGAP` and `CPX_PARAM_SOLNPOOLGAP`.

Macro CPXMIP_OPTIMAL_RELAXED_INF

Definition file: cplex.h

CPXMIP_OPTIMAL_RELAXED_INF

123 (MIP only) enum: OptimalRelaxedInf
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) with the parameter `CPX_PARAM_FEASOPTMODE` (or `FeasOptMode`) set to `CPX_FEASOPT_OPT_INF` (or `OptInf`) on a mixed integer problem. A relaxation was successfully found and a feasible solution for the problem (if relaxed according to that relaxation) was installed. The relaxation is optimal.

Macro CPXMIP_OPTIMAL_RELAXED_QUAD

Definition file: cplex.h

CPXMIP_OPTIMAL_RELAXED_QUAD

125 (MIP only) enum: OptimalRelaxedQuad
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) with the parameter `CPX_PARAM_FEASOPTMODE` (or `FeasOptMode`) set to `CPX_FEASOPT_OPT_QUAD` (or `OptQuad`) on a mixed integer problem. A relaxation was successfully found and a feasible solution for the problem (if relaxed according to that relaxation) was installed. The relaxation is optimal.

Macro CPXMIP_OPTIMAL_RELAXED_SUM

Definition file: cplex.h

CPXMIP_OPTIMAL_RELAXED_SUM

121 (MIP only) enum: OptimalRelaxedSum
This status occurs only after a call to the Callable Library routine `CPXfeasopt` (or the Concert Technology method `feasOpt`) with the parameter `CPX_PARAM_FEASOPTMODE` (or `FeasOptMode`) set to `CPX_FEASOPT_OPT_SUM` (or `OptSum`) on a mixed integer problem. A relaxation was successfully found and a feasible solution for the problem (if relaxed according to that relaxation) was installed. The relaxation is optimal.

Macro CPXMIP_OPTIMAL_TOL

Definition file: cplex.h

CPXMIP_OPTIMAL_TOL

102 (MIP only) enum: OptimalTol
Optimal solution with the tolerance defined by epgap or epagap has been found

Macro CPXMIP_POPULATESOL_LIM

Definition file: cplex.h

CPXMIP_POPULATESOL_LIM

128 (MIP only) enum: PopulateSolLim
This status occurs only after a call to the Callable Library routine `CPXpopulate` (or the Concert Technology method `populate`) on a MIP problem. The limit on mixed integer solutions generated by populate, as specified by the parameter `CPX_PARAM_POPULATELIM`, has been reached.

Macro CPXMIP_SOL_LIM

Definition file: cplex.h

`CPXMIP_SOL_LIM`

104 (MIP only) enum: SolLim
The limit on mixed integer solutions has been reached

Macro CPXMIP_TIME_LIM_FEAS

Definition file: cplex.h

CPXMIP_TIME_LIM_FEAS

107 (MIP only) enum: TimeLimFeas
Time limit exceeded, but integer solution exists

Macro CPXMIP_TIME_LIM_INFEAS

Definition file: cplex.h

CPXMIP_TIME_LIM_INFEAS

108 (MIP only) enum: TimeLimInfeas
Time limit exceeded; no integer solution

Macro CPXMIP_UNBOUNDED

Definition file: cplex.h

`CPXMIP_UNBOUNDED`

118 (MIP only) enum: Unbounded
Problem has an unbounded ray